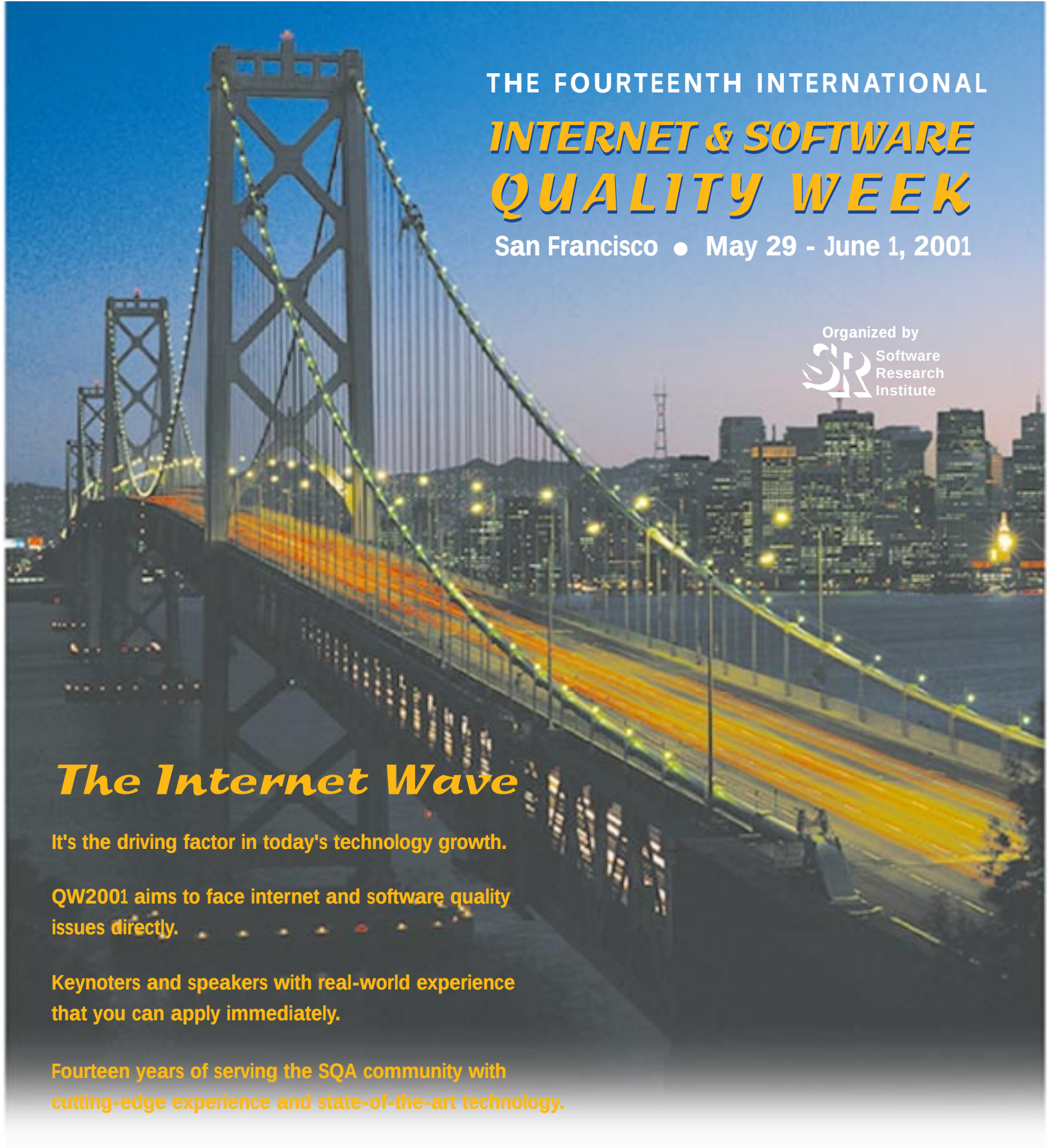


THE FOURTEENTH INTERNATIONAL

INTERNET & SOFTWARE QUALITY WEEK

San Francisco • May 29 - June 1, 2001

Organized by
Software Research Institute

The Internet Wave

It's the driving factor in today's technology growth.

QW2001 aims to face internet and software quality issues directly.

Keynoters and speakers with real-world experience that you can apply immediately.

Fourteen years of serving the SQA community with cutting-edge experience and state-of-the-art technology.

In cooperation with
ASQ Software Division

Industry Sponsors
SOFTWARE RESEARCH
COMPUWARE
eValid
TESTING LABS
VANTEON
VeriTest

Media Sponsors
Application Development Trends
C/C++ Users Journal
Dr. Dobb's Journal
SD Times
Software Business
Windows Developer's Journal

Quality Week 2001 Program

NOTE: The program selections, speaker biographies, and presentation abstracts may be incomplete in some instances; this material is being updated constantly. All material presented is based on the best information available and may be subject to change. Updated 23 February 2001.

QUICK ACCESS TO THE FOUR DAY PROGRAM AT QW2001	
Pre-Conference Tutorials	Tuesday 29 May 2001
Conference Day 1	Wednesday 30 May 2001
Conference Day 2	Thursday 31 May 2001
Conference Day 3	Friday 1 June 2001
Post Conference Workshops	Friday 1 June 2001
Vendor Demonstration Sessions	Wednesday 30 May 2001 Thursday 31 May 2001

The QW2001 program is linked to speaker biographies and presentation abstracts. Click on a ***Speaker*** or on a ***Title*** to read the Presentation Summary and Author Biography. Paper descriptions will appear in a separate popup window.

REGISTER FOR QW2001

[BACK TO TOP]

T — Tuesday 29 May 2001 PRE-CONFERENCE TUTORIALS

	Tutorial A1	Tutorial B1	Tutorial C1	Tutorial D1	Tutorial E1	Tutorial F1	Tutorial G1
8:30 - 12:00	Mr. Tom Gilb (Result Planning Limited) *Software Inspection For The Internet Age: How To Increase Effect And Radically Reduce the Cost* (Part I)	Mr. Erik Simmons (Intel Corporation) *Writing Good Requirements*	Dr. Norman F. Schneidewind (Naval Postgraduate School) *A Roadmap To Distributed Client-Server Software Reliability Engineering*	Dr. Gualtiero Bazzana (ONION, S.P.A) *Web Testing Techniques and Tools* (Part I)	Mr. Robert A. Sabourin (AmiBug.Com) *Getting Started -- Stressing Web Applications: Stress Early -- Stress Often*	Mr. Ross Collard (Collard and Co.) *Test Estimating*	Mr. Thomas Drake (Integrated Computer Concepts, Inc ICCI) *The Quality Challenge For Network Based Software Systems*
12:00 - 1:30	TUTORIAL DAY LUNCH AND NETWORKING						
	Tutorial A2	Tutorial B2	Tutorial C2	Tutorial D2	Tutorial E2	Tutorial F2	Tutorial G2
1:30 - 5:00	Mr. Tom Gilb (Result Planning Limited) *Software Inspection For The Internet Age: How To Increase Effect And Radically Reduce the Cost* (Part II)	Mr. Bill Deibler (Software Systems Quality Consulting) *Making the CMM Work: Streamlining the CMM for Todays Projects and Organizations*	Dr. John D. Musa (Consultant) *More Reliable Software Faster And Cheaper*	Dr. Gualtiero Bazzana (ONION, S.P.A) *Web Testing Techniques and Tools* (Part II)	Dr. Edward Miller (eValid) *Client-Side WebSite Testing*	Dr. Cem Kaner (Florida Institute of Technology) *Teaching Testing: A Skills-Based Approach*	Mr. Ed Kit (SDT Corporation) *Establishing a Fully Integrated Test Automation Architecture*
5:00 - 6:00	Welcome Networking Reception						

[BACK TO TOP]

Wednesday 30 May 2001 CONFERENCE DAY #1

QW2001 Exhibition: 10:00 AM to 6:00 PM

1 — 8:30 - 10:00 — PLENARY SESSION

Plenary Session Introduction: Edward Miller (Software Research, Inc.)

Keynote 1P1: Mr. Ed Kit (SDT Corporation) Test Automation -- State of the Practice

Keynote 1P2: Mr. Hans Buwalda (CMG) The Three "Holy Grails" of Test Development (...adventures of a mortal tester..)

10:00 - 10:30 — REFRESHMENTS IN EXHIBIT HALL

2 — 10:30 - 12:00

Vendor Technical Track	Technology Track: Test Automation	Applications Track: Using Models	Internet Track: Web-Oriented Process	Management Track: Requirements	QuickStart Track
Session 2V1 Mr. Steve Nemzer (VeriTest) *Testing In Multi-CPU Environment*	**Paper 2T1** Mr. Tobias Mayer, Mr. Thomas Stocking (eValid, Inc., siteROCK) *The Web Site Testing Challenge*	**Paper 2A1** Mr. James Tierney (Microsoft) *Getting Started With Model-Based Testing*	**Paper 2W1** Mr. Adrian Cowderoy (Professional-Spirit Ltd) *Quality in a Dotcom Startup -- Fact or Fiction?*	**Paper 2M1** Mr. Scott Jefferies (Technology Builders, Inc.) *A Requirements-Based Approach To Delivering E-Business And Enterprise Applications*	**Session 2Q** Mr. Kent Beck (Author) *Extreme Programming Explained*
Session 2V2 Mr. Francois Charette (TestQuest) *Testing Configurable Product Platforms*	**Paper 2T2** Ms. Nancy Eickelmann & Mr. Allan Willey (Motorola Labs) *An Integrated System Test Environment*	**Paper 2A2** Mr. Klaus Olsen (Software-test.dk) *Using The W Model To Institutionalize Inspections, And Improve Knowledge Transfers*	**Paper 2W2** Mr. Todd Hsueh (IBM) *Innovative Web Test Process & Control Tool*	**Paper 2M2** Mr. Robert Benjamin, Ms. Ruth Pennoyer & Ms. Karen Law (Spherion Corporation) *Pre-Defining Success: Incorporating e-Metrics Into Business And Technical Requirements For Web And e-Business Solutions*	

12:00 - 1:30 — CONFERENCE LUNCH AND NETWORKING IN EXHIBIT HALL

3 — 1:30 - 3:00

Vendor Technical Track	Technology Track: Process Models	Applications Track: Outsourcing	Internet Track: Performance & Load	Management Track: New Methods	QuickStart Track
Session 3V1 Mr. Christian Hote (PolySpace Technologies) *Bug Detection Tools for Productivity*	**Paper 3T1** Mrs. Manjula Madan (Philips Software Centre, Bangalore) *Defect Reduction Using Orthogonal Defect Classification Methodology*	**Paper 3A1** Mr. Ralph Dalebout (IBM Printing Systems Division) *Beta Testing With Rapid Development*	**Paper 3W1** Ms. Nancy Landau (Alltel Technology Services) *Performance Testing Applications In Internet Time*	**Paper 3M1** Mr. Timothy Kelliher, Dr. Daniel Blezek, Mr. William Lorensen & Mr. James Miller (GE Research & Development) *Six-Sigma Meets Extreme Programming: Changing the Way We Work*	**Session 3Q** Mr. Tom Gilb (Result Planning Limited) *Planguage: A Defined Language for Clearer Requirements and Design*
Session 3V2 Ms. Peggy Fouts (Compuware) *Test Planning For Xtreme Times*	**Paper 3T2** Dr. Mark R. Blackburn, Mr. Robert Busser, Mr. Aaron Nauman & Dr. Ramaswamy Chandramouli (Software Productivity Consortium) *Model-based Approach To Security Test Automation*	**Paper 3A2** Mr. Roger M. Records (Boeing Commercial Airplane Group) *Assuring Quality In Outsourced Software*	**Paper 3W2** Mr. Steve Splaine (Splaine & Associates) *Modeling The Real World For Load Testing Web Sites*	**Paper 3M2** Ms. Elli Georgiadou (Middlesex University) & Naomi Barber (University of North London) *Investigating The Applicability Of The Taguchi Method To Software Development*	

3:00 - 3:30 — REFRESHMENTS IN EXHIBIT HALL

4 — 3:30 - 5:00

Vendor Technical Track	Technology Track: Metrics	Applications Track: Reliability	Internet Track: Test Practices	Management Track: Life Cycle Issues	QuickStart Track
Session 4V1 Mr. John Keller (TeamShare, Inc.) *Collaborative Product Design and the New Role of Testing*	**Paper 4T1** Prof. Warren Harrison (Portland State University) *A Universal Metrics Repository*	**Paper 4A1** Mr. Henk Keesom & Dr. John Musa (Ortho-Clinical Diagnostics) *Using Test Data To Calculate Software Reliability Growth*	**Paper 4W1** Mr. Nikhil Nilakantan & Mr. Ibrahim K. El-Far (Florida Tech) *Why Is API Testing Different*	**Paper 4M1** Mr. David Fern (Micros Systems Inc.) *How Testers Can And Should Drive Development Cycles*	**Session 4Q** Mr. Greg Clower (SDT Corporation) *Establishing a Telecommunication Test Automation System*
Session 4V2 Mr. Michael Smith (McCabe&Assoc)	**Paper 4T2** Mr. Suresh Nageswaran (Cognizant Technology Solutions CTS) *Test Effort Estimation Using Use Case Points (UCP)*	**Paper 4A2** Mr. Erik Simmons (Intel Corporation) *Product Triage: A "Medical" Approach To Predicting And Monitoring Product*	**Paper 4W2** Mr. Phil Hollows (RadView Software) *Best Practices in Web Performance Testing*	**Paper 4M2** Dr. Cem Kaner (Florida Institute of Technology) *Managing The Proportion Of Testers To Developers*	

5:00 - 6:00 — Mr. Brian Lawrence, Panel 4P Chair — Special Panel Session: *How DO You Test Internet Software?*

5:00 - 6:30 — EXPO RECEPTION — Drinks and hors d'oeuvres are served in the Expo Hall.

7:15 — QW2001 Special Event! Arizona Diamondbacks VS. San Francisco Giants at PacBell Park

[BACK TO TOP]

Thursday 31 May 2001 CONFERENCE DAY #2

QW2001 Exhibition: 10:00 AM to 6:00 PM

5 — 8:30 - 10:00 — PLENARY SESSION

Plenary Session Introduction: Edward Miller (Software Research, Inc.)

Keynote 5P1: Dr. Linda Rosenberg (GSFC NASA) Independent Verification And Validation Implementation At NASA

Keynote 5P2: Dr. Dalibor Vrsalovic (Intel Corporation) Issues in Design and Validation of Modern eBusiness Systems

10:00 - 10:30 — REFRESHMENTS IN EXHIBIT HALL

6 — 10:30 - 12:00

Vendor Technical Track	Technology Track: Advanced Automation	Applications Track: Field Reports	Internet Track: Complexity Estimation	Management Track: Release Criteria	QuickStart Track
Session 6V1 Ms. Lauri MacKinnon (Vanteon) *Three Customer Case Studies of Performance Testing using Segue SilkPerformer*	**Paper 6T1** Dr. Rainer Stetter (ITQ Gmbh & Software Factory Gmbh) *Test Strategies for Embedded Systems*	**Paper 6A1** Mr. Steve Whitchurch (Mentor Graphics Corp.) *Trials and Tribulations Of Testing a Java/C++ Hybrid Application*	**Paper 6W1** Mr. Mark Johnson (Cadence Design Systems) *How Are You Going To Test All Those Configurations?*	**Paper 6M1** Mr. Geert Pinxten (I2B) *The Extended Product Quality Model*	**Session 6Q** Ms. Jeanette Folkes (Modem Media) *WebTesting 101*
Session 6V2 Mr. Larry Markesian (Reasoning) *Improving Software Quality & Delivery Schedules Through Automated Inspection*	**Paper 6T2** Mr. Keith B. Stobie (BEA Systems, Inc.) *Automating Test Oracles and Decomposability*	**Paper 6A2** Mr. Juris Borzovs & Mr. Martins Gills (Riga Information Technology Inst.) *Software Testing in Latvia: Lessons Learned*	**Paper 6W2** Mr. Rakesh Agarwal, Mr. Santanu Banerjee & Mr. Bhaskar Gosh (Infosys Technologies Ltd) *Estimating Internet Based Projects: A Case Study*	**Paper 6M2** Ms. Johanna Rothman (The Rothman Consulting Group) *Using Requirements To Create Release Criteria*	

12:00 - 1:30 — CONFERENCE LUNCH AND NETWORKING IN EXHIBIT HALL

7 — 1:30 - 3:00

Vendor Technical Track	Technology Track: Functional Testing	Applications Track: Protocol Issues	Internet Track: Web Lifecycles	Management Track: Review Techniques	QuickStart Track
Session 7V1 Mr. Rick Banister (Sesame Technology) *Clothing Optional Relationships With Your Customers--How much should we expose to our customers when it comes to the product improvement process?*	**Paper 7T1** Mr. Don Cohen (Princeton Softech) *Requirements For A Comprehensive Testing Environment*	**Paper 7A1** Dr. Holger Schlingloff & Dr. Jan Bredereke (Technologie-Zentrum Informatik) *Specification Based Testing Of the UMTS Protocol Stack*	**Paper 7W1** Mr. Bhushan Gupta & Mr. Steve Rhodes (Hewlett-Packard Co.) *Adopting A Lifecycle For Developing Web Based Applications*	**Paper 7M1** Mr. Michael Ensminger (PAR3 Communications) *Walk & Stagger Through Review Process*	**Session 7Q** Mr. James Bach (Satisfice Inc) *High Accountability Exploratory Testing*
Session 7V2 Dr. Edward Miller (eValid, Inc.) *Universal Client-site Web Test Engine*	**Paper 7T2** Mr. James Lyndsay (Workroom Productions) *The Importance of Data In Functional Testing*	**Paper 7A2** Mr. Michael K. Jones (Dromedary Peak Consulting/Western International University) *High Availability Testing*	**Paper 7W2** Mr. Eric Patel (Nokia Home Communications) *Rapid SQA: Web Testing At The Speed Of The Internet*	**Paper 7M2** Prof. Warren Harrison, Dr. David Raffo & Dr. John Settle (Portland State University) *Process Improvement As A Capital Investment*	

3:00 - 3:30 — REFRESHMENTS IN EXHIBIT HALL

8 — 3:30 - 5:00

Vendor Technical Track	Technology Track: Novel Approaches	Applications Track: Low-Level Testing	Internet Track: Bug-Based Methods	Management Track: Risk-Based Methods	QuickStart Track
Session 8V1 Mr. Steve Smith (QualityLogic) *Develop Great Stuff, Repeatedly, On Time*	**Paper 8T1** Mr. J. D. Brisk (Exodus Communications) *Peer-to-Peer Computing: The Future Of Internet Performance Testing*	**Paper 8A1** Mr. Vince Budrovich (ParaSoft Corporation) *Increasing The Effectiveness Of Load Testing: Unit-Level Load Testing*	**Paper 8W1** Dr. James Helm (Univ. of Houston Clear Lake) *Web-Based Application Quality Assurance Testing*	**Paper 8M1** Ms. Sandy Sweeney (Compuware Corporation) *Risky Business -- Adding Risk Assessment To The Test Planning Process*	**Session 8Q** Mr. Robert A. Sabourin (AmiBug.Com) *The Effective SQA Manager: Getting Things Done*
Session 8V2 Mr. Shel Siegel (Tescom) *Component Based Architecture for Automated Testing*	**Paper 8T2** Mr. Erik Simmons (Intel Corporation) *Quantifying Quality Requirements using Planguage*	**Paper 8A2** Mr. Scott Trappe (Reasoning, Inc.) *Building Better Software Code: Finding Bugs You Never Knew You Had*	**Paper 8W2** Mr. Kim Davis & Mr. Robert Sabourin (My Virtual Model Inc.) *Exploring, Discovering and Exterminating Bug Clusters In Web Applications*	**Paper 8M2** Mr. Kamesh Pemmaraju (Cigital, Inc.) *Software Risk Management*	

5:00 - 6:00 — Nick Borelli, Panel 8P Chair — Special Panel Session: *ASK THE QUALITY EXPERTS!* Stump the Quality Experts If You Can! Post Your Questions LIVE on the Web! QW2001 Advisory Board Members Will Answer All Questions! (In Cooperation With Microsoft Corporation)

[BACK TO TOP]

Friday 1 June 2001 CONFERENCE DAY #3

9 — 8:30 - 10:00

Vendor Technical Track	Technology Track: Advanced Tools	Applications Track: Embedded Systems	Internet Track: Real-World Studies	Management Track: Organizational Issues	Panel Session
Session 9V1	**Paper 9T1** Mr. Greg Berger (Lawson Software) *Creating A Tool-Independent Test Environment*	**Paper 9A1** Dr. Harmen Sthamer & Mr. Joachim Wegener (DaimlerChrysler AG) *Evolutionary Testing Of Embedded Systems*	**Paper 9W1** Ms. Patricia D. Humphrey (Neoforma.com) *Quality Assurance and the Internet Site - How To Effectively Hit a Moving Target*	**Paper 9M1** Mr. Michael J. Hillelsohn (Software Performance Systems) *Organizational Performance Engineering: Quality Assurance For The 21st Century*	**Session 9Q** Dr. John D. Musa - Panel Chair (Consultant) *How Will The Internet Affect Software Quality Practice?*
Session 9V2	**Paper 9T2** Mr. Timothy Kelliher, Dr. Daniel Blezek, Mr. William Lorensen & Dr. James Miller (GE Corporate R&D) *The Frost Extreme Testing Framework*	**Paper 9A2** Mr. Hung Q. Nguyen (LogiGear Technology) *The Design and Implementation of a Flexible, Reusable and Maintainable Automation Framework*	**Paper 9W2** Mr. Alexey Kerov (Amphora Quality Technologies) Iterative Approach as Basis For Effective Testing	**Paper 9M2** Mr. Brian Lawrence (Coyote Valley Software) *Choosing Potential Improvements -- Comparing Approaches*	

10:00 - 10:30 — REFRESHMENTS

10 — 10:30 - 12:30 — PLENARY SESSION

Plenary Session Introduction: Edward Miller (Software Research, Inc.)

Keynote 10P1: Ms. Lisa Crispin (Tensegrent) The Need For Speed: Acceptance Test Automation in an eXtreme Programming Environment (QW2000 Best Presentation)

Keynote 10P2: Dave Lilly (siteROCK Corporation) Internet Quality of Service (QoS): The State Of The Practice

Keynote 10P3: Mr. Thomas Drake (Integrated Computer Concepts, Inc ICCI) Riding The Wave -- The Future For Software Quality

AWARDS PRESENTATION

1:00 - 2:00 — CONFERENCE LUNCH

[BACK TO TOP]

Friday 1 June 2001 POST-CONFERENCE WORKSHOPS

W 2:00 - 5:00	Technology Workshop W1	Applications Workshop W2	Internet Workshop W3	Management Workshop W4
	Dr. Cem Kaner (Florida Institute of Technology) *Developing The Right Test Documentation*	Mr. John Paul (Minjoh Technology Solutions) *Automating Software Testing: A Life-Cycle*	Mr. Robert A. Sabourin (AmiBug.Com) *Bug Priority And Severity*	Ms. Johanna Rothman, Elizabeth Hendrickson (The Rothman Consulting Group) *Grace Under Pressure: Handling Sticky Situations in Testing*

[BACK TO TOP]

SB — Standby Presentations

Technology Track	Applications Track	Internet Track	Management Track
Mr. Michael K. Jones, Assistant Professor (Dromedary Peak Consulting/Western International University) Test Strategy Derivation	Dr. John D. Musa (Consultant) More Reliable Software Faster And Cheaper -- An Overview	Dr. Kobad Bugwadia, Kris Mohan (Intel Corporation) Quality Issues, Requirements & Challenges For Multimedia Streaming On The Internet	Ms. Joanna Rothman (The Rothman Consulting Group) Successful Test Management: 10 Lessons Learned

QW2001 Tutorial A1 & A2

Mr. Tom Gilb
(Result Planning Limited)

Software Inspection For The Internet Age: How To Increase Effect And Radically Reduce the Cost

Key Points

- Streamline inspections through sampling and measurement focus
- Refocus on defect prevention, not debugging
- The one hour flat Silicon Valley version of Inspection

Presentation Abstract

Software Inspections were initially (1973 and on at IBM) used to clean up bugs, and to get some quantitative insight into bugs that might be introduced into testing and the field. This seminar will focus on a radical transformation of the original inspection. We do not even try to clean up bad work! It probably would pay off to 'burn and rewrite'. Inspection is used to measure the Major defect content of any software engineering specifications; requirements, design, test plans, code, user manuals, test cases, outsourcing contracts and web-based business plans. The cost can be kept down by sampling a few pages. The entire inspection cycle is reduced to one single hour, as done at a major web business client of ours. The key decision is to be able to 'Exit' the spec with less than one Major Defect per page, compared to the over 100 Majors/page which are common today. The key judgement is whether the spec matches a few critical best-practice 'Rules' of specification. In short, inspection should not be 'debugging' of bad work. It can be Quality Control through measurement and sampling.

About the Author

Tom Gilb was born in Pasadena in 1940, emigrated to London 1956, and to Norway 1958, where he joined IBM for 5 years, and where he resides when not travelling.

He has mainly worked within the software engineering community, but since 1983 with Corporate Top Management problems, and 1988 with large scale systems engineering. He is an independent teacher, consultant and writer. He has published eight books, including the early coining of the term "Software Metrics" (1976) which is the basis for SEI CMM Level 4. He wrote "Principles of Software Engineering Management" (1988, now in 13th printing, with 3 chapters on Evolutionary delivery methods), and "Software Inspection" (1993). Both titles are really systems engineering books in software disguise. His pro bono systems engineering activities include several weeks a year for US DoD and Norwegian DoD, and environmental (EPA) and Third-World Aid charities or organizations.

His clients include Hewlett Packard, Boeing, Microsoft, Ericsson, Alcatel, Nortel, Oracle, Sun, British Aerospace, UK Civil Aviation Authority, Litton PRC, Siemens, Medtronic and many others.

QW2001 Tutorial B1

Mr. Erik Simmons
(Intel Corporation)

Writing Good Requirements

Key Points

- Discuss and choose from different techniques to specify requirements
- Improve written requirements and tell good requirements from bad ones
- Write non-functional requirements so they are verifiable

Presentation Abstract

Presented at PNSQC 2000, attended by 54 people. First presentation outside of Intel, where it was taught to more than 1,000 students around the world this year. Of the 45 PNSQC evaluations returned, 44 would recommend the workshop to others and found it either "valuable" or "very valuable". People like the fast pace and depth of information presented. Poorly written requirements result in lost productivity, increased re-work, dissatisfied customers, poor end product quality, and even project cancellations. So, why are good requirements so hard to write? Many people do not know the key attributes of a "Good Requirement", and have not been exposed to the various effective ways to specify requirements.

This 1-day workshop focuses on and applies the best-known methods behind improved requirements writing. Based closely on a popular course taught at Intel, the course covers the different types of requirements and what activities are important when specifying requirements. The emphasis is on practical solutions to common problems, and contains valuable real examples from Intel documents in both original and improved formats. Students will gain an understanding of the attributes of a good requirement, and learn ways to identify whether the requirement is unambiguous, concise, necessary, correct, and traceable. Many useful "take it home and use it tomorrow" techniques for writing both functional and non-functional requirements are presented. Several exercises are included to reinforce the techniques. Attendees are invited to bring their existing requirements documents for use in the final exercise if desired.

About the Author

Erik Simmons has 15 years experience in multiple aspects of software and quality engineering. Erik currently works as a Platform Quality Engineer within the Corporate Quality Network at Intel Corporation. He leads the corporate Software Engineering Process Team that is charged with improving software development capabilities across Intel's product development groups, and is responsible for Intel's

product requirements engineering practices.

QW2001 Tutorial C1

Dr. Norman F. Schneidewind
(Naval Postgraduate School)

A Roadmap To Distributed Client-Server Software Reliability Engineering

Key Points

- Software Reliability Engineering
- Distributed Client-Server Systems
- Software Reliability Standards

Presentation Abstract

The objective of this tutorial is to help practitioners implement or improve a software reliability program in their organizations, using a step-by-step approach based on an enhanced version of the ANSI/AIAA Recommended Practice for Software Reliability, the IEEE Standard Dictionary of Measures of the Software Aspects of Dependability, and case studies from the NASA Space Shuttle and the United States Marine Corps logistical systems. Modeling methods, prediction techniques, and defect analysis for distributed systems will be emphasized.

About the Author

Dr. Norman F. Schneidewind is Professor of Information Sciences and Director of the Software Metrics Research Center in the Division of Computer and Information Sciences and Operations at the Naval Postgraduate School, where he teaches and performs research in software engineering and computer networks. Dr. Schneidewind is a Fellow of the IEEE, elected in 1992 for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance". He is the developer of the Schneidewind software reliability model that is used by NASA to assist in the prediction of software reliability of the Space Shuttle, by the Naval Surface Warfare Center for Trident software reliability prediction, and by the Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction. This model is one of the models recommended by the American National Standards Institute and th! e American Institute of Aeronautics and Astronautics Recommended Practice for Software Reliability. In addition, the model is implemented in the Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS), software reliability-modeling tool. He has published widely in the fields of software reliability and metrics.

In 1993 and 1999 he received an award for outstanding research achievements by the Naval Postgraduate School. He was Chairman of the Working Group that produced the IEEE Standard 1061-1992, Standard for a Software Quality Metrics Methodology and its revision in 1998. In 1993 he was given the IEEE Computer Society's Outstanding Contribution Award "for work leading to the establishment of IEEE Standard 1061-1992". In addition, he was given the IEEE Computer Society Meritorious Service Award "for his longÓterm committed work in advancing the cause of software engineering standards".

QW2001 Tutorial D1, D2

Dr. Gualtiero Bazzana
(ONION, S.P.A)

Web Testing Techniques and Tools (D1) (D2)

Key Points

- The tutorial focuses on testing methods and tools which can be successfully applied to the testing of Web-based applications, notably, presented from a technical point of view
- Internet WWW servers
- Intranet dynamic applications
- Extranet 30commerce application

Presentation Abstract

First of all, peculiarities of Web-based applications will be presented from a technical point of view, explaining their effects on testing practices. Moreover, the tutorial will deal with testing management aspects which are fundamentally affected by the nature of Web applications, including: RAD, regression issues, Testing solution will then be presented, both for static aspects (related to HTML, pictures, XML) and dynamic aspects (ASP, CGI, Proxies, Cookies, etc.).Room will be devoted also to commercial tools available in order to give the audience an overview of the existing technologies, highlighting also experience reports from their introduction, including ROI analysis. Special emphasis will then be given to the Web Accessibility Initiative (WAI) guidelines that have been issued by W3C and can significantly help testing of Web-based applications. Last but not least, the tutorial will touch the issues raised by integration testing between ERP and Web and in the validation of E-business solutions. Case studies will cover: testing of e-commerce sites, testing of commercial Internet Web sites, testing of Intranet sites, testing of home banking/ trading on-line applications

About the Author

Born in 1966, at university, he graduated with 110/110 and honour in Information Science at the University of Milan, in February 1989. His PhD won the special AICA award for topics related to quality in Information Technology. After working as software developer in a telecommunication company and as consultant/ manager in a consulting company he set-up ONION. His activities cover two areas of interest: consulting - projects in software engineering for various industrial companies and research in the field of software quality and networking (especially in the Internet/ Intranet domain). As far as consulting/ projects in industry are concerned, he matured and exploited know-how in conducting various medium

sized and large projects for several companies in various application domains (telecommunications, data processing, MIS, process control, etc.), covering topics like: Internet services, Intranet applications, Supply Chain management, etc. Moreover he has matured significant experiences in the ERP domain, notably with SAP R/3. He has matured significant technical experiences especially in the telecommunications domain (notably: switching systems and GSM mobile radio systems) in CIM and in networking, including Internet/ Intranet/ Extranet services and solutions. He has also dealt with sw development and testing, testing methods and tools, quality planning, test planning, reliability analysis, software product evaluation, process assessment and improvement, definition of quality systems in accordance to ISO 9001 and 9000/3, reviews and inspections, FDA computer system validation and so forth His research activity spanned in various fields of software engineering, ranging from Petri Nets to development methodologies, functional and structural test coverage, metrics and related tools, CAST, reliability evaluation, software development process evaluation and improvement, management by metrics, software product quality evaluation, security technology transfer and total quality management techniques. Moreover, he has co-ordinated several European Research Projects. He has published a book: ("Software Metrics for Product Assessment", McGraw Hill, London, 1995, International Software Quality Assurance Series, ISBN 0-07-707923-X), contributed to 4 other books (in the last one he has been author of four chapters dedicated to Software Process Improvement; it has been published by IEEE Software) and published over 50 papers at international conferences on topics related to software quality and software testing

QW2001 Tutorial E1

Mr. Robert A. Sabourin
(AmiBug.Com)

Getting Started -- Stressing Web Applications: Stress Early -- Stress Often

Key Points

- Stress Testing
- Load Testing
- Performance Testing

Presentation Abstract

This tutorial outlines very practical steps to follow when setting out to stress web applications - preferably well before they actually go live. Real examples from recent e-commerce projects are described - what works - what does not work - when to use what tools or technique? How to avoid blowing the budget!

What can we tell about the way our application will react to success? Will the application scale? How well? Can we tell? What can we tell? What can we simulate?

Can we set up a test lab organization which allows for stress testing at the Unit, Integration and System testing phases?

Techniques used in live e-commercce development sites are described. Where can and should stress testing be implemented, at what stage of development? What do we want to measure, how and why?

Practical tools and techniques are reviewed!

Examples are from the author's current experience related to testing some key e-commerce sites - noteably the Virtual Model Shopping Experience at www.landsend.com - www.jcpenny.com and many more extremely popular e-commerce sites!

The tutorial also includes elements of test planning and some practical examples of "risk analysis" applied to internet testing!

About the Author

Robert Sabourin has been involved in all aspects of development, testing and management of software engineering projects. Robert graduated from McGill University in 1982. Since writing his first program in 1972, Robert has become an accomplished software engineering management expert. He is presently the President of AmiBug.Com, Inc.; a Montreal-based international firm specializing

in software engineering and and software quality assurance training, management consulting and professional development. AmiBug helps companies set up software engineering and quality assurance teams and process through a combination of training and management consulting. Robert was the Director of Research and Development at Purkinje Inc where he was charged with developing world class critical medical software used by clinicians at the point of care. Previously, Robert managed Software Development at Alis Technologies for over ten years. He has built several successful software development teams and champions the implementation of "light effective process" to achieve excellence in delivering on-time, on-quality, on-budget commercial software solutions.

Robert has championed many complex international multilingual software development and globalization efforts involving several intricate business partnerships and relationships including international government (Czech, Egypt, France, Morocco, Algeria...) and commercial entities (Microsoft, IBM, AT&T, HP, Thompson CSF, Olivetti...). Systems included concurrent coordinated multilingual multiplatform product releases.

Robert's pioneering work with Infolytica Corporation led to the development of the first commercially available platform independent graphics standard GKS and several toolkits which allowed for cross platform development and porting of complex CAD, Graphics, Analysis and Non-Destructive Simulation systems.

Robert is a frequent guest lecturer at McGill University where he relates theoretical aspects of Software Engineering to real world examples with practical hands-on demonstrations.

In 1999, Robert completed a short book illustrated by his daughter Catherine entitled"I Am a Bug" (ISBN 0-9685774-0-7).

Robert has received professional recognition for many accomplishments over the years. At TEPR 2000 - award for best electronic patient record product to EHS using the Purkinje CNC component. Byte Middle-East's 1992 Product of the Year for the AVT-710 product family achieving a ZERO FIELD REPORTED software defect rate with over 15,000 units installed. (Project involved over 27-man month's effort!); Quebec Order of Engineers' recognition for creating and managing the Alis R&D Policy Guide - Development Framework and process.

QW2001 Tutorial F1

Mr. Ross Collard
(Collard and Co.)

Test Estimating

Key Points

- Test Estimating Techniques
- What you Need to Know to Estimate the Testing Effort
- Guidelines for the Estimating Process

Presentation Abstract

Question: When will the system testing be completed? (At the time the question is asked, you do not know (a) the final scope of the functionality, (b) when the developers will be done, and (c) what test resources you will have available. The boss wants a definitive answer in two minutes anyway.)

Answer: Take a wild guess and multiply by two.

Question: What do you do when the boss cuts your agreed-on test duration by 85%?

Answer: Whine pathetically that you thought that he or she really understood quality is important.

Developing realistic and credible estimates is a critical survival skill for test professionals and managers.

Topics covered:

Test Estimating Techniques

What you Need to Know to Estimate the Testing Effort
Guidelines for the Estimating Process
Rules of Thumb:
Estimating the Number of Test Cases
Estimating the Test Resources
Estimating the Elapsed Time for Testing
Deadline Pressures & How to Handle Them

About the Author

Ross Collard is president of Collard & Company, a consulting firm which is

headquartered in Manhattan, New York City.

His consulting and training clients have included: ADP, Alcatel, American Express, Anheuser-Busch, Apple, AT&T, Banamex, Bank of America, Bechtel, Blue Cross/Blue Shield, Boeing, British Airways, the CIA, Ciba Geigy, Cisco, Citibank, Computer Associates, Dayton Hudson, Dell, EDS, Exxon, General Electric, Goldman Sachs, Federal Reserve, Ford, Hewlett-Packard, Hughes Aircraft, IBM, Intel, Johnson & Johnson, JP Morgan, McGraw Hill, MCI, Merck, Microsoft, Motorola, NASA, Nortel, Novell, Procter & Gamble, Prudential, Sears Roebuck, Swiss Bank, U.S. Air Force, Verizon and Worldcom.

He has conducted seminars on business and information technology topics for businesses, governments and universities, including George Washington, Harvard and New York Universities, MIT, Stanford and U.C. Berkkeley.He has a BE in Electrical Engineering from the University of New Zealand (where he grew up), an MS in Computer Science from the California Institute of Technology and an MBA from Stanford University. His set of books on software testing is due to be published next year.

QW2001 Tutorial G1

Mr. Thomas Drake
(Integrated Computer Concepts, Inc ICCI)

The Quality Challenge For Network Based Software Systems

Key Points

- Internet quality for network centric systems
- Enterprise testing
- Quality network systems theory

Presentation Abstract

This tutorial will introduce a biologically inspired model-based conceptual framework for network-centric testing and systems level quality assurance. It involves an approach that can deal with computers and software viewed as a set of interactive and dynamic behavioral objects that are themselves part of a larger system rather than strictly for data processing and number crunching.

This conceptual framework for testing and quality assurance allows for examining and dealing with a range of application behaviors and outcomes and the possible interactions for these application objects without the necessity for fully understanding them in advance! This can permit testing the fundamental structure of a program and the application environment with the executable functional mechanisms and interfaces underneath and across the network.

It permits an “inside out” and end-to-end approach such that testing and quality engineering activities are based on the “genetic” makeup of the expected and anticipated dynamic “state” attributes and characteristics of the system using its own behavioral specifications as the test instruments for locating and stimulating the “weak” links.

In addition, this tutorial will provide an overview for what I have coined the Rapid Application Network Testing approach or RANT and examine the significant challenges posed by network-based software systems for testing and quality assurance. Tutorial will also consider the context and the background for understanding the daunting task faced by quality specialists and information technology management in dealing with the future, today!

This tutorial is meant to engage the participants and have you think “out of the box” when it comes to the testing and quality assurance choices and challenges we face every day in our increasingly networked environment.

About the Author

Mr. Drake is a software systems quality specialist and management and information technology consultant for Integrated Computer Concepts, Inc. (ICCI) in the United States. He also consults to industry and government on quality management and software quality engineering and code development issues.

As part of an industry and government outreach/partnership program, he holds frequent seminars and tutorials covering code analysis, software metrics, OO analysis for C++ and Java, coding practice, testing, best current practices in software development, the business case for software engineering, software quality engineering practices and principles, quality and test architecture development and deployment, project management, organizational dynamics and change management, and the people side of information technology.

He is the principal author of a chapter on “Metrics Used for Object-Oriented Software Quality” for a CRC Press Object Technology Handbook published in December of 1998. In addition, Mr. Drake is the author of a theme article entitled: “Measuring Software Quality: A Case Study” published in the November 1996 issue of IEEE Computer. He also had the lead, front page article published in late 1999 for Software Tech News by the US Department of Defense Data & Analysis Center for Software (DACS) entitled: “Testing Software Based Systems: The Final Frontier.” He is also one of the featured leading computer scientists interviewed in the textbook entitled: Problem Solving, Abstraction, & Design Using C++, Third Edition, 2000 by Friedman and Koffman from Addison Wesley Longman. Mr. Drake is a member of IEEE and an affiliate member of the IEEE Computer Society. He is also a Certified Software Test Engineer (CSTE) from the Quality Assurance Institute (QAI).

QW2001 Tutorial A2

Mr. Tom Gilb
(Result Planning Limited)

Software Inspection For The Internet Age: How To Increase Effect And Radically Reduce the Cost

Key Points

- Streamline inspections through sampling and measurement focus
- Refocus on defect prevention, not debugging
- The one hour flat Silicon Valley version of Inspection

Presentation Abstract

Software Inspections were initially (1973 and on at IBM) used to clean up bugs, and to get some quantitative insight into bugs that might be introduced into testing and the field. This seminar will focus on a radical transformation of the original inspection. We do not even try to clean up bad work! It probably would pay off to 'burn and rewrite'. Inspection is used to measure the Major defect content of any software engineering specifications; requirements, design, test plans, code, user manuals, test cases, outsourcing contracts and web-based business plans. The cost can be kept down by sampling a few pages. The entire inspection cycle is reduced to one single hour, as done at a major web business client of ours. The key decision is to be able to 'Exit' the spec with less than one Major Defect per page, compared to the over 100 Majors/page which are common today. The key judgement is whether the spec matches a few critical best-practice 'Rules' of specification. In short, inspection should not be 'debugging' of bad work. It can be Quality Control through measurement and sampling.

About the Author

Tom Gilb was born in Pasadena in 1940, emigrated to London 1956, and to Norway 1958, where he joined IBM for 5 years, and where he resides when not travelling.

He has mainly worked within the software engineering community, but since 1983 with Corporate Top Management problems, and 1988 with large scale systems engineering. He is an independent teacher, consultant and writer. He has published eight books, including the early coining of the term "Software Metrics" (1976) which is the basis for SEI CMM Level 4. He wrote "Principles of Software Engineering Management" (1988, now in 13th printing, with 3 chapters on Evolutionary delivery methods), and "Software Inspection" (1993). Both titles are really systems engineering books in software disguise. His pro bono systems engineering activities include several weeks a year for US DoD and Norwegian DoD, and environmental (EPA) and Third-World Aid charities or organizations.

His clients include Hewlett Packard, Boeing, Microsoft, Ericsson, Alcatel, Nortel, Oracle, Sun, British Aerospace, UK Civil Aviation Authority, Litton PRC, Siemens, Medtronic and many others.

QW2001 Tutorial B2

Mr. Bill Deibler
(Software Systems Quality Consulting)

Making the CMM Work: Streamlining the CMM for Todays Projects and Organizations

Key Points

- CMM, Process Assessment, Capability Maturity Model
- ISO 15504, ISO 12207, Software Process Improvement, Software Quality Models
- Software Quality Assurance, Tailoring, Small Projects

Presentation Abstract

BACKGROUND

The SEI Software CMM is a comprehensive model that can serve as a basis for assessing and improving the effectiveness of software development organizations. The CMM was derived from the requirements of government purchasing agencies overseeing large, complex, third-party development projects. Because of their large project focus, the practices described in the CMM can appear to small, internal, or commercial software development organizations to be inapplicable or burdensome and bureaucratic. Version 1.1 of the CMM is published in two technical reports containing a total of nearly 600 pages. The size of the CMM makes it difficult to uncover the interrelationships among the elements that are essential to tailoring the model to a small software development environment. It also makes the model intimidating.

LEARNING OBJECTIVES

This tutorial allows participants to identify and leverage the strength of the CMM to improve software development practices in their company. The tutorial prepares the attendee to build durable, maintainable software development practices that exploit the CMM framework. The tutorial ensures that the participant will be able to: * Implement a realistic and useful strategy for deploying software development practices in today's commercial organizations * Simplify the CMM to support appropriate, effective, flexible software development processes for any size organization * Resolve apparent discrepancies between the guidance in the CMM and the needs of small, commercial and internal software development projects and organizations * Identify and prioritize elements of advanced levels that should be considered by every organization.

About the Author

William J. Deibler II has an MSc. in Computer Science and 20 years experience in the computer industry, primarily in the areas of software and systems development, software testing, and software quality assurance. Bill has extensive experience in managing and implementing CMM- and ISO 9001-based process improvement in software engineering environments.

Bill is a principal of SSQC. Since 1990, SSQC has specialized in supporting organizations in the definition and implementation of Software Engineering Practices, Software Quality Assurance and Testing, Business Process Reengineering, ISO 9000 Registration and CMM implementation. SSQC offers HM2, a unique, hybrid appraisal method that defines and correlates the position of an organization with respect to both ISO 9001 and the CMM. The results of an HM2 assessment are a plan and framework for improving software engineering processes and for implementing the requirements of the two models. Bill has developed and published numerous courses, auditing tools, research papers, and articles on interpreting and applying the ISO 9000 standards and guidelines and the SEI Capability Maturity Model for Software. His articles have appeared in McGraw Hill's Quality Systems Update, IEEE COMPUTER, McGraw Hill’s ISO 9000 Handbook, CrossTALK, and Software Marketing Journal.

He has presented research papers at numerous national and international conferences, including those sponsored by the American Society for Quality Control (ASQC), Pacific Northwest Software Quality (PNSQC), the Software Publishers Association (SPA), Software Technology Support Center (STSC), the Software Engineering Institute (SEI) and Software Research Inc.. SSQC courses have been attended by software engineering professionals from many of the country's leading technology companies. SSQC courses have been sponsored for their members by professional associations, including the ASQC, CSU Long Beach's Software Engineering Forum for Training, Semiconductor Equipment and Materials International (SEMI), Software Engineering Institute (SEI), UC Berkeley and UC Santa Cruz.

SSQC is an active United States TAG member in the ISO/IEC JTC1 SC7 - Software Engineering Standards subcommittee which is responsible for the development and maintenance of ISO 12207 and ISO 15504 (SPICE). SSQC's software development clients have successfully achieved ISO registration and advanced CMM maturity levels.

QW2001 Tutorial C2

Dr. John D. Musa
(Consultant)

More Reliable Software Faster And Cheaper

Key Points

- Testing
- Operational profiles
- Software reliability engineering

Presentation Abstract

Software reliability engineering (SRE) can help those who are stressed out by competitive pressures to produce more reliable software faster and cheaper. It is a standard, proven, widespread best practice with substantial benefits that has been used successfully by organizations such as Alcatel, AT&T, Bellcore, CNES (France), ENEA (Italy), Ericsson Telecom, France Telecom, Hewlett Packard, Hitachi, IBM, Lockheed-Martin, Lucent Technologies, Microsoft, MITRE, Motorola, NASA's Jet Propuls ion Laboratory and Space Shuttle Project, Nortel, Raytheon, Saab Military Aircraft, Tandem Computers, US Air Force, and US Marine Corps.

SRE is based on two powerful ideas:
· Determine how often your customers will use the various functions of your product; then focus your resources in proportion to use and criticality. This approach greatly increases your development efficiency and hence your effective resource pool for adding customer value to your product.
· Further increase customer value by setting quantitative reliability objectives that precisely balance customer needs for reliability, timely delivery, and cost; engineer project strategies to meet them; and track reliability during test to guide release.
You can apply SRE to any system (new or legacy) using software and to members of software comp onent libraries. You can start with the next release.

This tutorial quickly, efficiently teaches the practical basics of how to apply this to your project. It uses a simple, realistic example throughout to illustrate the points. Participants are strongly encouraged to relate the tutorial material to their experience and to ask questions. A book Software Reliability Engineering: More Reliable Software, Faster Development and Testing was written in coordination with the tutorial; although not provided with the tutorial, its availability as a follow-on and supplement to the tutorial is very useful for those wishing to pursue the topic in more detail.

About the Author

John D. Musa is one of the creators of SRE, with more than 30 years varied and extensive experience as a software development practitioner and manager. Principal author of the highly-acclaimed pioneering book Software Reliability and author of the practical Software Reliability Engineering, Musa has published more than 100 papers on SRE. Elected IEEE Fellow in 1986 for many seminal contributions, he was recognized in 1992 as the leading contributor to testing technology. His leadership has been noted by every recent edition of Who's Who in America and American Men and Women of Science. Musa, widely recognized as a leader in SRE practice, initiated and led the effort that convinced AT&T to make SRE a "Best Current Practice." Musa has helped a wide variety of companies with a great diversity of software-based products deploy SRE. He is an experienced international speaker and teacher (over 200 major presentations) A founder of the IEEE Technical Committee on SRE, he is closely networked with SRE leaders, providing a broad perspective.

QW2001 Tutorial E2

Dr. Edward Miller
(eValid)

Client-Side WebSite Testing

Key Points

- In the past few years WebSites have grown from simple, static collections of HTML pages to complex pieces of software using advanced technologies including ASP, XML, script languages, e-commerce and more
- Testing of such complexity is a forever-growing challenge
- Testing of passive vs. interactive pages & static vs. dynamic pages will be explored
- The use of a ‘Test Enabled Web Browser’ will be emphasized as the most effective way of realistically testing most WebSites

Presentation Abstract

The Web is a complex place. There is much that is very importnat that can go wrong. What really counts in terms of quality is how users perceive a site. Because the client view is all-important, eValid's approach to WebSite testing is through a Test Enabled Web Browser.

This presentation outlines the reasons why the client-side view is important, and describes how a Test Enabled Web Browser can help sort out the WebSite quality issue.

About the Author

Dr. Edward Miller is President of Software Research, Inc., San Francisco, California, where he has been involved with software test tools development and software engineering quality questions. Dr. Miller has worked in the software quality management field for 25 years in a variety of capacities, and has been involved in the development of families of automated software and analysis support tools.

He was chairman of the 1985 1st International Conference on Computer Workstations, and has participated in IEEE conference organizing activities for many years. He is the author of Software Testing and Validation Techniques, an IEEE Computer Society Press tutorial text. Dr. Miller received his Ph.D. (Electrical Engineering) degree from the University of Maryland, an M.S. (Applied Mathematics) degree from the University of Colorado, and a BSEE from Iowa

State University.

QW2001 Tutorial F2

Dr. Cem Kaner
(Florida Institute of Technology)

Teaching Testing: A Skills-Based Approach

Key Points

- Training software testers involves teaching culture, vocabulary, concepts and skills
- We know what many of the basic testing skills are, but we can improve our training of them
- Skills are trainable. This session explores techniques for training key testing skills

Presentation Abstract

Training software testers involves teaching culture, vocabulary, concepts and skills. I think that many of the commercial seminars (and certification review courses) teach vocabulary and many concepts quite well. Some of them address cultural issues. Fewer address skills, but skills development is essential for new testers.

I've been training testers for 18 years, sometimes personally coaching them, sometimes teaching a pretty successful commercial seminar, and now teaching undergraduate and graduate students. Over the past year, I've rethought my approach to teaching.

When I studied mathematics, we learned a lot of conceptual material, but we also did a lot of drill--exercise upon exercise--from the course text and from exercise books like the Schaum's Outlines. These exercises forced the student to learn how to work with the concepts, and how to apply them under a wide range of circumstances.

Working primarily with James Bach, James Whittaker and Alan Jorgensen, I've been trying to develop a list of specific skills that testers use in the normal course of their work, and then develop exercises that will practice them in those skills. I have several such exercises now and am developing more as I go.

This session will go through practice exercises in bug reporting, domain testing, combination testing (all pairs), specification analysis for ambiguity, specification analysis for holes, and possibly some other areas. The more time we have, the more techniques we look at. In this session, I'll share course notes, quizzes, exam

questions, and explain how I use them.

About the Author

Cem Kaner is Professor of Computer Sciences at the Florida Institute of Technology.

Prior to joining Florida Tech, Kaner worked in Silicon Valley for 17 years, doing and managing programming, user interface design, testing, and user documentation. He is the senior author (with Jack Falk and Hung Quoc Nguyen) of TESTING COMPUTER SOFTWARE (2nd Edition) and (with David Pels) of BAD SOFTWARE: WHAT TO DO WHEN SOFTWARE FAILS.

Through his consulting firm, KANER.COM, he teaches courses on black box software testing and consults to software publishers on software testing, documentation, and development management. Kaner is also the co-founder and co-host of the Los Altos Workshop on Software Testing, the Software Test Managers' RoundTable, the Workshop on Heuristic & Exploratory Techniques, and the Florida Workshops on Model-Based Testing.

Kaner is also attorney whose practice is focused on the law of software quality. He is active (as an advocate for customers, authors, and small development shops) in several legislative drafting efforts involving software licensing, software quality regulation, and electronic commerce. Kaner holds a B.A. in Arts & Sciences (Math, Philosophy), a Ph.D. in Experimental Psychology (Human Perception & Performance: Psychophysics), and a J.D. (law degree). He is Certified in Quality Engineering by the American Society for Quality.

QW2001 Tutorial G2

Mr. Ed Kit
(SDT Corporation)

Establishing a Fully Integrated Test Automation Architecture

Key Points

- Discover the components of an effective test automation architecture
- Learn how to bridge test design and automation to provide an integrated test solution
- Understand the balance between process and technology
- Learn how to design and document highly inspectable tests
- Share experiences
- Examine a Web-based case study

Presentation Abstract

Learn the latest, 3rd generation approach to test design and automation to enable you to separate, yet integrate test design and automation, accomplish your software testing more quickly using fewer technical testers and setup a test architecture that requires far less maintenance. Experience testing a Web application will be examined as well as tips for getting started with test automation.

About the Author

Edward Kit, founder and president of Software Development Technologies (SDT), is well known as a test expert, author, and keynote speaker at testing conferences. An international software consultant with over 20 years experience in software engineering, Mr. Kit continues to advise clients on bringing practical and proven software quality practices to their development efforts.

Ed holds a BSEE and MSEE from Purdue University.

QW2001 Keynote 1P1

Mr. Ed Kit
(SDT Corporation)

Test Automation -- State of the Practice

Presentation Abstract

In a world of constantly changing technology, how we do business is changing - and how we need to test. Kit summarizes key shifts in software test automation and will present a new automation model which includes an integrated set of testing processes and techniques.

About the Author

Edward Kit, founder and president of Software Development Technologies (SDT), is well known as a test expert, author, and keynote speaker at testing conferences. An international software consultant with over 20 years experience in software engineering, Mr. Kit continues to advise clients on bringing practical and proven software quality practices to their development efforts.

Ed holds a BSEE and MSEE from Purdue University.

Test Automation - State-of-the-Practice

Edward Kit

www.sdtcorp.com **sdt@sdtcorp.com**

SOFTWARE DEVELOPMENT TECHNOLOGIES

Agenda

- **Automated Software Quality:**
  - **Industry Summary**
  - **Good News / Bad News / Key Trends**
- **Test Design and Automation for Complex Systems**
- **Recommendations for Effective Test Automation**

The Automated Software Quality Industry

- **Worldwide Automated Software Quality (ASQ) Industry:**
  - **10+ Years Old**
  - **$1+ Billion Market**
  - **30+% Annual Growth Rate**
  - **IDC reports that the ASQ market will grow to over $2.6 billion by 2004**
- **Continues to:**
  - **Gain Respect, Business Acceptance**
  - **Be Fueled by the Web**

Key Types of Testing Tools

Test Planning, Management & Control

- Requirements Management
- Test Management
- Configuration Management
- Problem Management

Reviews and Inspections

- Technical Review Management
- Complexity Analysis

Test Design and Development

- Data Generators
- Test Case Generators

Test Execution and Evaluation

- Coverage Analysis
- Capture/Playback
- Third Generation Automation
- Memory Testing
- Client/Server
- Simulators & Performance
- Web Testing

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

The Internet

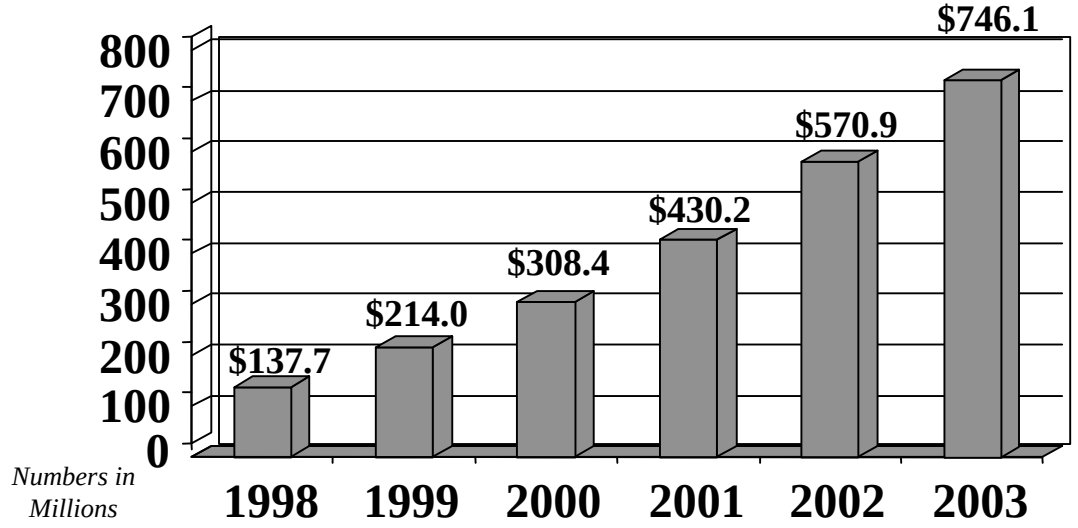
- **Permanently altered ASQ for the better**
- **Greatly increased the need for:**
  - **Test Automation**
  - **Performance, Security, Usability, Functional Regression Testing**
- **Enabled Web-based collaborative tools for Management of Requirements, Reviews, Defects, and Testing**
- **Opened the door for ASQ Service Providers**
- **Solidified the perception that automation is the only practical way to perform web load testing**

Web Load Testing is not Optional

- **Load Testing is the fastest growing segment of the ASQ market**
- **Web Application Performance:**
  - **directly influences user satisfaction**
  - **is tied to revenue, profits, brand value**
  - **leads to competitive advantage when customers are serviced efficiently**
  - **problems can lead to major disasters and losses**
- **Performance testing tools are required to carry out web load and stress testing**
- ***A good load tool with a poorly-designed load test will yield misleading and inaccurate results!***

The Load Test Revolution

Worldwide Load Test Market Growth Distributed Environments, 1998 – 2003



- **The market for load testing tools and services grew 55% in 1999**
- **Market revenue from Web application load testing tools and services increased 190% in 1999!!**

[Reference – Newport Group]

Load Testing – Have it Your Way

Flexible Delivery and Pricing Models:

- **Traditional buy tool / implement in-house**
- **Buy tool / contract with Load specialists to use in-house**
- **Load Test Tool Application Service Provider (ASP):**
  - **Customer rents tool, infrastructure**
  - **Customer designs, develops, manages tests and results**
- **Hosted Load Testing Management Service Provider (MSP):**
  - **Provide staff, expertise, objectivity, infrastructure, results**

Loads can be driven over the Internet

Great – But has Function Testing been forgotten?

Automated Function Testing

- **Traditionally addressed by capture / playback tools**
- **The secret is out: Capture / Playback alone does not work**
- **Automated Function Testing is taking a back seat while vendors regroup to deal with problems of:**
  - **Maintenance**
  - **Excessive Technical Resources required**
  - **Excessive tool total cost of ownership**
  - **Lack of attention to test design**

What is the value of doing the wrong thing in a big way?

E-Bugs in E-Software

- **Estimated worldwide Internet Commerce sales in 2003: $3.2 trillion [Reference: Forrester Research]**
- **Software is a critical success factor for nearly everyone**
- **Distributed system components multiply complexities making reliability more difficult to achieve**
- **Enormous visibility of E-Commerce software defects**
- **Expected levels of reliability and performance are generally not being met . . .**

E-Bugs are Expensive

- **Businesses worldwide lose $500 billion each year due to software failure**
- **Hershey suffered a $200 million decrease in revenues due to a software glitch that prevented Halloween candy from being shipped**
- **eBay market capitalization decreased $5.7 billion when investors lost confidence when systems crashed due to a software problem**
- **The SEC has fielded over 20,000 complaints from investors experiencing online trading and banking bugs**

[Reference: Payne]

Test Tools "Disturbingly Inadequate"

- **NASA administrator Daniel S. Goldin:**

  "Even as our reliance on software is increasing, the tools we have for verifying our software are disturbingly inadequate."

  "Only about 2 percent of all software programs are delivered on time, while meeting project requirements."

- **NASA-UC-Industry team is working on developing high-assurance software that can detect and correct errors in itself.**

[Reference: NASA]

The Current Sad Situation

- Project failure rates are still enormous
- Most software is still not tested well
- Early testing could have saved many failed projects
- The world of software development and testing has never been more complicated
- Test automation improves test effectiveness, but tools are often not successfully deployed
- An unfortunate casualty: Sun Java Testing Tools
- We're in the early days of routine, effective automation

Web Application Scalability Study Conclusions

- **More than half of recently deployed transaction-based Web applications did not meet expectations for how many simultaneous users the applications could handle**
- ***94% of the applications that scaled well used automated load testing tools prior to deployment!***

[Reference: Newport Group]

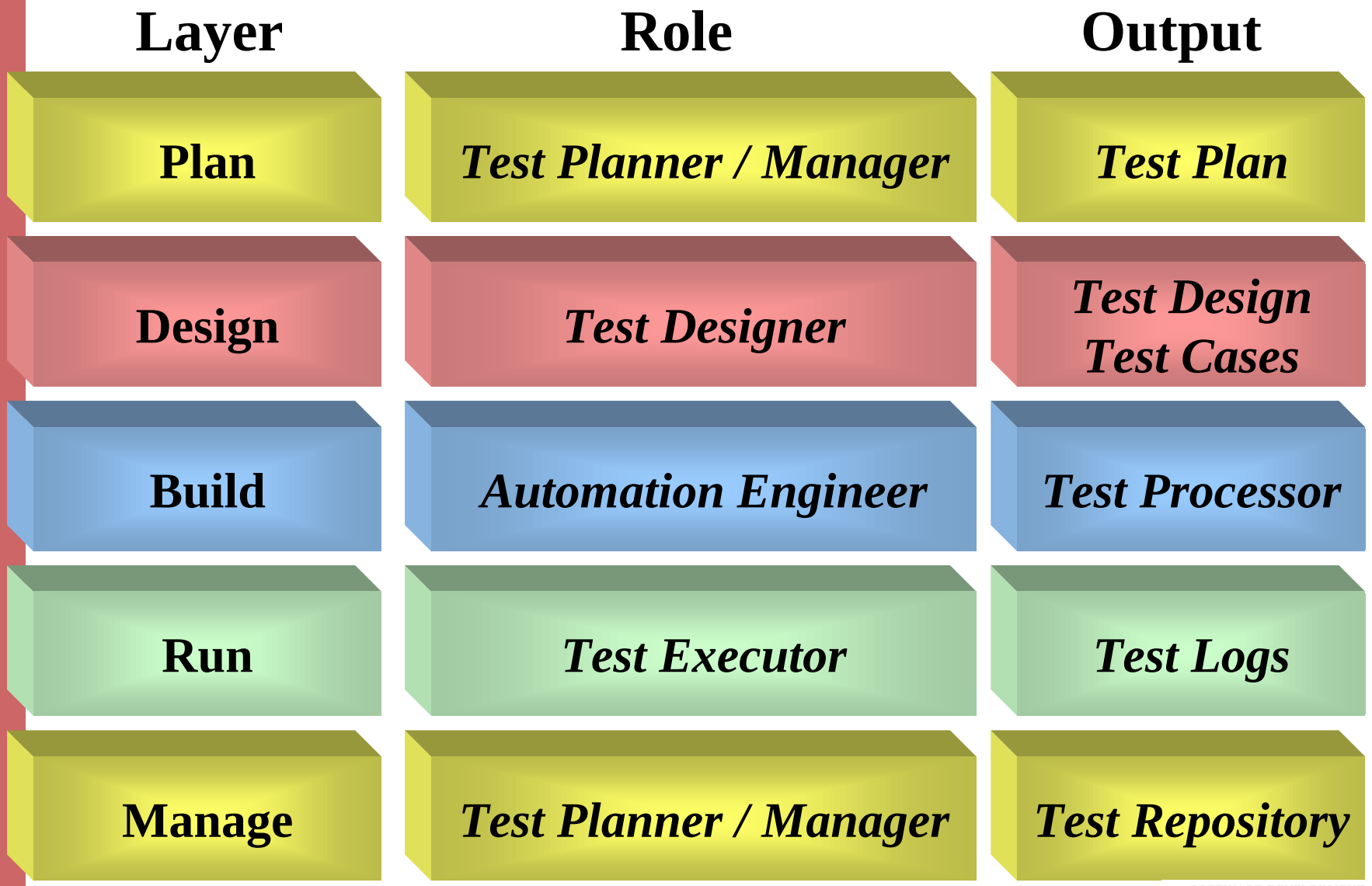
Test Automation: Serious Problems

- Lack of senior management knowledge and support
- Lack of an effective test automation architecture
- Lack of required specialized competencies:
  - Test Design
  - Technical Automation
  - Application
- Lack of sufficient resources:
  - Not enough people and / or staff turnover
  - Not enough time for automation implementation
  - Dedicated capital equipment

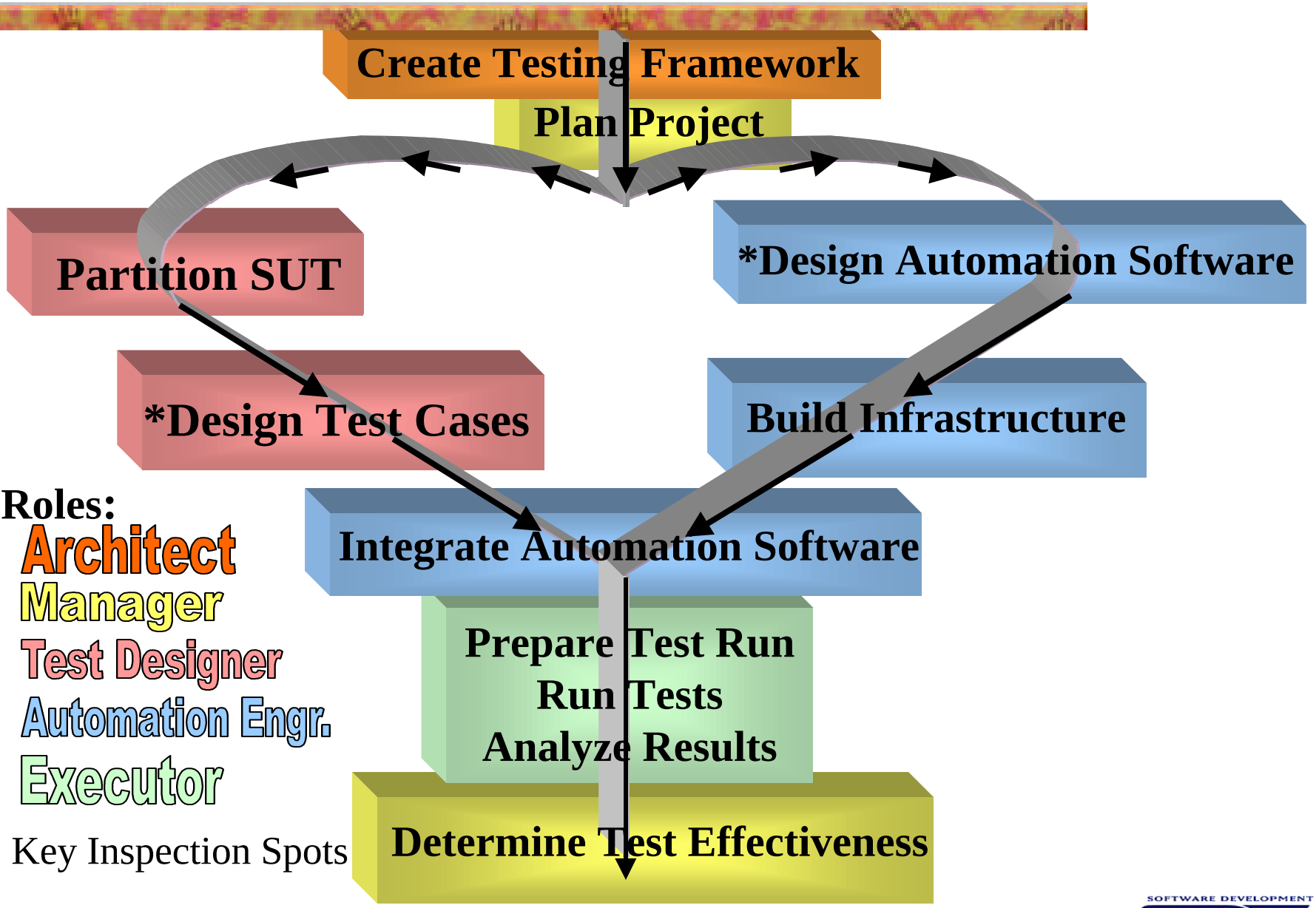
It's Time to Specialize: Roles-Based Testing

- **Test Architect -- Creates the testing framework, i.e., overall approach to verification and validation, including an integrated approach to test process and automation**
- **Test Planner/Manager -- Provides test planning, schedule, scope, resources, etc.**
- **Automation Engineer -- Creates automated test case processing capability**
- **Test Designer -- Creates and documents test design, participates in test design inspection**
- **Test Executor -- Runs and evaluates tests**

Test System: Layer / Role /Output



Roles-Based Key Activity Overview



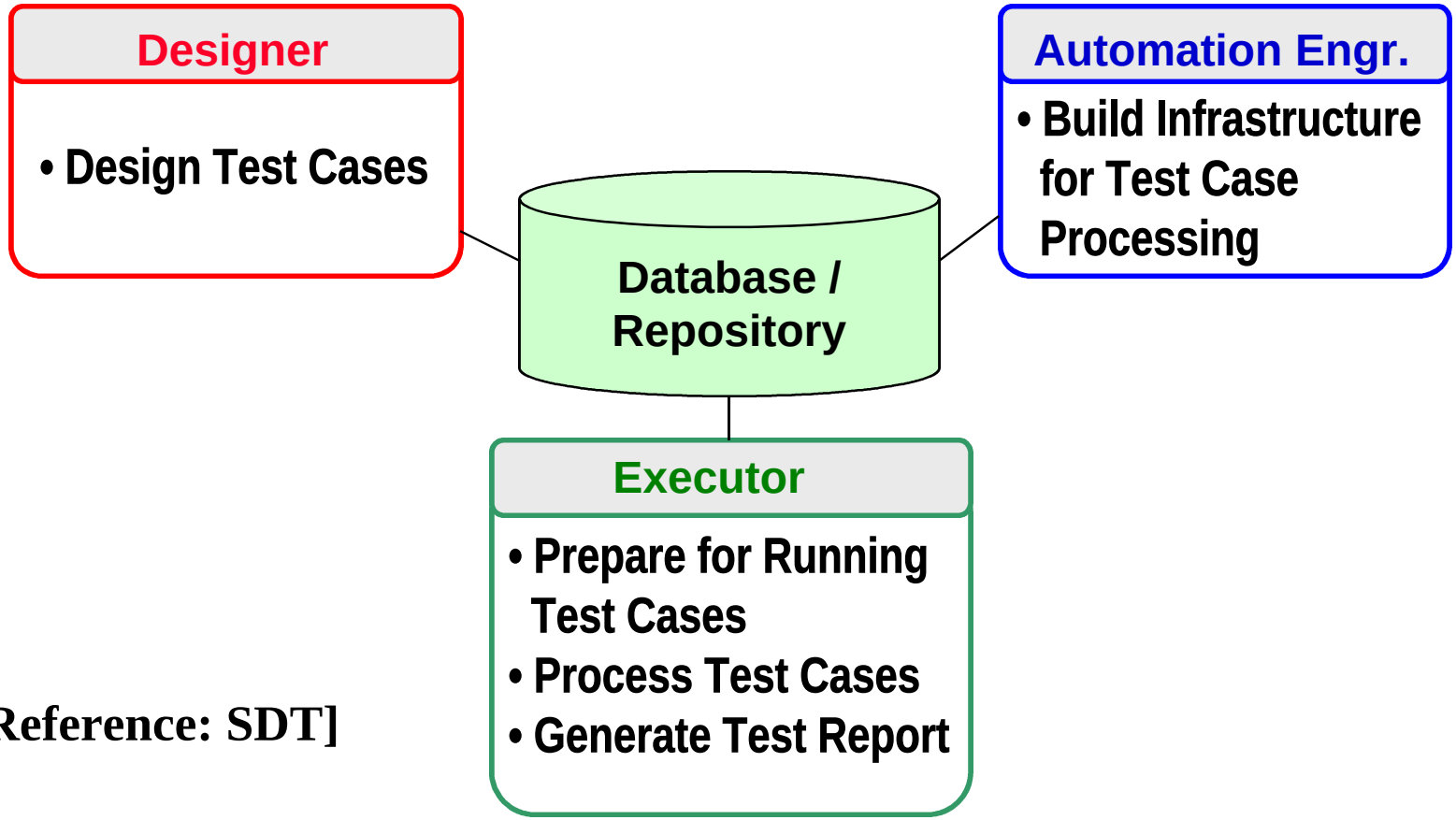
Roles:

- Architect
- Manager
- Test Designer
- Automation Engr.
- Executor

* Key Inspection Spots

Early Trend:
Roles-Based Test Environment

Tools are beginning to support Roles-Based Testing:

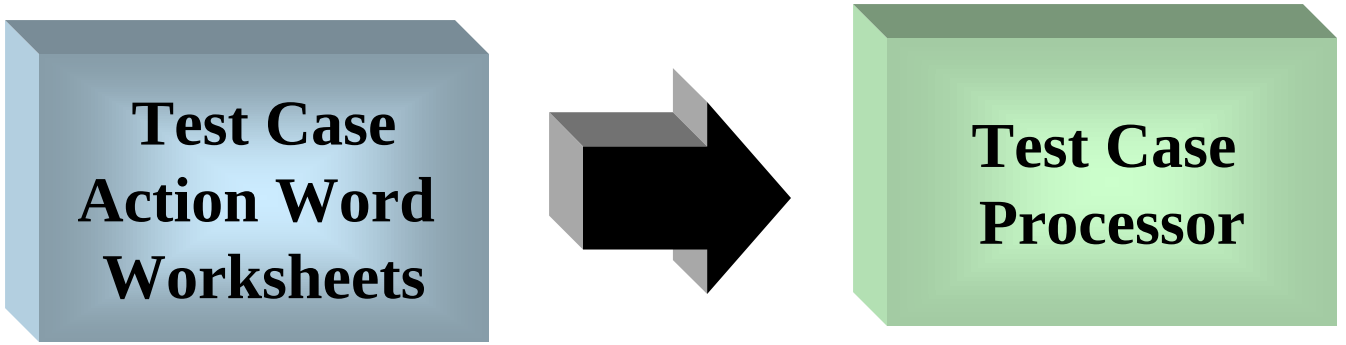


[Reference: SDT]

Recommendation:
Separate Designer and Automation Engr Tasks

Test Designer:
Functional Test Development

Automation Engineer:
Technical Test Execution



A	B
. . .	
set position	300 250 50
check response	
move	X 10
check position	X 310 2.5
. . .	

```
case action:
    "set position": ...
    "check response": ...
    "move": ...
    "check position": ...
end
```

[Reference: Kit, Buwalda]

Action Word Test Case Spreadsheet

	A	B	C	D	E
1	testcase	TC001.01			
2	sheet	Test the industrial robot position feature			
3	version	1.56			
4	date	June 7, 2000			
5	author	Lisa Robotson			
6	designer notes	Expected result: Pass – position set and checked			
7	designer notes	Submitter organization: RSTBU			
9	section	1. Position Robot			
10		*X*	*Y*	*Z*	
11	set position	300	250	50	
12	check response				
13		*axis*	*value*		
	move	X	10		
14	check response				
15	section	2. Check End Position			
16		*axis*	*value*	*tolerance*	
17	check position	X	310	2.5	
18	check position	Y	250	2.5	
19	check position	Z	50	2.5	
20					

Recommendation: Assess Your Roles-Based Balance

- Discuss having a balanced Roles-Based Testing Group
- Assess the skills of your existing group (see next slide)
- Identify your problem areas (largely influenced by your test automation model)
- Adjust your hiring plan to re-balance the group
- Recruit and hire needed candidates

Example Roles-Based Assessment

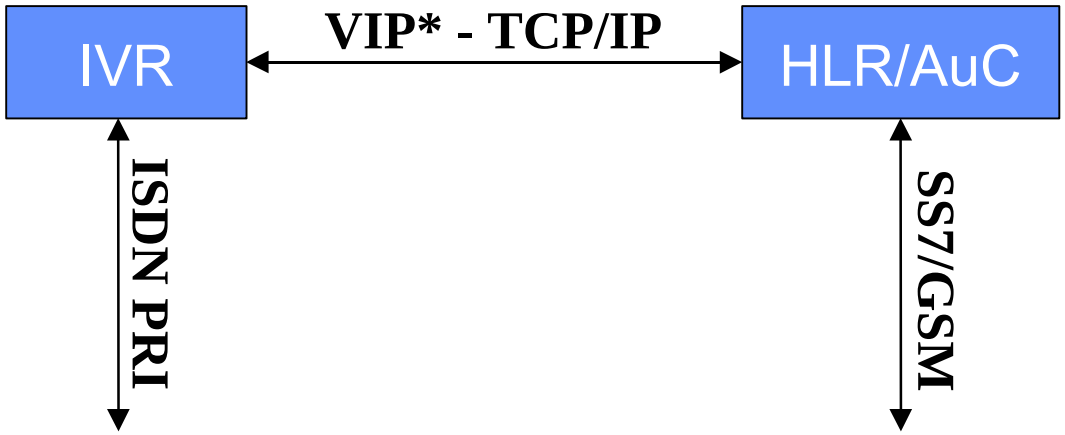
	Architect	Planner	Automation Engineer	Designer	Executor
Peter		Y	Y	Y	Y
Ravi				Y	Y
Mary					
John				Y	Y
James				Y	Y
Yves					Y
Anna				Y	Y
Li					Y

Recommendations: Preventing E-Bugs

- Verify the Architecture - Inspection
- Architect the Validation – Automation Architect
- Validate the Architecture - Automate Function Testing
- Test for Performance Regression - Automate Performance Testing
- Actively manage site performance - Hire a service provider to monitor and optimize your site performance
- Manage and Test Expansion - Start executing expansion plans when capacity reaches 60% of capacity

Different solutions / tools are required for each step

Telecommunication: Key Elements / System to be Tested



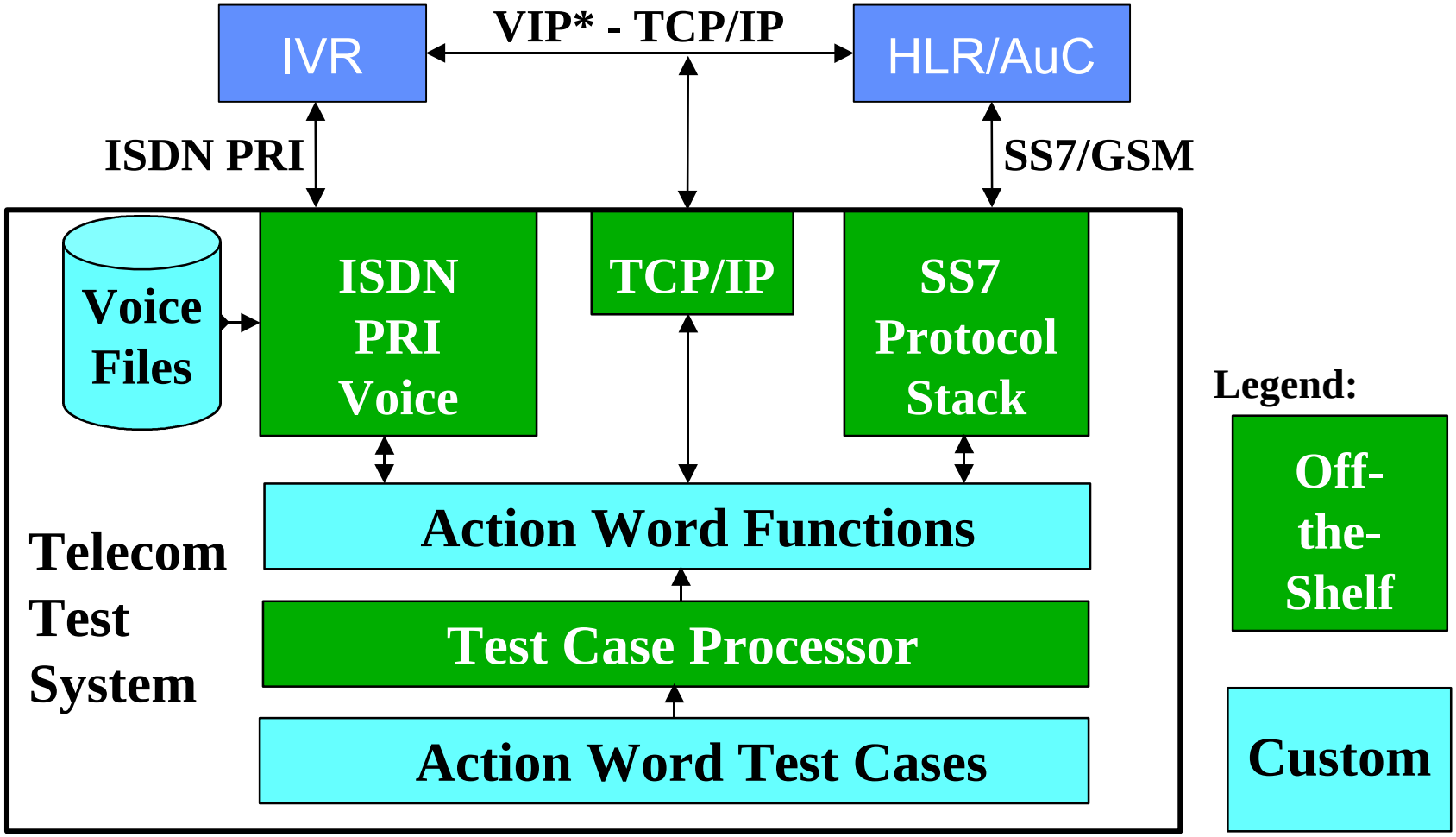
- Interactive Voice Response Application – (IVR)
- Home Location Register/Authentication Center - (HLR/AuC)
- Proprietary Protocol over TCP/IP – (VIP*)

The Wireless Telecommunication Challenge

Key Challenges:

- **Test Automation of complex element functionality**
- **Test Automation for complex integrated system**
- **Synchronized control of Voice, SS7, TCP/IP interfaces**
- **GSM / TCAP, ISDN PRI, and Proprietary Protocol Support**
- **Rapid deployment, cost-effective, maintainable, extensible**
- **Advanced troubleshooting and reporting capability**

Telecommunication Test Design and Automation Solution



Project Benefits

- Cost effective
- Software solution
- Off-the-shelf hardware
- Separation of Test Design and Execution
- Integrated Test solution
- Absolute Interface Control
- Extensible
- Enhanced Test Productivity

See also Greg Clower talk Weds at 3:30

Key Testing Success Factors

- Supported by senior management; treated as critical
- Adequately resourced with skilled test engineering professionals involved continuously with the right tools
- Clear and effective ownership and integration of test technology and process
- Proper training of participants - testing skills, like programming skills, cannot be mastered overnight
- Testing discipline independent of development
- Proper mix of verification and validation (V & V)
- Measurement program to know where you are standing

Bottom Line

E-Software that contains

E-Bugs carries extreme

Business Risks that can lead to

Business Failure

Sound business strategies include an approach to

Risk Management that includes identifying

Software Business Risk and an approach to

Testing that software is reliable and safe

Summary

- Consider Roles-Based Testing
- Strive for a balance of:
  - Process and Technology
  - Test Design and Automation
  - Function and System (e.g., Load Testing)
- Focus on the Keys to Success
- Early Testing can save many projects from failure
- Invest in people, provide training to enhance skills

References

- **Buwalda, Hans, *Testing with Action Words*, STAR May 1998**
- **Forrester Research:**
  see http://www.forrester.com/
- **Graham, Dorothy & Fewster, Mark, *Software Test Automation*, Addison Wesley Longman, 1999**
- **Jorgensen, Paul C., *Software Testing - A Craftsman' s Approach*, CRC Press, 1995**
- **Kit, Edward, *Software Testing in the Real World*, Addison Wesley Longman, 1996**
- **Kit, Edward, *Integrated, Effective Test Design and Automation*, Software Development Magazine, February 1999**

References

- **NASA News - High Dependability Computing Consortium:**
  see http://amesnews.arc.nasa.gov:80/releases/2000/00_81AR.html
- **Newport Group Study, June 2000**
  see http://www.newport-group-inc.com/
- **The Payne Report, Sept 2000**
  see http://www.cigital.com/paynereport/archive/sep2000.html
- **Sabourin, Robert, *I am a Bug*, Sabourin 1999**
- **SDT – Test Design and Automation:**
  see Unified TestPro at http://www.sdtcorp.com/unifiedtestpro.pdf
- **SDT – Inspection and Technical Review Automation:**
  see ReviewPro at http://www.sdtcorp.com/reviewpr.htm

The End

(The Beginning?)

QW2001 Keynote 1P2

Mr. Hans Buwalda
(CMG)

The Three "Holy Grails" of Test Development
(...adventures of a mortal tester...)

Key Points

- Key considerations for test development
- Suggestions to improve the test process
- Implications for management and control

Presentation Abstract

In the test development process, there are some important choices to make, which tend to have a key in the influence on its success. These choices can be compared with "Holy Grails". The Holy Grail is part of the ancient English legend of King Arthur and the Knights of the Round Table. It is a symbol for something that can be strived for, but not easily be reached. This is also true for at least some of the choices to be handled in the development process. In the talk three main topics will be presented, that have proven in practice to be the most important issues to address.

The model of the Holy Grails is based on experience with literally hundreds of testing and test automation projects. The grails provide a quick and very practical instrument to either plan test development or understand a large amount of problems in existing test development project, ranging from lack of coverage to unmotivated testers.

Part of the talk (of the “second grail”) is an introduction to the principle of “Soap Opera Testing”. In this technique, tests are formulated as small stories. They are based on the every day business practice for which the system under test is designed, but represent this compressed and exaggerated, comparable to the many soap opera shows on television. The stories are expressed with so-called “Action Words”, which are direct input to automated execution of the tests.

About the Author

Hans Buwalda is project director at CMG, a leading European information technology services group. He is responsible for new developments around the TestFrame approach for testing and test automation of which he is the original developer and main architect. In 1996, he presented the main ideas for the first

time to an international audience in a speech called 'Testing with Action Words, abandoning record and playback'. Since then the method has become in use in an increasing number of countries and Hans has become a frequent speaker at industry conference, tutorials and workshops.

The 3 "Holy Grails" of Test Development

Hans Buwalda

Test Consultant

buwalda@happytester.com

The Quest

(introduction)

Typical problems around testing

- Time consuming
- Costly
- Tends to be neglected
- Experienced as boring to do (in particular the execution)
- Hard to start in time

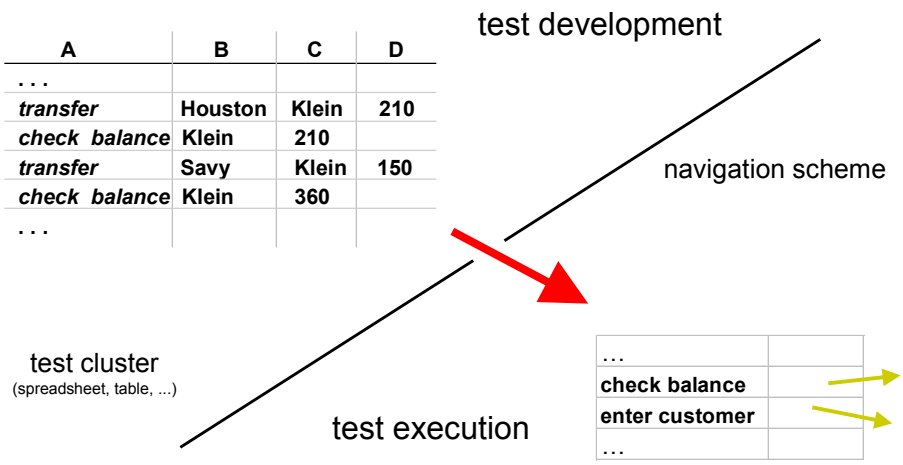
Even more typical problems with testing

- Difficult to manage:
  - What is the progress
  - What is the quality
- The proper resources (users, specialists) are not (made) available when needed
- Only small parts of the test execution are automated

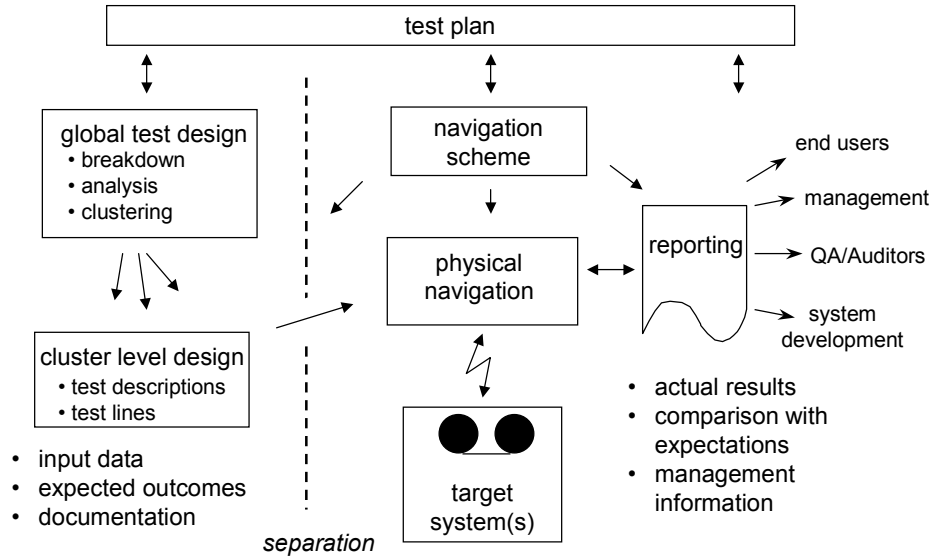
Challenges for a Test Process

- Testing should be fun
- Testing should be under control
- Testing should be effective
- Testing should be efficient

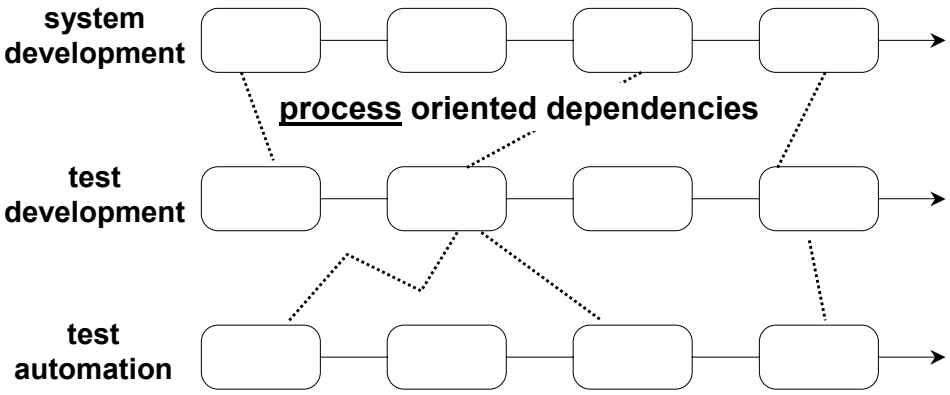
Re-usable test products



The product life cycle



Independence of life cycles



The Holy Grail

- Part of the legend of King Arthur
- Most of all a symbol
- To find the holy grail a knight had to be “pure at hart”
- To search for it is as important as finding it

The Holy Grail as a model

- General principles, good to “strive” for, but difficult to fully achieve
- The three principles suggested in this presentation are for test development
- Coming close to the grails will help quite a bit
- Being far away is a good recipe for trouble
- The first “victim” is the manageability

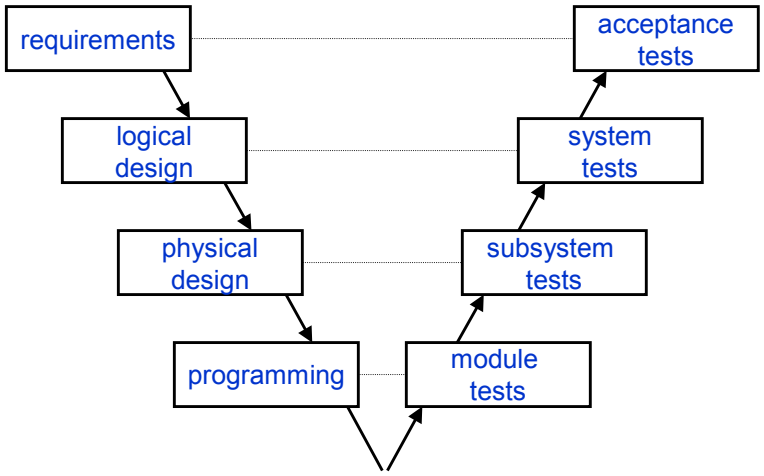
The three “holy grails” of Test Development

- effective “clustering” (break down) of tests
- the right approach per cluster
- the proper level of test specification

The First Grail

clustering

Well known, the V Model (simplified)



(inflexible) use of the V Model can involve risks

(see Brian Maricks article: www.testing.com/writings/new-models.pdf)

The “cluster model” as an alternative

- fit to project situation as an explicit step
- V Model is “member of the class”
- further decisions are per cluster:
  - how -> which technique, automate or not
  - when -> develop, execute
  - who -> stake holders, tester, auditor, ...
  - why -> business risk, complexity, "*made by John*", ...

leading thee in darkness . . .

Typical examples of what is not likely to be good clustering

- combinations of low and high level tests
- when you have to change all of them for a new system release
- if all clusters look alike
- clusters are dependent on each other
- you can't start developing them now

what could be thy fate . . .

Typical consequences / symptoms

- complaining people, no fun
- unnecessary high test maintenance (high impact of system changes on the test)
- difficulties in running any test
- "unpleasant" test process
- no (sense of) control

Clustering recommendations

- Logical to all concerned
- Independent from other clusters
- Well differentiated and clear in scope
- Fitting the priorities and planning of the project
- Balanced in size and amount

Examples of Clustering Criteria

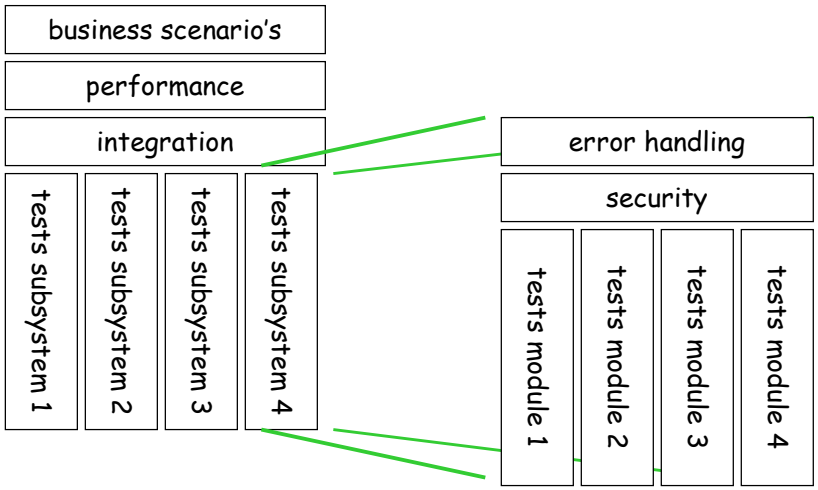
- Architecture of the system under test
- Functionality and other requirements
- Quality attributes
- Level of detail

} STRAIGHTFORWARD

- Planning and control
- Level of risks involved
- Complexity of the test
- Technical aspects of test execution
- Stake holders
- Code hand-offs (see Brian Marick)

} ADDITIONAL

Clustering in levels (example)



The Second Grail

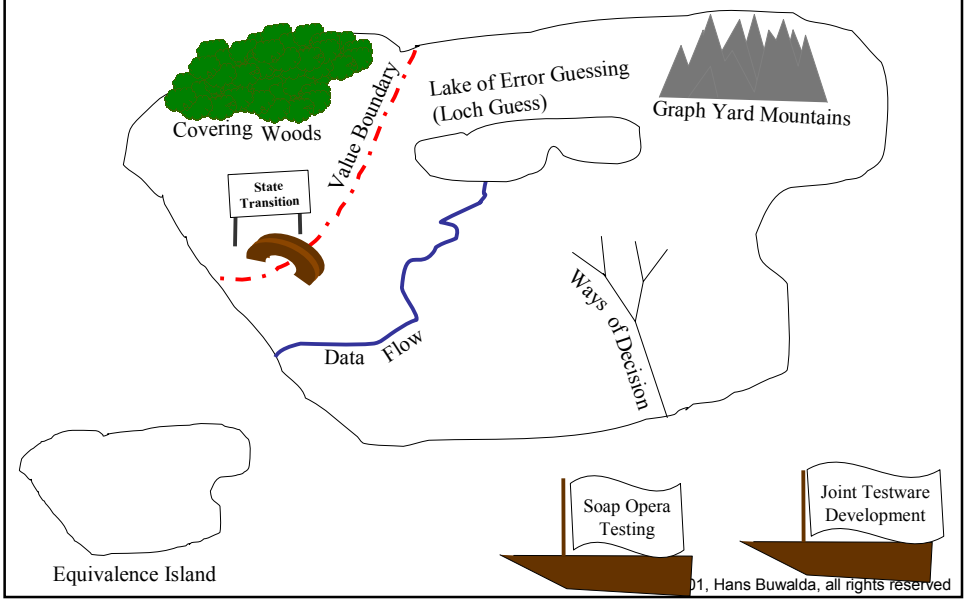
approach per cluster

Oh bethink thee, thou courageous tester . . .

Not very cool . . .

- the same approach for everything
- the wrong level of techniques, like using a boundary value analysis in a high level business test
- lots of hard labour without interesting results ("monks work")

THE REALM OF TECHNIQUES



The “mechanical approach” for test development (example)

- start with (preferably long) list of requirements
- make a test case for every requirement
- use a standardized test technique to translate the requirements into the test cases
- hire (many) people to peform the tests by hand
- ….

Some pitfalls with a too mechanical approach

- no fun at all
- inhibiting creativity
- coverage is focussed at single requirement level
- any defects should probably have been found in an earlier test
- suggests false sense of control
- testset hard to maintain
- doesn’t catch mistakes in the requirements
- ….

Soap Operas

Ashley hears about Jack's deposit when he thought he had to go. Victoria lectures her father about what's wrong with him and Nikki but Victor advises her that it's none of her business Olivia learns Dru has no regrets about leaving and takes great satisfaction in having Lily as her companion. Dru then asks Olivia why she is raking Malcolm over the coals. Stopping by Gina's, Nikki spots Brad and sits with him, admitting she doesn't want to be alone tonight. Victor stops by Mack's party at the Crimson Lights. Ashley takes a home pregnancy test. Worried about Billy, Raul makes call and J.T. claims he doesn't know where Billy is. Raul rushes over and finds Billy out cold in the snow Raul worries when he can't find a pulse. . . .

Properties of Soap Operas

- About “real life”
- But condensed
- And more extreme

Soap Operas for testing

- Define a scope of the test to develop
- Identify with the business environment
- Which elements would make things difficult
- Draft scenario's (typical some dozen lines)
- Write them down in clusters

Examples of story lines when used for testing

Pension Fund

William starts as a metal worker for Industrial Enthropy Incorporated in 1955. During his career he becomes ill, works part time, marries, divorces, marries again, gets 3 children, one of which dies, then his wife dies and he marries again and gets 2 more children….

World Wide Transaction System for an international Bank

A fish trade company in Japan makes a payment to a vendor on Iceland. It should have been a payment in Icelandic Kronur, but it was done in Yen instead. The error is discovered after 9 days and the payment is revised and corrected, however, the interest calculation (value dating)…

Example of test lines

	from	to	amount	valuta	trans nr
enter payment	123421344	4124244123	120000	yen	&keep tx1
check value dating	&tx1	$0.47			
wait days	9				
order to reverse	&tx1				
	from	to	amount	valuta	trans nr
enter payment	123421344	4124244123	1200000000	IKr	&keep tx2
check value dating	&tx2	$7,701.56			
. . . .					

Soap Operas (in testing) are <u>not</u> necessarily:

- “Extreme”
- Far fetched
- Long and elaborate
- Pieces of art and creativity

"Killer Soaps"

- More specifically aimed at finding hidden problems
- Run when everything else has passed
- One option: put a killer soap at the end of a normal cluster
- Ask the "specialists" for input

What can joint sessions give you

- Test Strategy
- Acceptance Criteria
- Cluster Grouping
- Test conditions
- Evaluation of Results
- Starting up development of scenarios

Joint Testware Development (JTD)(tm)

- Moderator / chairman
- Users
- Business specialists
- Developers
- Testers

Set-up of a joint session for a telecom provider

- 1st session
  - Introduction by moderator and project manager
  - explanation about the JTD procedure
  - explanation of the functional area by a specialised user
- 2nd session
  - start of production of test conditions
- 3rd session
  - start of production of test scenarios
- 4th session
  - evaluation test scenarios

The Third Grail

level of actions

Composition of Action Words

- Specification of an action, a check or a documentary statement
- Communication between Navigation and Test Cases
- Consistent
- Standard
- By-product of the test analysis

What is probably not a good action level

- “test all”
- low level “navigation” in a high level test:

```
type          "Hans"
press key     "<tab>"
type          "Buwalda"
click button  "OK"
...
```

- hard to understand “insiders only” language

```
set code   Fc122x   XX33
...
```


what could be thy fate . . .

Typical consequences / symptoms

- tests become quite unreadable (especially for non experts)
- unpleasant work to make the tests
- hard to understand the results
- high risk of mistakes
- and, of course, heavy maintenance dependency

Designing Action Words

- What actions does the test tool perform with a specific Action Word
- Scope of the test determines the Action Word level
- Manage the set of Action Words
- Document information about the Action Words

Low Level Actions

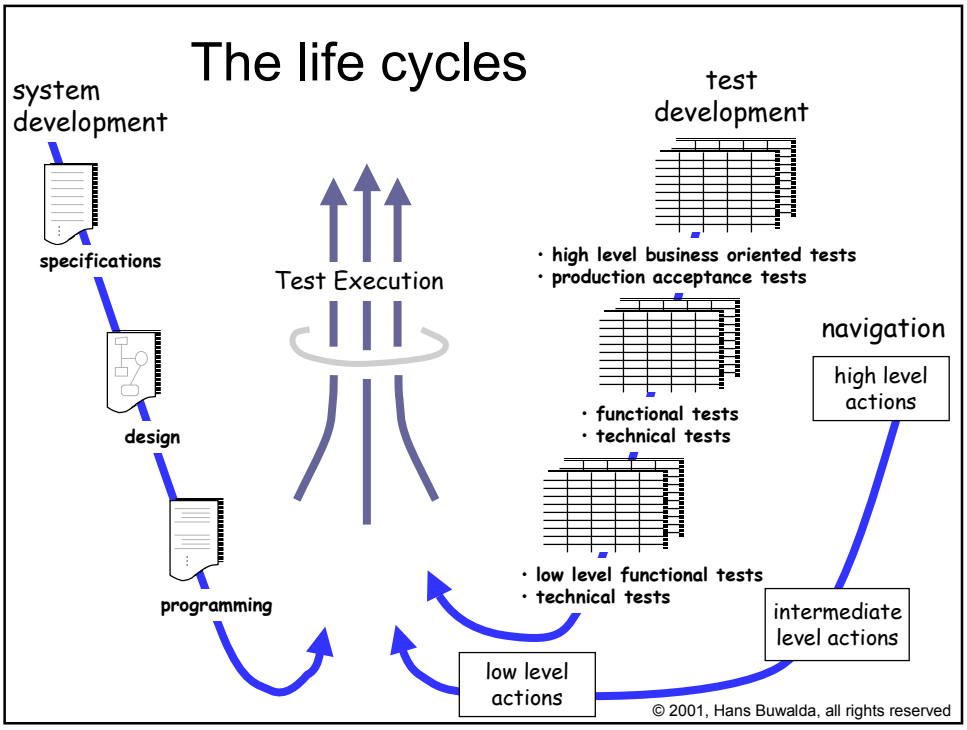
- Aimed at the platform(s) (techical angle)
- Low-level actions take care of the technical communication with the application
- Low-level actions are used by the higher level actions
- Typical examples:
  - push button
  - select list box item
  - capture text

Intermediate Level Actions

- Aimed at the interface(s) of the system under test
- Optional, use for complex navigation
- Typically the navigation at the level of:
  - one window in a gui
  - one record in a database
  - one message in a protocol
  - ...
- Used by high level actions (usually in combination with low level actions)

High Level Actions

- Aimed at the test (not necessarily the system under test)
- Defined by testers,
  - (not by navigators and not in advance)
- Typically the navigation of high level actions:
  - adds default values
  - moves across windows
  - takes care of unexpected situations
  - ...
- Uses low level and intermediate level



Acknowledgements

- Cem Kaner
- Hung Nguyen
- Brian Marick
- Dorothy Graham
- Brett Pettichord
- numerous former colleagues at CMG*
  (like Dirk van Dael, Erik Jansen, Dennis Janssen, Chris Schotanus, Jan Willem de Gruijter and André Kok)

* see also CMG's test approach: (www.testframe.com)

TESTFRAME

References

- Buwalda, Hans, *Testing with Action Words, Abandoning Record and Playback*, Eurostar May 1998
- Buwalda, Hans, *Testing with Action Words*, STAR May 1998
- Buwalda, Hans, and Kasdorp, Maartje, *Getting Automated Testing Under Control*, STQE Magazine, November 1999
- Buwalda, Hans, *Soap Opera Testing*, STAR East May 2000
- Graham, Dorothy, and Fewster, Mark, *Automating Software Testing*, 1999
- Kaner, Cem, Nguyen, Hung Quoc, and Falk, Jack, *Testing Computer Software, 2nd Edition*, John Wiley & Sons, 1999
- Kit, Edward, *Software Testing in the Real World*, Addison Wesley Longman, 1996

QW2001 Panel 4P

Mr. Brian Lawrence
(Coyote Valley Software)

How DO You Test Internet Software?

Key Points

- What is working testing Internet software
- How to analyze the your situation to help form a testing strategy
- Experience reports on Internet testing from the real world

Presentation Abstract

There have been quite a few vendors suddenly proclaiming that they are “the Internet Experts” or “the E-Business Experts” within the last year or so. These are the same vendors that were claiming something else prior to that. No doubt they have something useful to contribute, but where did their instant Internet expertise come from? Do they really have the last word on what to do, or are they just re-positioning what they were already doing to conform to the latest jargon?

How about asking your questions of people who’ve been there and done Internet testing?

This is a 90 minute panel proposal to discuss real circumstances testing Internet software. My panelists will tell you how they figured out testing strategies, and what types of testing they chose to perform. They’ll tell you how well it worked, and what they learned. You’ll have the opportunity to pose questions about your circumstances, and get their opinions on how to proceed. I’ve chosen thoughtful panelists who have had a wide variety of testing experience, both in Internet as well as other types of software. After I introduce the panelists, each panelist will take a few minutes to present their perspective on testing Internet software. Then I will open the rest of the session to audience questions.

About the Author

Brian Lawrence has moderated panels in a number of conferences, including at several past Quality Weeks. He has served as a program chair for the SEPG’97 Conference as well as the 1998 International Conference on Requirements Engineering. Brian teaches and facilitates requirements analysis, peer reviews, project planning, risk management, life cycles, and design specification techniques. Brian serves on the editorial board of IEEE Software and as the editor of Software

Testing and Quality Engineering magazine.

Elisabeth Hendrickson is currently an independent consultant specializing in software quality. With over a decade in the software industry, Elisabeth has participated on many projects--some successful, some not successful, and some disasters. Prior to becoming an independent consultant, Elisabeth was the Director of Quality Engineering and Project Management at Aveo Inc.

James Bach heads up Satisfice, a software testing consulting firm with a world class test lab located in rural Northern Virginia. James has extensive experience in a variety of testing situations, including for Silicon Valley startups, and larger organizations such as Microsoft, Borland, and Apple Computers.

Keith Stobie is the QA Process & Test Architect for BEA Systems, a leading provider of e-business solutions. Prior to working a BEA, Keith served in a leading role in defining testing strategies at firms like Informix and Tandem Computers.

Melora Svoboda is the Director of Engineering Services at Peakstone in Silicon Valley. Melora has worked in senior QA management positions at several e-business software firms on the west coast. Prior to that she defined testing strategies for Microsoft and other software firms.

QW2001 Paper 2V1

Mr. Steve Nemzer
(VeriTest)

Testing In Multi-CPU Environment

Key Points

- Testing challenges in multi-CPU “scale-up” environments
- Top 10 reason applications fail in multi-processor environments
- Strategies for testing database scalability

Presentation Abstract

Join VeriTest founder Steve Nemzer for an insider’s view into the unique challenges of testing software applications in multi-processor datacenter environments. In this presentation, you will learn the top reasons applications fail to scale on multi-CPU systems. You’ll also discover the testing strategies used by OEMs to uncover these flaws early in the development cycle.

About the Author

One of the pioneers of outsourced testing, Steve Nemzer co-founded VeriTest in 1987 to provide services to market-leading hardware and software developers. Over the last 14 years, he has led VeriTest to prominence as the premier test lab to the IT community. He is a frequent speaker at industry conferences including StarEast, Quality Week, COMDEX, LISA, and SD Forum. VeriTest now operates internationally as a service of Lionbridge Technologies, a leading provider of test and localization services.

Agenda

VeriTest.

Presenter:

Steve Nemzer, Vice President
VeriTest Business Unit

Agenda:

- **Scalability Defined**
- **Best Practices**
- **Tools to Consider**
- **Success Stories**
- **Questions**

What is Scalability?

- **Scalability Dimensions**
  - "Scale-Up" Testing
  - "Scale-Out" Testing

Internal Scalability

- **"Internal" Scalability**
  - Hardware Platform
  - Operating System
  - Database Platform
  - Application Design, Middleware components
  - Web Hosting Platform
  - Third-party Dependencies
  - Internationalization (I18N)

Scalability Test Emphasis

- **Performance**
  - Heavy client load
  - High transaction count
  - Transaction throughput
  - Response time SLA

Best Practices

- **Planning: Buy or Build?**
  - Buy
    - eBiz builders
    - Managed Services and Co-Location
    - “Integrated” Turnkey Apps
  - Build
    - Ensure scalability is in the spec from the start
  - Establish the success criteria
  - Test Strategy
    - “White box” architecture testing
    - “Black box” user experience testing

Best Practices: “Black Box” Testing

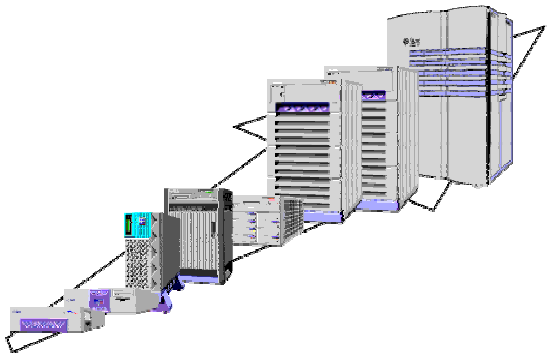
Example: eCommerce Site with six different simultaneous pipelines

- Customer profiling pipeline
- Advertising push pipeline
- Catalog search pipeline
- Shopping cart pipeline
- Credit card transaction pipeline
- Application management pipeline

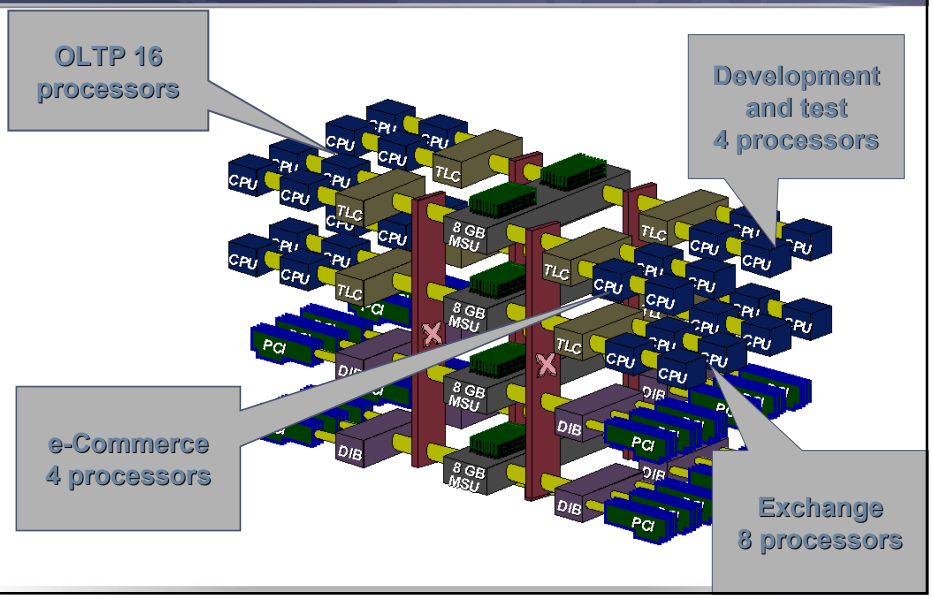
Best Practices

- **Hardware Platform Provisioning**
  - Scaling up: Multi-CPU platforms
    - 4-, 8-, 16-, or 32-way?
  - Scaling out: Clustered Servers
- **Operating System Platform**
  - Solaris 8, Windows 2000 Data Center

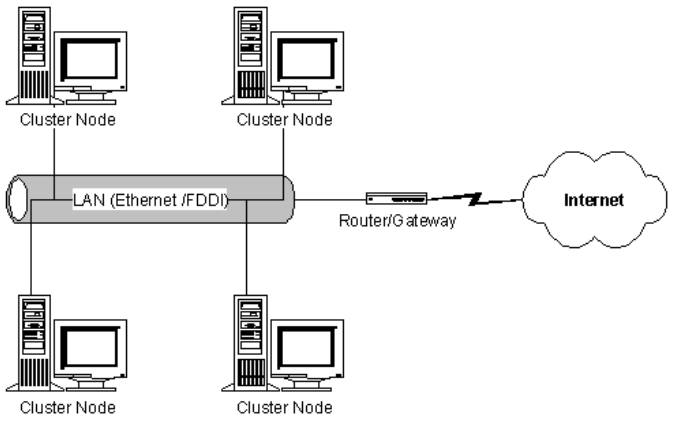
"Scaling Up": External View



"Scaling Up": Internal View



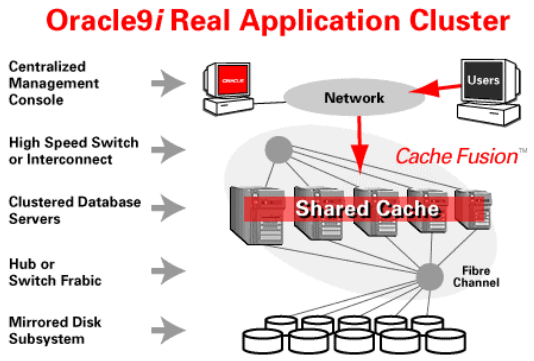
Scaling Out



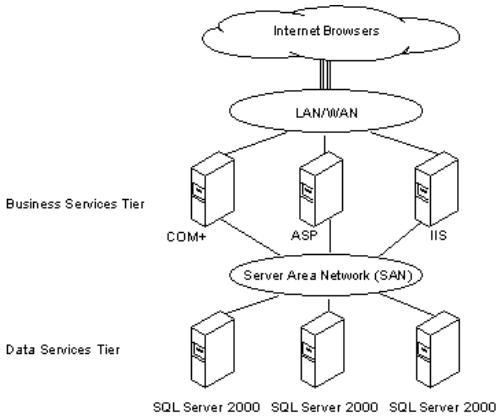
Best Practices

- **Database Platforms**
  - Oracle 9i, MS SQL 2000, Informix, Sybase
  - “Parallel Server” Technologies

Database platform scalability



Database platform scalability



Best Practices

- **Third-party Dependencies**
  - Middleware
  - Commerce Engines
  - Client-side plug-ins
- **Internationalization**
  - Scaling in the linguistic dimension
  - Enablement

Best Practices

- **Performance**
  - Architect the web app with certified modular components
  - Ensure end-to-end functionality testing
  - Identify and test the simultaneous functional pipelines
  - Predict most common ‘business processes
  - Predict initial and forecasted client loads
  - Develop load test cases
  - Execute locale-specific enablement testing

Tools to Consider

- **No right way to scale**
- **Hardware**
- **Software**
- **Network**
- **Testing**

SUCCESS!

- **Case Studies**

Contact Information

Steve Nemzer, Vice President
VeriTest Business Unit
Lionbridge Technologies, Inc.
ph. 310-636-8680
Steve_Nemzer@veritest.com

VeriTest Services

LIONBRIDGE

VeriTest.

- **Comprehensive Testing**
  - Scalability
  - Functionality
  - Globalization
  - Configuration
  - System/Migration
  - Compatibility
  - Accessibility
- **Consulting**
- **Certification**
  - Commercial and Enterprise

Certification

LIONBRIDGE

ORACLE e-certified warehouse

Certified For Microsoft Windows
Windows 2000 Datacenter Server
Windows 2000 Advanced Server
Windows 2000 Server
Windows 2000 Professional

Microsoft .net Enterprise Servers

TERMINAL SERVER TESTING PROGRAM
Microsoft ® Windows NT® Server 4.0 Terminal Server Edition

BMC SOFTWARE Certified

Powered by Microsoft Windows CE

COMMERCE SERVER 2000

Designed for Microsoft BackOffice

Built with ObjectARX AutoCAD

IBM Netfinity ServerProven

N YES TESTED & APPROVED Novell

Featuring Microsoft Visual Basic for Applications

QW2001 Paper 2V2

Mr. Francois Charette
(TestQuest)

Testing Configurable Product Platforms

Key Points

- Test Automation
- Ingrated Devices
- Test Generation and Techniques

Presentation Abstract

Electronic systems manufacturers are under tremendous pressure to rapidly deliver competitive products to market. Product development lifecycles have shrunk enormously and this has forced product engineering to be innovative. One recent approach borrowed from the manufacturing world has been the use of a platform-based approach to the hardware and the software. A fully integrated platform approach to product design allows manufacturers to create better, full-featured, lower-cost devices.

Such devices are completely configurable in that the graphical user interface and the behavior of the applications completely changes depending on a set of inputs provided. Such systems pose unique challenges to the product development especially from a verification and validation perspective.

About the Author

Mr. Francois Charette, BSEE from the Royal Military College, Kingston, Canada, has more than a decade of Test Automation and Quality Assurance experience. He has consulted on a variety of projects for fortune one hundred companies regarding test automation, quality assurance, and software development for process control, infra-red devices, set top devices, digital/analog video and electronics manufacturing.

Testing Configurable Products

Francois Charette
Director, Solutions Consulting

testQuest

Platform-Based Products

1. Prevalent model in multiple industrial segments
   - Automotive, medical systems, consumer electronics, computing, control systems
   - Egs., K-car, VW, Pocket PC,
2. Based on a shared hardware and software base
3. Shared core functionality
4. Data and configuration dependent personalities and capabilities

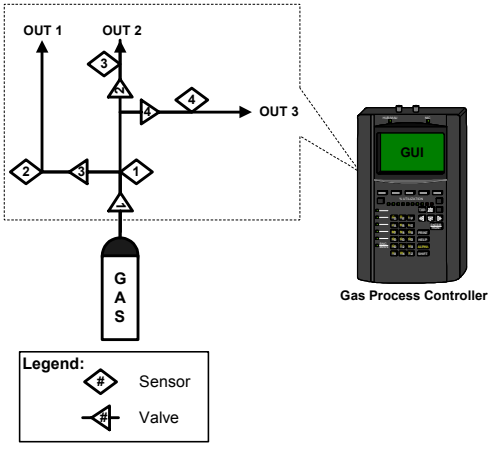
testQuest

Attributes of Configurable Products

1. Typically is a member of a platform family
2. Product behaviors either factory configured (static) or field configured (dynamic)
3. Hardware and software constituents/behaviors changed through configuration changes – multiple data sets

testQuest

Configurable Gas Control System



testQuest

Test Requirements

1. High fidelity testing of a changing product interface
2. Fully simulated in-context testing with all electro-mechanical sub-systems active
3. Full characterization of product behavior correlated to permutations of valid input stimuli
4. All valid configurations to be functionally tested
5. Invalid configurations tested – failure management
6. Reusability of all test artifacts

testQuest

Challenges – Tools & Frameworks

1. Deploying a functional test tool chain which enables high-fidelity in-context testing of configurable products
2. Structuring test frameworks to reflect multiple possible valid and invalid configurations of the configurable product.

testQuest

Functional Test Tool - Design Targets

- Advanced operating system platforms
- High performance applications
- Non Windows conformant applications
- Heterogeneous applications
- I/O intensive applications
- Fixed function computing

Test Automation Tool – Design Goals

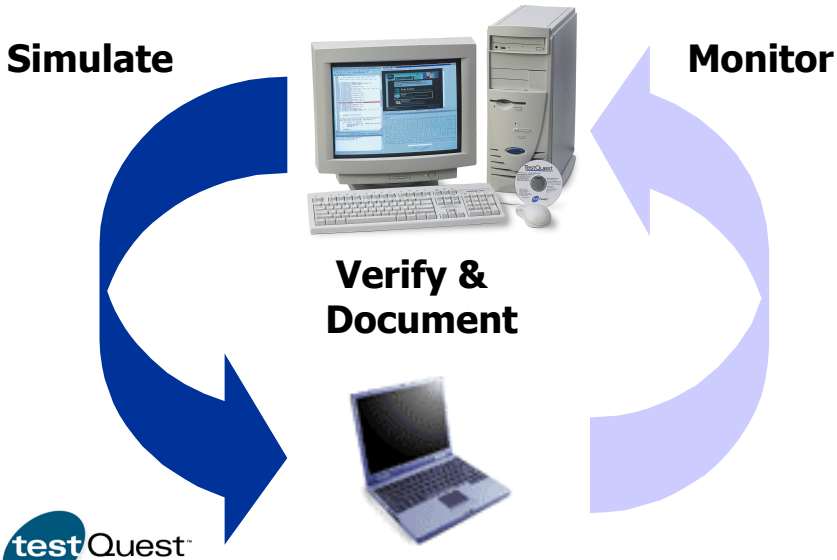
- Target agnostic
- Complete non-intrusivity on SUT
- Eliminate necessity for manual inputs
- Reflect high fidelity usage scenario
- Cover standard use cases
- Expandable to test accessories

TestQuest Pro – Overview

- Test Automation Tool
- Non-Intrusive black box functional testing for computer-based systems
- High productivity script development environment
- Industry standard scripting language - C/C++
- Open platform for integration of optional modules

testQuest

TestQuest Pro System Approach



testQuest

TestQuest Pro I/O Connectivity

Simulate Device Input

Simulation Examples

- Keypad
- Pushbuttons
- Keyboard
- Infrared
- Pointing Devices
  - Mouse, Trackball...
- Touch Panels
- RS-232/422 Serial Comm.
- Analog
- Discrete
  - Power On/Off...

testQuest

TestQuest Pro I/O Connectivity

Monitoring Examples

- VGA
- Digital LCD
- S-Video
- RS-232/422 Serial Comm.
- Analog
- Discretes
  - LED's
  - Audio

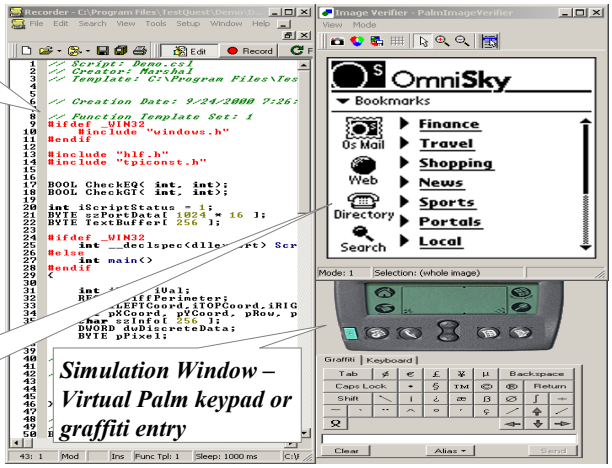
Monitor Device Output

testQuest

TestQuest Pro - IDE

Recorder Window - Full Featured Editor, C Interpreter and Debugger

Image Verifier Window - Displays Product screen During Script Development



Simulation Window – Virtual Palm keypad or graffiti entry

Develop, Run & Debug Test Scripts in the Interpretive Script Recorder Environment

testQuest

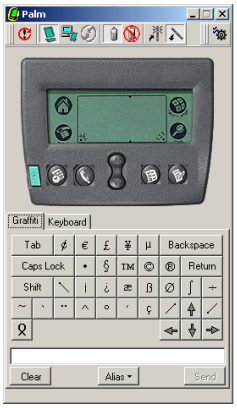
TestQuest Pro Palm System Modules

Palm Test Platform Hardware

- Hard wired to a single Palm target
- Available in Palm III, V, and VII models
- Enables Keypad functions
- Simulates touch screen functioning
- Simulates docking for Hot Sync
- Enables discrete functions- power on/off
- Enables functioning of peripheral devices

testQuest

TestQuest Pro Palm System Modules



Palm Test Platform Plug-in

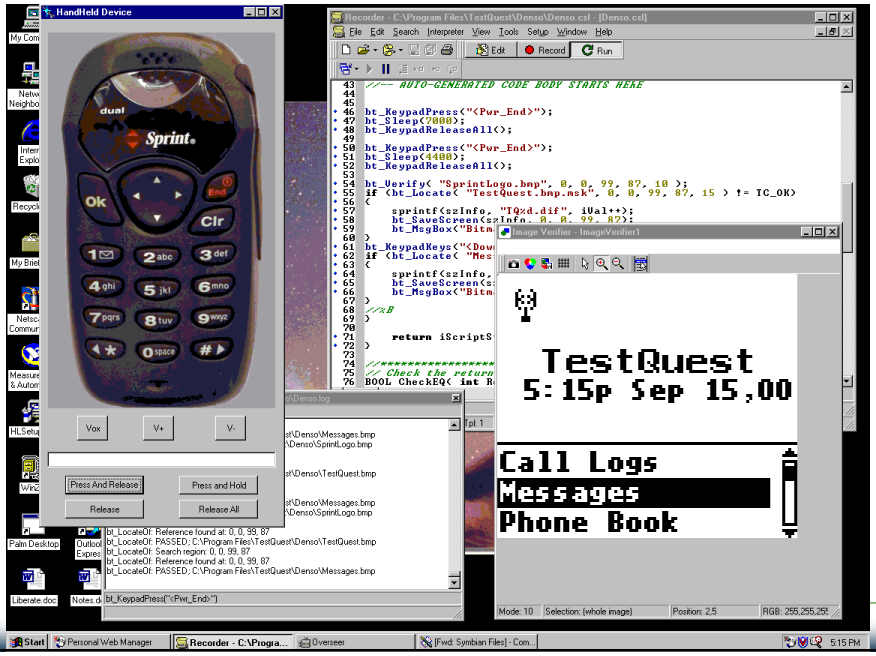
- Full Feature Palm plug-in
- Virtual Palm device that simulates Palm Test Platform
- Actions performed on virtual Palm will automatically insert code into test script
- Graffiti Simulation
- Actions can be performed by clicking on buttons or onscreen keyboard

testQuest

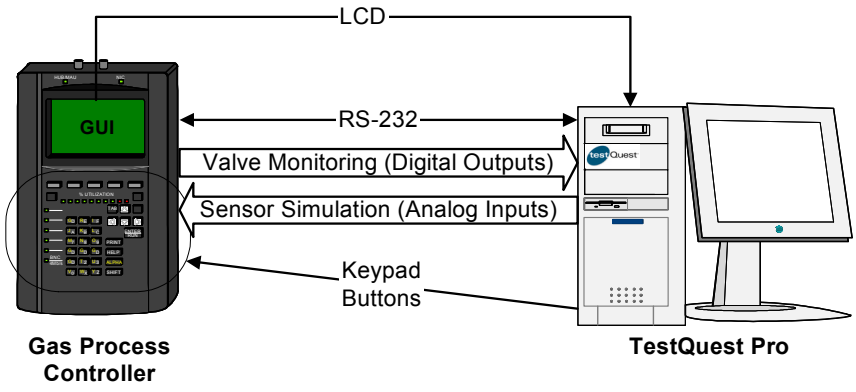
Cell Phone – Single Screen

testQuest

Cell Phone – Single Screen

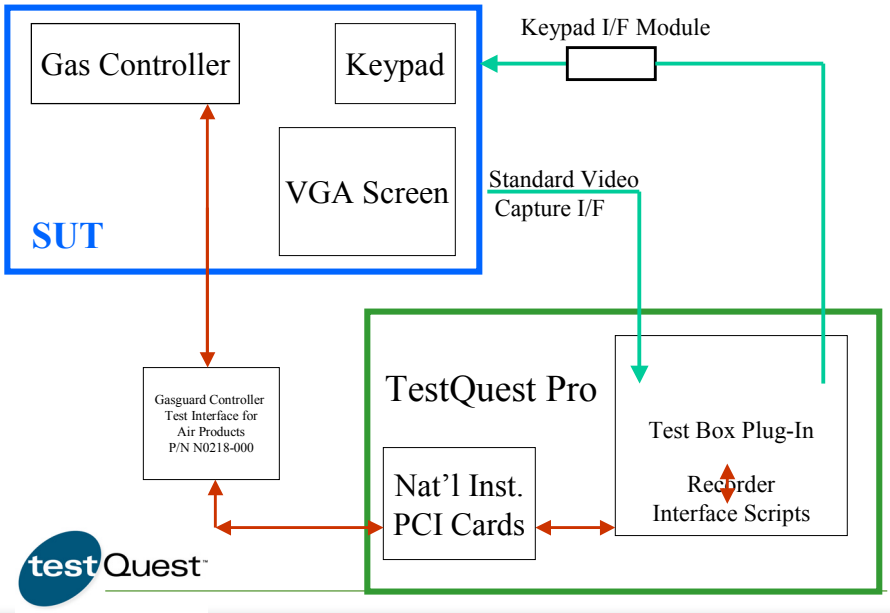


Integration with Gas Controller



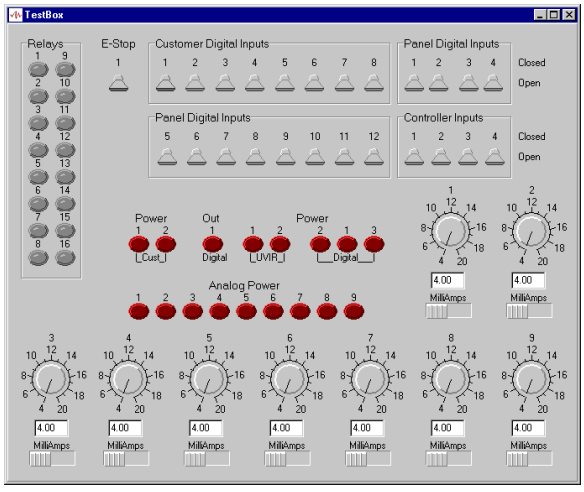
testQuest

Gas Controller Integration Topology



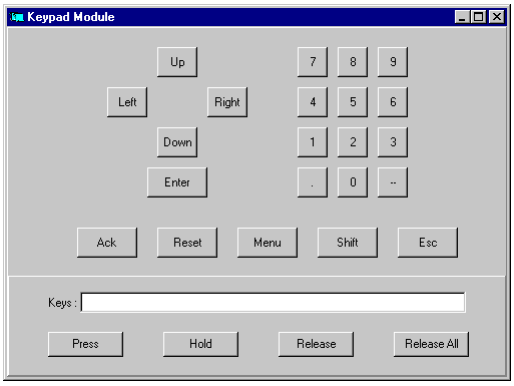
testQuest

Integration UI Component



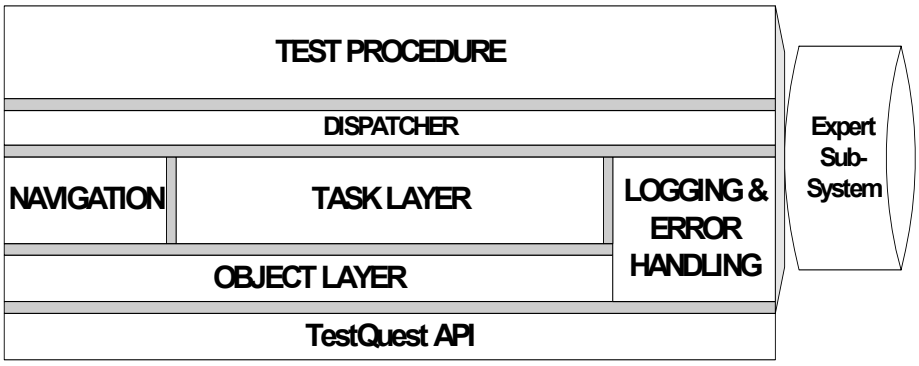
testQuest

Integration UI Components



testQuest

Test Framework Architecture



testQuest

Test Framework Software Modules

1. Test Procedure
   - Closely corresponds to test cases
2. Task Layer
   - Common sequences of actions that are used in multiple procedures
   - Set of domain specific utilities
3. Object Layer
   - Abstraction layer that encapsulates target screen artifacts
4. Dispatcher
   - Abstract execution layer used by test procedures and tasks
5. Expert Sub-System
   - Imports and parses core product configuration data to generate test data sets

testQuest

Lessons Learned

1. Domain Abstractions
2. Data Independence/Configuration Independence
3. Shared Configuration Data
4. "Getting There" vs. "Testing There"
5. Capture-Replay?

testQuest

Testing Configurable Product Platforms

François Charette
Solution Consulting, Director
TestQuest Inc.

Keywords: Test Automation, Integrated Devices, Test Generation and Techniques

Abstract

Electronic systems manufacturers are under tremendous pressure to rapidly deliver competitive products to market. Product development lifecycles have shrunk enormously and this has forced product-engineering groups to be innovative. One recent approach borrowed from the manufacturing world has been the use of common hardware and software platforms across product lines. A fully integrated platform-based approach to product design allows manufacturers to create better, full-featured, and lower-cost devices.

Such devices are often completely configurable in that the graphical user interface and the behavior of the application completely change depending on a set of inputs. Such devices pose unique challenges to product development groups especially from a verification and validation perspective.

Background

Today's device manufacturers design and productize multi-function and configurable system instead of fixed configuration systems. This model provides manufacturers

with the ability to quickly deploy a variety of large-scale system with specific and unique configurations, all using a single configurable platform. This methodology eliminates the need to create custom solutions to meet customer requirements. This increases the manufacturers return on investment and reduces long-term maintenance costs. This design approach is commonly utilized in segments and industries as varied as defense, command and control, automotive, process control, and consumer electronics. It is now finding its way into small, embedded, fixed function devices such as, warehouse inventory scanners, cell phones, pagers, handheld, set top boxes, etc.

Originally, this concept was applied to hardware components utilizing standard communication channels and protocol to integrate into a larger system. However, in today's complex product architectures, software components are designed using the same concept. The introduction of CORBA, COM/DCOM and now, .NET, has created a similar standard communication protocol for creating flexible and configurable software systems.

To design and implement these systems, engineering teams are tasked with creating and validating complex software and hardware architectures. The complexity of these systems, combined with the short development lifecycles imposed by time to market pressures can be quite difficult on the engineering team.

The product-engineering group that experiences the most pressures is usually the test organization which is confronted with:

- Shrinking test cycle: Technology companies must engineer and deploy new and enhanced products quickly to stay competitive. Hence the engineering group is faced with a much shorter development and test cycle.

- Reduced manpower: Most test organizations are understaffed. This is usually attributable to two factors: a very tight employment market and the difficulty of attracting fact qualified test engineers.

- Increasing number of test requirements: With increased product complexity and flexibility, there are significantly more usage permutations to be tested. Hence the scope of testing is broadened.

- Increasing test complexity: Because of the product's flexibility and its inherent complexity, testing scenarios are becoming more and more complex and lengthy. Therefore, such test scenarios can only be created by technical staff that possess intimate knowledge of the system under test.

- High number of test scenarios and permutations: Finally, the number of product configuration can be very large. Therefore, the permutations of test cases to be performed must be identified carefully to provide acceptable test coverage.

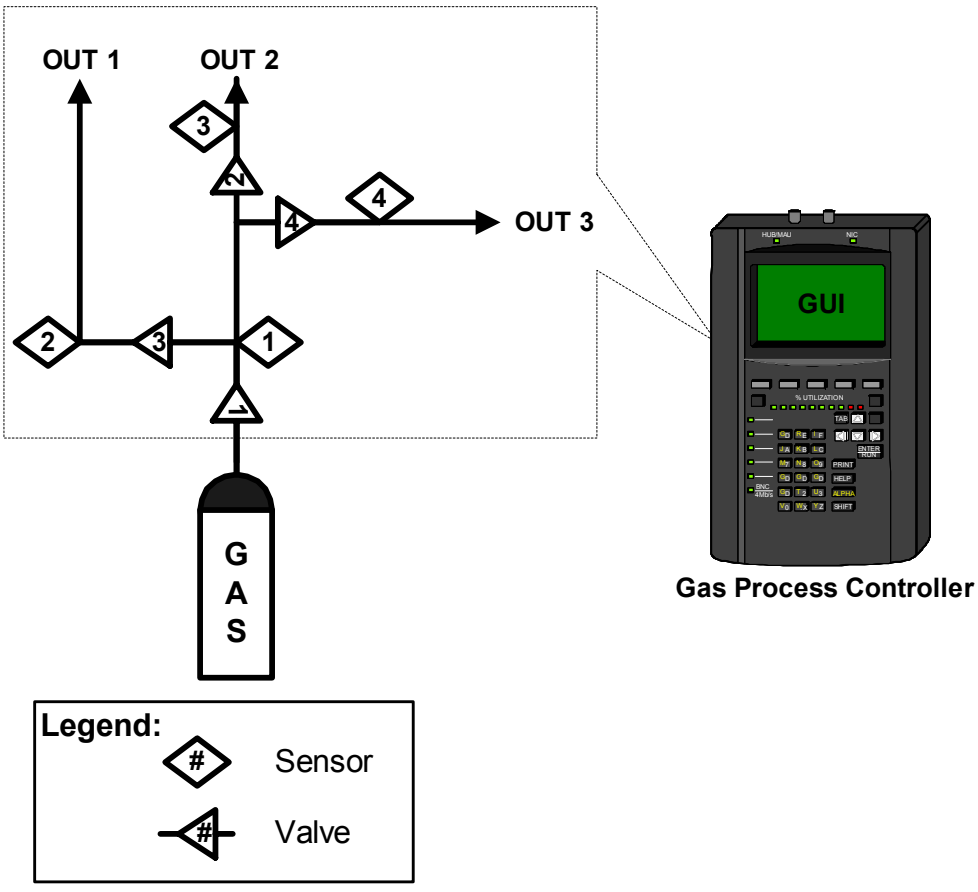
This paper proposes one method for helping test organizations fulfill these new test scenarios by utilizing a configurable, automated, test solution. To better illustrate this

we use suitably modified real-world application to describe the testing challenges and the resolutions to these problems.

The System Under Test

The system under test is an embedded gas process controller. The gas process controller is used to monitor gas flow and to raise warnings and alarm sequences when specific conditions occur. This product can be configured to monitor one or more gas lines. The gas process controller monitors the gas lines using pressure sensors and controls the warning and alarm sequences using a configurable number of valves.

The following diagram illustrates a simple configuration composed of one gas cylinder, three gas lines, four valves and four pressure sensors.



Gas Process Controller

A single gas process controller can monitor and control the gas flows from up to four gas cylinders. Each gas cylinder can have:

- Up to four lines.
- Up to eight sensors.
- Up to eight valves.

For each sensor, pressure warning and alarm thresholds can be set which can trigger action sequences to recover from the fault.

The system is physically configured and a configuration file containing information about the following is downloaded to the gas process controller:

- The number and position of gas cylinders.
- The number and position of the gas lines.
- The number and position of the valves.
- The number and position of the sensors
- The associated warning and alarm sequences which are also configurable.

The configuration information will determine specific aspects of the graphical user interface of the gas process controller.

Testing Challenges of Configurable Systems

Configurable systems with complex functional behaviors and user interfaces present a unique set of challenges. Tests must be designed to handle the myriad possible configurations. Test tools, especially test automation tools, must support these test designs and must provide a powerful set of capabilities to exercise all functional elements of the system under test.

In the example of the gas controller described above, full coverage functional testing requirements include:

1. The graphical user interface of the gas controller must be fully exercised in a manner identical to normal operator usage. This must be done with a high degree of fidelity.

2. The complete operating context of the gas controller system, including operations of valves, sensors, and other electro-mechanical devices must be fully simulated to a high degree of fidelity.

3. The functional behavior of the gas controller must be fully characterized and correlated to various permutations of input stimuli. When the gas controller is

stimulated with a complete set of inputs (GUI, valve position, sensor inputs, etc.) does its complete characteristic output behavior (GUI, control signals, audio annunciators, etc) match expected output behavior?

4. All legal configurations of the gas controller must be functionally tested using a uniform and consistent set of tests.

The two main challenges arising from these requirements are:

1. Deploying a functional test tool chain which enables an extreme fidelity, in-context functional test of the gas controller

2. Structuring the functional test scripts to reflect the multiple possible configurations of the gas controller.

In the next section we describe a functional test automation tool which powerfully satisfies such complex test requirements. In the subsequent section we describe the test strategy and the associated test scripts that were designed to address the challenges posed by target configurability.

TestQuest Pro – A Functional Test Automation Tool

TestQuest Pro is a functional test automation that was designed specifically to enable the testing of complex systems such as the gas controller described earlier. Through a system of electrical connections to the I/O points of the device under test, TestQuest Pro can completely, accurately, and with very high fidelity simulate an operator for the device under test (DUT).

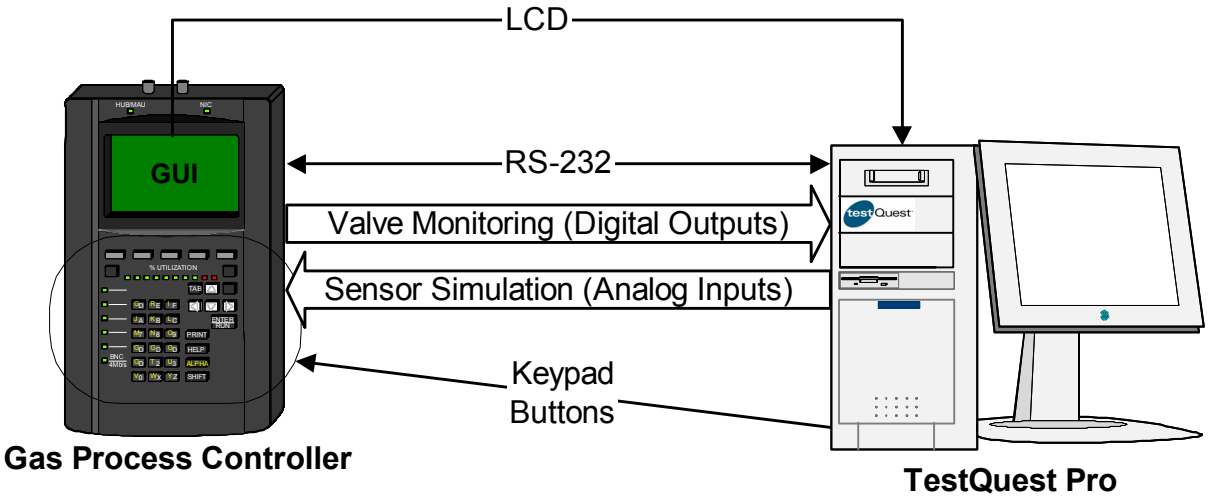
Thus, it can stimulate the DUT with synthesized keyboard, mouse, touchscreen, and button inputs. It also simulates the visual verification that an operator performs by capturing the display screens of the DUT and verifying the captured screens for location and content.

In a general sense TestQuest Pro can synthesize, to a high degree of fidelity, not only a “virtual operator” for the DUT but also its complete operating context. This capability allows the DUT to be tested exactly as it would be used in a regular deployment.

TestQuest Pro provides these capabilities through standard stimulation modules for keyboard, mouse, touchscreen, keypad, discrete I/O, RS232, etc. and standard monitoring modules for VGA, Raster LCD, Command LCD, discrete I/O, RS232, etc. A script development and execution environment controls these modules and provides access to the complete verification API of the test tool.

In order to validate the functionality of the gas process controller, TestQuest Pro was integrated with its human and process I/O points. The touch panel and keypad were instrumented for stimulation and the digital LCD was instrumented for monitoring. In addition, to fully simulate the operating environment of the controller and to provide a fully closed-loop, automated test solution, the valve connectors were instrumented to monitor their status and the sensor lines instrumented to simulate different pressures. TestQuest Pro programmatically stimulates and monitors these connection points using integrated digital and analog monitoring and simulation lines. Finally, the configuration file communication channel using a bi-directional RS-232 was routed to the TestQuest Pro system.

The integrated combination of TestQuest Pro and the instrumented gas process controller provide complete functional testability for every possible configuration of the gas controller.



Test Strategy

The single most important requirement in any automated test solution is the maintability of the test scripts. There are two important aspects regarding the maintability of the gas process controller automated test solution. First off, how easy will it be to make changes without causing unintended effects? This is an important aspect to consider because of the inherent complexity associated with the gas process controller. In many cases when the application under test is complex, the automated test solution is overly complex. Functions are overloaded and the automated test solution loses conceptual clarity. In order to minimize complexity, test functions should be conceptually coherent.

The second factor that affects maintainability is the specific ability of a test suite to be modified when design changes are made in the system under test. There are two general strategies for minimizing the effort to keep a test suite easy to maintain in the face of design changes:

1) Functions must be written to test tasks. Most design changes will only require changes to these functions.

2) Create a model of the user interface as a set of objects. Changes to fonts, bitmaps and text can be handled simply by updating the object definition information.

However, how do you create a maintainable and flexible automated test solution when the system under test is completely configurable? Do you want to create every possible configuration and create static scripts for each of them? Alternatively, should you select a configuration subset, but which one? This is where known automated test methods can fail. In this case, the task created should be aware of the configuration and perform the necessary associated verifications based on information for the configuration file.

To create configurable automated test solution, the information contained in the configuration file can be used to gather expected results, trigger situational events and even created test cases automatically. Furthermore, the automated test solution can also be configured to automatically gather the configuration information and perform tests for the specified configuration. This means that every, or a subset of the alarm and response sequences could be exercised.

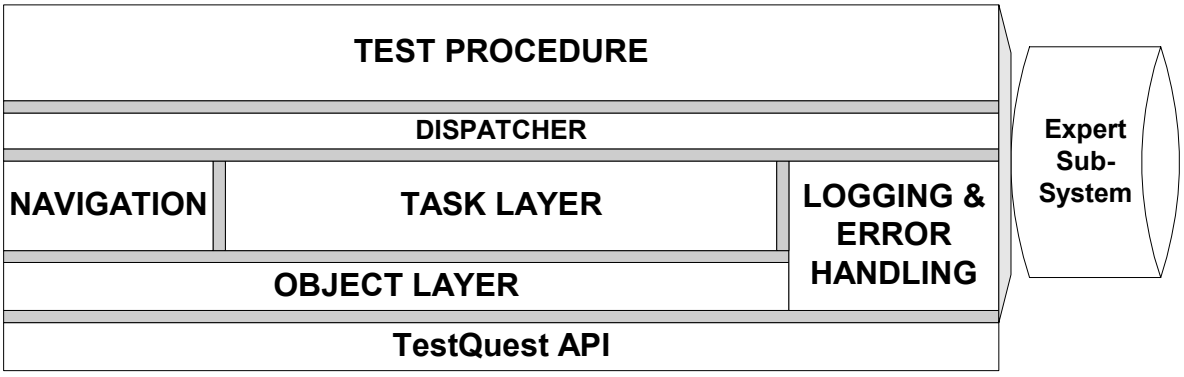
Test Automation Architecture

The test architecture used for the automated test solution is composed of three basic layers as shown in the figure below:

- Test Procedure Layer
- Task Layer
- Object Layer

Test Procedures Level

The test procedure scripts correspond to defined test cases as closely as possible. They are specified to a similar level of detail and are contained in simple script files.



- Roughly corresponds with the manual test descriptions in terms of detail and specificity.
- Can be read by anyone: test technicians, managers, developers, etc.
- Includes calls to task level functions and object methods.

Task Layer

Tasks are common sequences of actions that often appear repeatedly in the tests. They may take place on a single screen or span a couple screens, but usually do involve multiple objects. Tasks may also trigger events and verify that the appropriate responses are performed by the gas controller.

- Tasks will be exposed as keyword functions
- Tasks are composed of calls to other tasks and/or object level functions.
- They represent sequences of events.

Object Layer

The object layer is an abstraction that encapsulates all references to screen coordinates fonts and bitmaps. Neither tasks nor test procedures shall refer directly to the TestQuest core API functions, which supports this lower level of functionality.

The object layer is composed of the function interfaces for the objects and screens as well as the object definitions themselves. Object definitions are separately

specified and can be updated, revised or modified without requiring changes to the code itself.

- Uses object definitions
- Standard functional interfaces to objects
- Implicit verification
- Screen Based
- Includes testability workaround if needed.

Dispatcher

The dispatcher wraps the code that will be present in all the task code. The dispatcher is used to hide the error handling functionality from non-technical users. The dispatcher code is also responsible for error recovery as well as error cleanup.

Expert Sub-System

The expert sub-system will be used to parse the gas controller configuration file and extract the information required to test it. The Task layer will use the information gathered from the configuration file to identify the verification process and expected result of a specific operation.

Architecture Example

Let's take the implementation of the following example:

> "Start a Gas Process and verify that the process is started properly."
> "Generate Fault Alarm and verify appropriate alarm sequence is generated"
> "Stop the Gas Process"

The test procedure layer would look similar to the following:

```
ConfigureGasController("myconfigfile.gcf");

StartCase("VMB7.2.1");
    StartGasProcess();
    GenerateFault();
    StopGasProcess();
EndCase();
```

The script would consist of sequential commands, free of complex decision statements. If looping were required, it would be implemented using statements that are as close as possible to natural language.

In this example, the first command `ConfigureGasController`, would send the specified configuration file to the gas controller, verify that it is configured properly

and setup the Expert Sub-System. If an error is generated during the configuration process, the error handler will attempt to recover from the error. If unsuccessful, the test will be aborted.

Next, the test case will be bounded by a call to `StartCase and EndCase`, which will be used to identify the start and the end of a test case. These commands will also setup the logs and trap failures that may have occurred within the tasks that were executed.

Each task in the example (`StartGasProcess`, `GenerateFault` and `StopGasProcess`) will trigger an event and verify that the gas controller control software takes the expected actions. This functionality will closely emulate how a real test technician would perform the operation. In the case of `StartGasProcess` the following would be performed:

- Enter the gas controller UI menu.
- Enter password.
- Navigate to and select start process.
- Verify that the sequence of events associated with starting the process occurs as specified in the configuration file.

Conclusion

We have described a highly configurable process control application and outline the challenges it poses from a test automation perspective. We have described a commercial functional test automation tool which is designed to interface to and test complex configurable systems. We have also described a test automation framework and architecture which makes it possible to test the multiple configurations that such a target system can assume.

QW2001 Paper 3V1

Mr. Christian Hote
(PolySpace Technologies)

Bug Detection Tools for Productivity

Key Points

- Efficiency: PolySpace Verifier turns detection of run-time errors on its head thanks to its unique technology. It detects run-time errors at compilation time and thus before starting tests.
- Quality: PolySpace identifies and checks each possible source of run-time errors of your software application against all possible behaviors and input values. Thanks to our technology, we can figure out the future of software applications and predicts where it will crash or give erroneous computation due to run-time errors.
- Productivity: PolySpace Verifier does not requires human help to run. It has a very short learning curve and requires a very small amount of time to setup. It is a non-intrusive software product that you can plug-in without modifying your developement process.

Presentation Abstract

Presentation abstract to be supplied.

About the Author

Christian HOTE obtained his PhD in Physics in 1991. He joined Verilog (European leader CASE Tools provider) as product manager and participated in several European Research programs (Eureka, Esprit) for embedded systems design and development. He joined PolySpace Technologies at its creation and manages US business development and operations.

QW2001 Paper 3V2

Ms. Peggy Fouts
(Compuware)

Test Planning For Xtreme Times

Key Points

- Planning for testing is important
- It is possible for web-time to accommodate test planning
- Planning includes testing activities and the associated environment(s)

Presentation Abstract

Many organizations are implementing new web-based applications or “webifying” existing applications. Up front investment in planning can have a significant impact on the success of the testing efforts and ultimately the success of the implementation. This is independent of the development methodology employed.

Some feel that implementation speed dictates that we should skip planning and move directly into test execution and deployment. However, investing time in planning allows more efficient and predictable execution activities that will contribute to meeting the time constraints. In addition, good planning of the initial effort should allow reuse of the deliverables for future projects.

About the Author

Peggy Fouts is a Senior QA Specialist for Compuware Corporation. She is employed as a consultant in Quality Assurance and Testing Solutions services for the Minneapolis Professional Services branch has been involved in the software industry for over 25 years. She serves both as member of the Compuware corporate-wide planning group for Quality Assurance products and services and is also involved in the development of the internal training for Compuware's QA and test personnel.

Test Planning for Xtreme Times

CompuwareCorporation

Quality Week 2001

Presented by Peggy Fouts
Compuware Corporation

Gen. Dwight D. Eisenhower on his assessment of the invasion of Normandy. "The plan is nothing, the planning is everything."

CompuwareCorporation

Test Planning

- The act of planning is valuable
- A plan helps the team to know what to do and when

But there's no TIME to plan in our fast-paced environment!

Considerations for Web environment

- Speed (WEB time, E-time, Xtreme…)
- Incremental deliveries recommended
- Set up automated regression
- Automate load, performance, site integrity
- What about test environments?
  - Must be efficient
  - Must be available
  - Must be accurate representations of production

What do I need to know to plan?

- Business goals
- Requirements (business and technical) and their priorities
- Dependencies
- Acceptance criteria
- Time allotted from requirement specification to deployment
- Personnel available for test and their skill sets
- Application type

Then what?

- Quickly develop a Test Strategy
- Quickly develop Test Plans
- Start on test cases as soon as possible

Test Strategy

- If possible modify one from another project
- What types of testing will be performed?
- Should some testing be outsourced to a lab?
  - Performance
  - Site integrity
  - Load
- Should some testing be performed by specialists - e.g. security
- Should some testing be automated?

Outsourcing

- They have all the standard hardware platforms, browsers, operating systems, etc.
- Financial impact in trying to reproduce that onsite and in rerunning test cases on different platforms
- Remote testing of web sites on a subscription basis
- Specialty testing - e.g. security

Automating

- Decide what to automate
- Requires tool purchase and training

Test Plans

- Create one for each type of testing
- If possible, reuse old ones
- Base them on the identified risks and priorities
- Make sure that testing can report reliably on the quality of the application

Test Environment Plan I

- Roles and Responsibilities
- Environment verification – How to verify that what you think is there is actually there
- Security and controls – Who has access, how is it applied
- Applications and interfaces to be maintained in sync with production
- Tools in use for environment maintenance and testing support
- Hardware

Test Environment Plan II

- Restoration/refresh processes
- Backup plans
- Promotion from development environment to test environment
- Data (files, databases, etc.)
- Archiving artifacts

Summary

- A planning cycle provides the opportunity to ask questions for understanding
- Ask questions to cover the items presented
- What your documents look like is less important than that they provide the direction to cover the key items - risks and priorities

Questions

Test Planning for Xtreme times

Many organizations are implementing new web-based applications or "webifying" existing applications. Up front investment in planning can have a significant impact on the success of the testing efforts and ultimately the success of the implementation. This is independent of the development methodology employed.

Some feel that implementation speed dictates that we should skip planning and move directly into test execution and deployment. However, investing time in planning allows more efficient and predictable execution activities that will contribute to meeting the time constraints. In addition, good planning of the initial effort should allow reuse of the deliverables for future projects.

1 Test Planning

Planning revolves around what has to be tested and how it has to be tested. Testing is constrained by the acceptance criteria and the time allotted. Although we would like to think that we have control over schedule we often do not. So we need to do the best with what we have – clearly describe the associated risks and make recommendations related to the testing activities. Focus planning on managing risks and understanding deployment risks as a result of the testing activities.

1.1 The act of planning has value in itself.

It's what you learn from the plan that's important.
The act of planning implies that we will be communicating with stakeholders, documenting their desires and figuring out how to please everybody - manage risk. It implies that we are thinking of what needs to happen and checking your ideas against those of others (similar to peer review – better that working in a vacuum and forging ahead on our own).
Get it down on paper, maybe it will change later, but …

1.2 The plan helps team members know what to do (and what is being done) and when.

It is a communication vehicle. Project failure can often be traced back to communication problems. It is important for the whole team to have a consistent view of the project activities.

This plan should be more or less detailed based on the size of the team and the complexity of the testing requirements. If the entire test and development team is five co-located people, we can probably communicate face to face and might only create a checklist for a plan. On the other hand, if the team is ten people and requires a complicated test environment in order to simulate production the Test Plan might be quite a bit more detailed.

1.3 Considerations for a WEB-based implementation

These are items that "direct" (constrain) the testing activities for Web implementations. Items two through four can contribute substantially to fast

turnaround – support of item one, and can be implemented with just a little bit of planning.

1.3.1 Speed (WEB time, E-time, Xtreme…)

Initial implementations are often rolled out in record time, and any changes to the production application often must be made "immediately" if the user is impacted.

1.3.2 Incremental deliveries recommended

The prioritization of requirements and incremental deliveries of feature sets as they are available facilitate fast rollout. These deliveries might be to test only or might be scheduled to move from test into production.

1.3.3 Set up automated regression testing

Most "light" – agile – methodologies recommend the automation of both unit and system testing. These tests are run frequently and are required before the application is put into production. Automation is a necessity to meet these criteria.

1.3.4 Automate load, performance, site integrity testing

To facilitate fast turnaround these tests should also be automated or outsourced to a remote lab – which is covered later.

1.3.5 What about the test environment(s)?

The test environment setup and maintenance is a major factor in an efficient, successful test effort.

- Must be efficient
- Must be available
- Must be accurate representation(s) of production
- Must be "secure" and controlled

Good planning and up front implementation of a test environment that meets these criteria is essential. If you skip anything, don't skip this.

2 What do I need to know to plan?

1. Business goals
2. Requirements (business and technical) and their priorities
3. Dependencies
4. Acceptance criteria
5. Time allotted from requirement specification to deployment
6. Personnel available for test and their skill sets
7. Application type

Each of these items has a definite impact on how to conduct the testing effort. If the business goals include to be first on the web with a particular type of application, testing priorities will be different from the case where the business goal is primarily to provide a better user experience than the competition.

These goals should also affect the priorities assigned to requirements and the acceptance criteria. The acceptance criteria should capture all critical requirements.

The total time allotted and the time between deliveries drive the automation decisions. Incremental deliveries with short time spans between each delivery indicate that we should automate as much as possible.

3 What do I do with what I know?

1. Quickly develop a Test Strategy based on 1 through 7 above
2. Quickly develop Test Plan(s) – if possible, reuse old ones
3. Start on Test Cases as soon as possible, but preferably after strategy has been approved

3.1 Test Strategy

If possible, modify a Test Strategy completed for another project. This implies that there is an asset repository where assets are readily available for modification and reuse

At least have standard template – it may not look like IEEE or…: it may look more like a checklist. The goal is to convey to all team members what testing will be performed, how it will be performed and why – so that changes downstream can be weighed against original goals.

The strategy will be focused on risks. So identify and address risks and gain consensus on the overall test approach. The strategy should address the risks of most concern.

Devise new test techniques and methods that address the particular constraints of the technologies – like rapid turnaround requirements. It is imperative that the strategy make best use of test automation throughout the test effort.

Make sure the testing results will provide thorough test evidence to help stakeholders to make the correct release decision. This implies that the tests are comprehensive and that the results will be available for review. The test cases may be executed many times, so the test management framework should allow for each of the testing cycles to be reviewed. (Sometimes it is important to know whether or not a particular test passed on the previous iteration.)

All testing strategy decisions are made based on business goals and accepted priorities.

3.1.1 Outsourcing

Should some testing be outsourced to a lab?
One possibility to consider for web applications is to outsource some of the testing to a testing lab. They are set up to provide all of the standard hardware platforms, browsers, operating systems, etc. There is a significant financial impact in trying to reproduce that onsite and in rerunning test cases on different platforms. So, if the test strategy dictates that this environment is necessary, it might be worthwhile investigating outsourcing to a testing lab.

Remote testing of web sites is also available on a subscription basis. This approach allows those with the specialized expertise to handle the testing removes the necessity to purchase automated testing tools. These services cover a wide variety of options. For instance, some may monitor the performance of customer selected web transactions using load testing and network diagnostic technology. The automated monitoring occurs on a regular (perhaps hourly) basis, 24 hours a day, 7 days a week or on demand
.
In addition to performance testing they also check the integrity of the entire website regularly. The scans search for common website problems, ranging from broken links, to page warnings and image catalog problems. The integrity check provides summary and detailed information in the areas of accessibility, usability and functionality of the customer's site. Results of the performance monitoring are delivered daily or on demand. The performance trend summaries and integrity check results might be delivered weekly or as desired.

Should some testing be performed by specialists – e.g. security?
How secure does the site have to be? Are we handling confidential information? All major products on the market, including browsers, operating system software, networking software, web server and development products have security vulnerabilities. Published vulnerabilities have countermeasures or patches provided by the software suppliers, but local system administrators have the responsibility to stay up to date and apply the patches. There are standard ways to test security that require in-depth technical knowledge of the systems under attack, the tools available to assist and the approach to crack into systems. It is a highly specialized skill and you may want to hire or outsource to specialists to perform security testing.

3.1.2 Automation

Should some testing be automated?
In the Xtreme methodology, all tests are automated. There are two levels of test - unit and acceptance (system). These tests are also used for regression testing. Xtreme programming is based on providing incremental releases. This methodology implies that there are frequent integrations, so tests are executed every day – maybe more than once a day. They can verify the results of small changes that occur frequently.

During the life of a project an automated test can save you a hundred times the cost to create it. It provides an efficient way to detect and guard against bugs. Automated unit tests offer a payback far greater than the cost of creation.

Load tests for web applications should be automated in order to replicate the user load more efficiently, and to allow for frequent execution of the tests after changes to the site.

If automating test cases, the test team may require training on the use of the tools.

3.2 Test Plans

It might be necessary to create a Test Plan for each type of testing that was referenced in the Test Strategy. Again if possible, modify Test Plans completed for other projects. Make them as simple as possible, but be sure to cover all risks and all high priority requirements. In addition, make sure that planned testing can report reliably on the quality of the application based on the risks and priorities.

4 Test Environment Preparation

4.1 General

A stable environment is the most critical piece of the test process. An Environment Setup and Maintenance Plan should be in place. It should cover at least the following topics:

- Roles and Responsibilities
- Environment verification – How to verify that what you think is there is actually there
- Security and controls – Who has access, how is it applied
- Applications and interfaces to be maintained in sync with production
- Tools in use for environment maintenance and testing support
- Hardware
- Restoration/refresh processes
- Backup plans

The following may be specific to the application being tested, but the Environment Setup and Maintenance Plan should cover them in general terms.

- Promotion from development environment to test environment
- Data (files, databases, etc.)
- Archiving – Items from the environment that require saving following completion of a test cycle, e.g. automated test logs and data recovery restart and checkpoints

Once this plan is created it can most likely be reused for future test efforts.

5 One approach

This is a sample activity/responsibility list for an organization that has an enterprise wide test environment that is used for testing. Access to this environment must be scheduled in advance. For an Xtreme approach, it is probably more likely that the test environment is constantly available to the team. However, a lot of the tasks still apply.

Test Planning and Preliminary Setup Activities		
Activities	**Tasks**	**Responsibilities**

Definition and Planning	Create a Test Plan that includes: ◦ testing objectives, ◦ scope, ◦ approach, ◦ procedures, ◦ products required, ◦ team responsibilities, and ◦ schedule. Review and approve the Test Plan with appropriate application development management. Create Test Environment Requirements document that includes: ◦ application or system software requirements, ◦ hardware configurations ◦ data requirements ◦ backup and restoration procedures ◦ environment verification processes. Deliver Test Environment Requirements document to the Test Environment Manager. Review Test Environment prerequisites for entry, procedures related to test time scheduling, test suspension and test resumption, set up of and removal of data and infrastructure support. Schedule entry into Test Lab.	Application team with support from Environment team Application management Application team with support from Environment team Application team Application team Application team and Environment team
Preparation and Validation	Generate tests and procedures and assemble test data ◦ Include testing of interfaces to other systems, applications, or subsystems as required. Communicate the satisfactory completion of prior system testing to the Test Environment Manger. Test Environment Manager enables physical access to the lab for the application test team.	Application team Application team Environment Manager

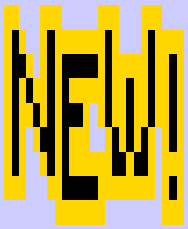
Test Planning and Preliminary Setup Activities		
Activities	**Tasks**	**Responsibilities**
Definition and Planning	Create a Test Plan that includes: ◦ testing objectives, ◦ scope, ◦ approach, ◦ procedures, ◦ products required, ◦ team responsibilities, and ◦ schedule.	Application team with support from Environment team
	Review and approve the Test Plan with appropriate application development management.	Application management
	Create Test Environment Requirements document that includes: ◦ application or system software requirements, ◦ hardware configurations ◦ data requirements ◦ backup and restoration procedures ◦ environment verification processes.	Application team with support from Environment team
	Deliver Test Environment Requirements document to the Test Environment Manager.	Application team
	Review Test Environment prerequisites for entry, procedures related to test time scheduling, test suspension and test resumption, set up of and removal of data and infrastructure support.	Application team
	Schedule entry into Test Lab.	Application team and Environment team
Preparation and Validation	Generate tests and procedures and assemble test data	Application team
	Establish and verify the test application in the Test Environment (consider access, platforms, operating systems, databases and networking) that will simulate production.	Environment Manager
	Load new versions of system software or application into the testing environment.	Application team with Environment

Test Planning and Preliminary Setup Activities		
Activities	**Tasks**	**Responsibilities**
Definition and Planning	Create a Test Plan that includes: ◦ testing objectives, ◦ scope, ◦ approach, ◦ procedures, ◦ products required, ◦ team responsibilities, and ◦ schedule.	Application team with support from Environment team
	Review and approve the Test Plan with appropriate application development management.	Application management
	Create Test Environment Requirements document that includes: ◦ application or system software requirements, ◦ hardware configurations ◦ data requirements ◦ backup and restoration procedures ◦ environment verification processes.	Application team with support from Environment team
	Deliver Test Environment Requirements document to the Test Environment Manager.	Application team
	Review Test Environment prerequisites for entry, procedures related to test time scheduling, test suspension and test resumption, set up of and removal of data and infrastructure support.	Application team
	Schedule entry into Test Lab.	Application team and Environment team
Preparation and Validation	Generate tests and procedures and assemble test data	Application team
	Set up PCs or other terminals for Tester use.	team support Lab team
	Ensure appropriate lab support, such that all required testing could be accomplished.	Environment Manager
	Define details for test sessions and related schedules.	Application team with Environment

Test Planning and Preliminary Setup Activities		
Activities	**Tasks**	**Responsibilities**
Definition and Planning	Create a Test Plan that includes: ◦ testing objectives, ◦ scope, ◦ approach, ◦ procedures, ◦ products required, ◦ team responsibilities, and ◦ schedule.	Application team with support from Environment team
	Review and approve the Test Plan with appropriate application development management.	Application management
	Create Test Environment Requirements document that includes: ◦ application or system software requirements, ◦ hardware configurations ◦ data requirements ◦ backup and restoration procedures ◦ environment verification processes.	Application team with support from Environment team
	Deliver Test Environment Requirements document to the Test Environment Manager.	Application team
	Review Test Environment prerequisites for entry, procedures related to test time scheduling, test suspension and test resumption, set up of and removal of data and infrastructure support.	Application team
	Schedule entry into Test Lab.	Application team and Environment team
Preparation and Validation	Generate tests and procedures and assemble test data	Application team team support
	Conduct a pre-test review with the test team and the Test Lab Manager.	Application team and Environment Manager

6 Summary

If we go through a planning cycle and cover the items presented here, we will learn what we need to know to drive a successful development and testing effort. What the documents look like is less important than that they provide the direction to cover the key items - risks and priorities.



QW2001 Paper 4V1

Mr. John Keller
(TeamShare, Inc.)

Collaborative Product Design and the New Role of Testing

Key Points

- The Collaborative Product Design (CPD) methodology is essential in today's economic climate because companies can no longer afford to build products that miss the mark.
- The role of testing in a CPD paradigm expands beyond simply "Does it work" to "Is it what the customer is looking for."
- When it comes to building a CPD development environment, technology isn't everything, but it is a critical enabler.

Presentation Abstract

Collaborative Product Design (CPD) is a relative newcomer to development methodologies, but it builds on foundational principles found in other more well-known development paradigms including Extreme Programming, Customer Relationship Management, and Outward Focused corporations. CPD is essentially all about creating a collaborative environment where customers, partners and new product development groups can join together to build products and services that have a rapid time to market, and to market embrace. CPD is in direct response to the economic climate of today, where gone is the mentality of "if we build cool technology, they will come", and in its place is the mentality "we're not going to build it if there aren't customers who will buy into it."

Historically, testing organizations focused attention on isolating defects in products simply in terms of finding features that are broken. In the CPD environment, the testing organization must be more intimately involved with specifically what customers are looking for, and build test plans that address features in terms of customer needs.

This presentation will discuss the concepts behind CPD and what it takes to build a CPD development environment.

About the Author

Speaker bio to be supplied.

Collaborative Product Design and the New Role of Testing

John Keller

Product Business Manager

TeamShare, Inc.

Quality Week 2001

Agenda

- How today's economy sets the stage
- What it's all about and where it came from
- The new role of testing
- What it takes to build a CPD environment

Today's Market Conditions

In these economic conditions, companies cannot afford to build technology that is simply cool, but that has no viable market.

Motorola cuts 4000 jobs to remain competitive in a slowing economy - cuts are in Motorola's networks sector created to provide broadband & wireless communications products and systems.

3Com said a $1 billion restructuring plan includes the discontinuation of its consumer Internet appliance line to help return it to profitability.

The World as We Know It

- **70% of overall product cost is in the design stage[1]**
- **Customer profitability increases each year that the customer is retained, from 2% to 14%[2]**
- **Customers don't know what they want up front**
- **CPD is how we cut costs and still innovate**

[1] Beating the Competition with Collaborative Product Commerce, Leveraging the Internet for New Product Innovation, Aberdeen Group, Inc., October 2000 Update, page 4

[2] Customer Relationship Management: A Strategic Imperative in the world of e-Business by Stanley A. Brown, page 15

CPD – Aberdeen Market Definition

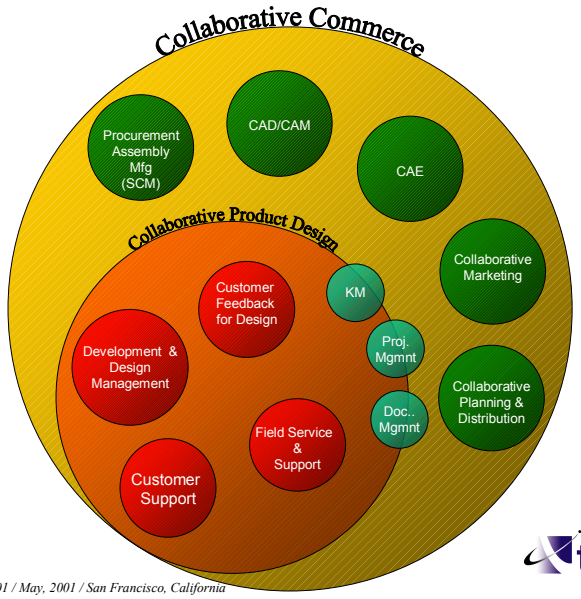
- **Aberdeen defines CPD as a *suite of software* and services that integrates several product-centric business processes across multiple independent enterprises into a *single, closed-loop solution*.**
- **CPD solutions are inherently *web-architected* and make extensive use of collaboration technologies to focus *disparate organizations* on a common task: producing the best designed, most advanced products possible.**

CPD - Business Needs

- **Organizations compete by *developing better products* in *less time*, at *less cost*, and with *fewer defects* than rivals.**
- **Customers demand *relevant, current products* that are delivered in a prompt and efficient manner.**
- **Enterprises are looking to generate a product that is *custom-tailored to customers' needs*. Decision-makers are being pressured to move from the traditional *make-to-stock* model to a *build-to-demand* model.** [4]

[4] Beating the Competition with Collaborative Product Commerce, Aberdeen Group, October 2000 Update, page 1

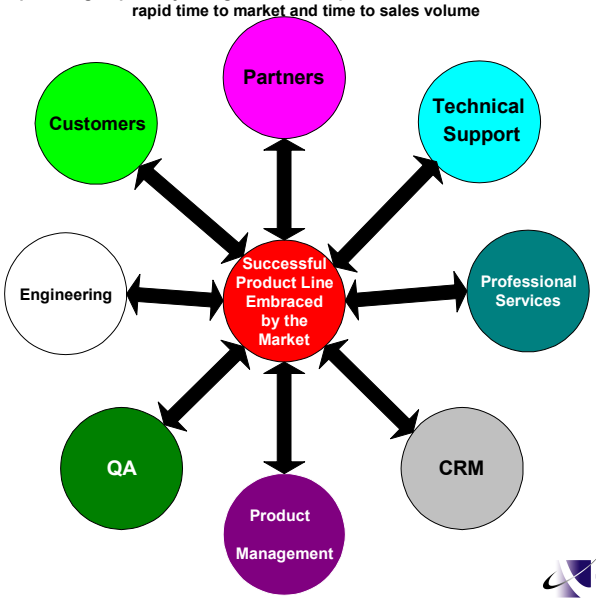
CPD – A Part of C-Commerce



teamshare®

Collaborative Product Design

Creating a collaborative environment where customers, partners and new product groups can join together to build products and services that have a rapid time to market and time to sales volume



teamshare®

CPD Benefits

- **Better products built through collaboration, driven by customer demand, developed in less time, at less cost, and with fewer defects than competitors**
- **Customer-focused development projects with an ear to the real world**
- **Long-term customer loyalty and profitability through products that continue to meet and exceed market demands**
- **"Communication + Collaboration = Innovation"**

Palm Source 2000

CPD and the New Role of Testing

- **Old School: "Find what's broken"**
- **New School: "Find the areas that aren't what the customer wanted"**
- **Test plans need to be customer focused**
- **Analyze the product and the market**
- **Success is not just measured in "time to market", but in "time to volume"**

What it takes to build a CPD Environment - 1

- Outward focused way of doing business
- No silos
- Expand product knowledge outside of development / QA walls
- Direct customer input and feedback

What it takes to build a CPD Environment - 2

- Web-based, platform independent design allowing universal accessibility
- Workflow architecture allowing intra- and inter-enterprise collaboration
- Built-in security minimizing risk of outside user break-in
- Common access to a virtual data repository, with data served up in a form applicable to users' roles

What it takes to build a CPD Environment - 3

- Integrated Development and Support solution that facilitates collaboration between key technical groups
- Licenses which allow direct customer involvement
- XML integrations which enable cross application data exchange
- Hosting option allowing rapid deployment

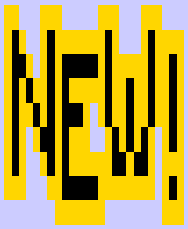
CPD and the New Role of Testing

For more information, contact:

John Keller, Product Business Manager,
TeamShare, Inc.

(719) 457-8884

john.keller@teamshare.com



QW2001 Paper 4V2

Mr. Michael Smith
(McCabe & Assoc)

QW2001 Keynote 5P1

Dr. Linda Rosenberg
(GSFC NASA)

Independent Verification And Validation Implementation At NASA

Key Points

- IV&V is a valuable tool for increasing sofware quality and reliability
- IV&V needs selection criteria for which projects should apply it
- NASA is now requiring IV&V evaluation for all projects and application on some

Presentation Abstract

This paper will discuss the management approach taken to consistently apply IV&V on NASA projects. We will discuss the development of the NASA policy stating that all projects and programs with few exceptions shall employ software V&V techniques for risk mitigation; criteria shall be applied to determined if IV&V is warranted. The IV&V selection criteria are based on determining the extent of risk. Developed by an Agency wide group, the criteria provide an algorithm for estimating probability of software failure based on quantitative software complexity, development team, management, and other factors. Factors for characterizing the consequences of software failure are determined, then thresholds for performing IV&V based on probability of failure and severity of impact (consequences) are applied. In this paper, we will discuss how these guidelines were developed and currently being applied. We will conclude with a discussion on the approach to projects, how do you convince the project manager to implement IV&V, and what is the cost to the project.

About the Author

Dr. Rosenberg is the division chief of the Software Assurance Technology Office (SATO) in the Office of Systems Safety and Mission Assurance Directorate at Goddard Space Flight Center (GSFC), NASA. She also serves as the Software Assurance technologist for GSFC and NASA. Dr. Rosenberg is a recognized International expert in the areas of software assurance, software metrics, requirements and reliability. She serves on IEEE program committees for software reliability, software metrics, and software requirements. She has chaired sessions at many international conferences, included those sponsored by NASA and AIAA. Dr. Rosenberg also reviews papers for the Department of Defense sponsored conferences, and other industrial organizations for software quality. She has presented papers and tutorials in many areas of software assurance, including reliability, at IEEE and ACM international conferences and the International Astronautical Congress. Dr. Rosenberg is currentl! y on the steering committee to

evaluate the metrics in the IEEE 982 Reliability Standard. Dr. Rosenberg is also an adjunct professor at University of Maryland, Baltimore for the Masters/Doctoral Program.

Dr. Rosenberg holds a Ph.D. in Computer Science from the University of Maryland, an M.E.S. in Computer Science from Loyola College, and a B.S. in Mathematics from Towson University. She is a member of Electrical and Electronic Engineers (IEEE), the IEEE Computer Society, the Association for Computing Machinery (ACM) and Upsilon Pi Epsilon.

Independent Verification and Validation Implementation at NASA

Linda H. Rosenberg, Ph.D.
Software Assurance Technology Office, Code 302
Goddard Space Flight Center, NASA
Greenbelt, MD 20771 USA
301-286-0087
Linda.Rosenberg@gsfc.nasa.gov

ABSTRACT
NASA is recognized as a leader in space technology, with cutting edge science probing galaxies never before seen by mankind. In keeping with this cutting edge technology, much of the functionality previously done through hardware has transferred to software, including mission critical functions. But technology implementation is moving so fast, that at times quality assurance cannot keep up, although we try. The NASA Independent Verification and Validation (IV&V) Facility was established in 1993 in West Virginia and tasked to provide the highest achievable levels of safety and cost effectiveness for mission-critical software. Despite this, NASA has experienced some failures recently that traced, in part, to less than adequate implementation of mission software. In response in part to these failures, NASA has taken steps to place an emphasis on implementing improvements in the Agency's software process. One of these steps relates to the increased focus on IV&V.

Every industry has data to support the statement that IV&V is cost effective, a very positive return on investment, yet most projects don't employ it. NASA decided to investigate the current use of IV&V, identify projects that should be using IV&V, and start using the expertise of the NASA IV&V Facility to improve the quality assurance on NASA projects through the appropriate implementation of IV&V. This paper is not about IV&V, but about NASA's new approach to the implementation of IV&V on all software development throughout the Agency.

Keywords
Independent Verification and Validation, IV&V, risk mitigation

1 INTRODUCTION

NASA determined the need for software Independent Verification and Validation (IV&V) after evaluating the causes of recent mission failures. These failures were due, in part, to software issues that should have been identified during development or testing. The NASA IV&V Facility in West Virginia was developed to be a Center of Excellence, but was under utilized. Projects that did use the Facility had proven benefits. The focus for improvement within NASA has turned to the application of Independent Verification and Validation of the software.

This approach started by looking at the resources available and projects that have applied IV&Vto determine if there were benefits in the NASA environment. It was quickly determined that only a few projects were implementing IV&V, not all were taking advantage of the Facility's expertise, and there were very definite proven benefits to NASA when IV&V was applied. These benefits ranged from cost savings to identifying mission critical errors not previously identified through testing; the application of IV&V on software resulted in increased safety and reliability of the mission. But deficiencies in the application of IV&V were also identified, specifically it was randomly applied by projects, with no consistency.

This paper will discuss the approach taken to increase the use of IV&V within NASA. We will start by defining independent, verification, and validation, what they mean. We will then discuss the policy written relative to the performance of software IV&V and the criteria developed to help projects quantify the need for IV&V. We will conclude with a discussion on the approach and how to convince the project manager to implement IV&V.

2 INDEPENDENT VERIFICATION AND VALIDATION (IV&V)

Although this is not a paper on IV&V, we do need to start with a basic understanding of what constitutes IV&V, starting first with the definitions of verification and validation, then determining what is required for "I" – independence. IEEE 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, defines verification and validation as follows.[1] Verification is defined as the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Validation is defined as the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. Another way of stating verification is "Did we built the system right?" and validation as "Did we build the right system?" IV&V is implement at all phases of the software development life cycle, not just in testing.

Independence in IV&V is defined by IEEE as three parameters: technical independence, managerial independence, and financial independence. Technical independence is achieved by personnel who are not involved in the development of the software. IV&V personnel use their expertise to assess development processes and products independent of the developer. They formulate their own understanding of potential problems and how the proposed system is solving them. Managerial independence requires responsibility for the IV&V effort to be vested in an organization separate from the organization responsible for performing the implementation of the system. The IV&V effort independently selects the segments of the software and system to analyze and test, chooses the IV&V techniques, defines the schedule of IV&V activities, and selects the specific technical issues and problems to act upon. Most projects view V&V sufficient and do not recognize the added value the independence brings. Finance independence has been harder to attain. All work on the project, including quality assurance, is funded directly by the project, hence, IV&V is also funded directly by the project. In theory, the project could remove IV&V funding if they are not satisfied with the findings, but with the implementation of the IV&V policy, the projects are now required to work with the IV&V Facility to reach an agreement on the amount of IV&V and funding.

Any changes to this must be agreed to by the Center Director with strong justification by the project manager.

3 IMPLEMENTATION APPROACH

All software project managers, whether government or industry, never have time or money to spare, so when NASA identified the need for IV&V, it was met with a cry of "Not on my project!" But the implementation of IV&V was part of a larger effort to improve the software developed at NASA, to achieve the highest levels of safety and cost effectiveness possible for mission critical software. To accomplish this, NASA established a group of software experts from all NASA Centers to advise and develop software policies and standards.[1] Each Center provided two experts in different aspects of software development to form the NASA Software Working Group (SWG). Their charter is to advise the Agency on software related matters and recommend software management, engineering and assurance policies, standards, best practices and guidance, quite a task. They have been instrumental in the implementation of IV&V at NASA as will be demonstrated in this paper.

In 1993, NASA established the IV&V Facility in Fairmont West Virginia to provide the highest achievable levels of safety and cost-effectiveness for mission-critical software. The mission of the IV&V Facility is to become the NASA Center of Expertise for the application of software verification and validation technology to the development of high quality, highly reliable software systems to support NASA missions. The IV&V Facility provides tailored technical, project management and financial analyses for NASA projects, industry, and other Government agencies, by applying software engineering "best practices" to evaluate software risk and criticality assessments throughout the system development life cycle. Through the Facility, NASA has the means for implementing IV&V, but it is under utilized. Only a few very large NASA projects, such as Space Station, chose to apply IV&V through the Facility. A few other projects applied IV&V but through another source, usually a contractor.

NASA now has the ability to implement software improvement throughout the Agency through the Office of the Chief Engineer. It has the body of software experts to write policy through the Software Working Group. It has the experts in IV&V at the Facility. The infrastructure is in place, now the process had to begin.

The first step was to write a policy requiring projects to investigate the necessity of doing IV&V. IV&V is an effective risk mitigation strategy, and since most NASA missions are cutting edge technology, they also are at high risk. But NASA develops many types of projects, from ground and flight systems, to instruments and data collection systems. The specific types of missions potentially requiring IV&V had to be identified. Recognizing that cost must be balanced against potential benefits, the "amount" of risk incurred by a project had to be calculated. In order to require projects to investigate the application of IV&V, a policy was developed identifying the types of projects that must potentially apply IV&V was developed. The process of determining the need for IV&V, the extent and approach are

[1] The author is Goddard Space Flight Center's primary representative to this group.

specified in the policy. The policy also states all IV&V will be done under the management of the NASA IV&V Facility, centralizing expertise and ensuring consistency.

The next step was to develop criteria for determining when IV&V must be considered - quantifying project risk for an initial assessment on which projects may require IV&V. Project risk is defined as a combination of the probability that an undesirable event will occur, and the consequence if the event does occur. The IV&V Criteria were written using probability and consequence. Factors influencing software development were identified, and risk factors associated with them for a calculation of the probability. Consequences of failure were classified as Grave, Substantial, Marginal, Insignificant. These are combined for a determination of the necessity of IV&V.

The final step, and the hardest in some respects, was to identify the projects that potentially required IV&V and determine to what level IV&V was needed. Money is always an issue, there is never enough in any software development, so what was the cost and what are the balancing benefits.

Therefore, the approach to implement IV&V consistently and logically on all NASA software was broken into 3 steps:

1 – Write a policy for the requirement of IV&V implementation
2 – Write the criteria for an initial determination of IV&V necessity based on project risk evaluation
3 – Work with projects to implement IV&V

Step 1 – IV&V Implementation Policy

The policy for IV&V implementation was to clearly specify the process of determining when a program must apply IV&V under the management of the NASA IV&V Facility. The initial version of the IV&V policy was stated in the IV&V Facility Business Plan in June 2000.[2] The final version of the policy will be distributed as a NASA policy by the end of 2001, giving it the authority of being required for all NASA software development.

One strength of the policy is the specification that the NASA IV&V Facility is responsible for the management of all software IV&V efforts within the Agency. The Facility's role is to provide a value-added service to the Agency's software development efforts. The cost to perform tailoring and implementation of IV&V is expected to be borne by the project or the sponsoring HQ Enterprise. All results are to be reported to the project manager and the cognizant Center Director. This creates a central repository of knowledge, tools, metrics and lessons learned that can be used to improve the IV&V efforts on future projects throughout NASA.

The policy identifies all projects that this policy pertains to as those that are covered in NASA Policy Guideline (NPG) 7120.5 "NASA Program and Project Management Processes and Requirements Highlight Code", or other projects within NASA with significant software effort as determined by the NASA Chief Information Officer (CIO), the NASA Office of the Chief Engineer (OCE), and the NASA Office of Safety and Mission Assurance (Code Q) or

Center Safety and Mission Assurance. Projects covered in NPG 7120.5 include "all programs/projects that provide aerospace products or capabilities, i.e., provide space and aeronautics, flight and ground systems, technologies, and operations."[3] This covers most of the projects but may exclude projects where NASA is a minor partner and those developed by Universities for educational purposes and considered at minor risk due to limited investment.

The policy states that each project must produce, document, and implement a plan that addresses the performance of V&V, and if appropriate, IV&V, over the life cycle of the software, from requirements through delivery and maintenance. The level of IV&V of software that is performed is based on the cost, size, complexity, life span, risk, and consequences of failure as defined using the Criteria (explained in the following section).

The policy specifies the following steps for IV&V determination and application:

a. The project manager is to evaluate their project against the criteria (described in Step 2) to determine if IV&V is indicated.
b. For projects where the criteria indicates software IV&V is warranted, the project manager is to discuss the results with a representative of the NASA IV&V Facility. Application of the IV&V criteria simply determines if a project is a candidate for IV&V – not the level of IV&V nor the resources associated with the IV&V. The Facility personnel will work jointly with the project office to provide recommendations tailored to the project on what sections and to what extent IV&V should be performed.
c. With this input from the Facility, the project manager will document in the project plan what IV&V is intended to be performed. Since IV&V compliments and enhances risk mitigation, projects are encouraged to achieve the most effective balance of risk mitigation strategies.
d. The level of IV&V activities selected in the project plan is subject to Facility review. The Center Director is responsible for resolving differences between the Facility and Project Office.
e. When IV&V is selected for a project, the Facility, with information from the project, will make the determination of which IV&V work will be done at the Facility location in WV, and which work will be done local to the project. The objective over the life of the project is that at least 50% of the work be done at the Facility.
f. When the project undergoes significant changes that impact the software subsystems, the project manager must revisit the criteria.

Step 2 - IV&V Criteria

In order to accomplish of goal of increasing the application of IV&V on all appropriate projects, it had to be determined what was meant by "appropriate" projects, hence the development of IV&V criteria. Again the members of the NASA Software Working Group were task with defining when IV&V should be applied to a project. Looking back on the objective of IV&V for risk mitigation, those projects with high risk had to be identified, but first, the criteria for what makes a software project high risk had to be defined and quantified. The quantification was the hardest part.

IV&V is intended to assist mitigating risk, hence, the decision to do IV&V must be risk based. NASA policy NPD 7120.5 defines risk as the "combination of 1) the probability (qualitative or quantitative) that a program or project will experience an undesired event such as cost overrun, schedule slippage, safety mishap, or failure to achieve a needed breakthrough; and 2) the consequences, impact, or severity of the undesired event were it to occur." [3] The exact probability of occurrence and consequences of a given software failure cannot be calculated early in the software lifecycle. However, there are realistically available metrics which give good general approximations of the consequences as well as the likelihood of failures.

Probability Evaluation

The probability of failure for software is difficult to determine at any phase in the software development life cycle. The NASA Software Working Group (SWG) has identified factors that impact the difficulty of the development. These factors were then calibrated to determine the extent of risk for successful software development. While the indicators are not precise and are currently in Beta testing by NASA software development projects and the IV&V determination team, they do provide order of magnitude estimates, which are adequate for assessing the need for IV&V. These factors are described below.

a. Software team complexity – Industry research has shown that the larger the team the more complex the communication and more points of failure. Smaller teams are generally co-located, making communication a common occurrence. The larger the development teams, the more formal communication, often loosing some subtle communication activities.
b. Contractor support – NASA software is developed by civil servants, contractors or a combination. The presence of a contractor (other than the prime system developer) introduces a layer of contract management between the government and the actual software developer.
c. Organization complexity – This is an indirect measure of communications challenges inherit in the software developer. A single organization working from multiple locations faces a slightly greater challenge than an organization in one location. When the software development is accomplished by multiple organizations working for a single integrator, the development is significantly complicated. If the developing organizations are coequal such as in an associate contractor relationship (or a similar relationship between government entities) then there is no integrator. Experience has shown this arrangement to be extremely challenging as no one is in charge.
d. Schedule pressure - A deadline is negotiable if changing the deadline is possible although it may result in slightly increased cost, schedule delays, or negative publicity. A deadline is non-negotiable if it is driven by immovable event such as an upcoming launch window. Project working under deadlines are more likely to decrease testing, hence potentially decreasing the reliability and safety, and increasing the risk.
e. Process maturity of software provider – It has been demonstrated in almost all cases, application of the Software Engineering Institute's Capability Maturity Model (CMM) increases the quality of the software developed, resulting in higher reliability. This model has 5 levels of attainment, 3 being the most common, and all projects start out at Level 1. While striving to attain Level 2, projects are implementing some of the

techniques that reduce project risk. The higher the CMM Level achieved, the lower the risks. A formal assessment is required to determine the CMM Level.

f. Degree of innovation – Innovation is usually found in most of NASA's software, as expected. But when attempting to formulate a new equation or concept, there is a higher risk than if the software has been proven in previous programs.
g. Level of integration – Programs that stand alone and are not integrated with any other components have a higher probability of success due to simplicity of their interfaces. The more extensive integration of multiple components, the higher the risk.
h. Requirement maturity – Requirements that are constantly change or change late, ambiguous or untestable are at a higher risk than those requirements whose objectives are well defined with few or no unknowns.
i. Software Lines of Code – It has been proven that the larger the project, the higher the risks. Lines of code is used as a size measurement and is defined to be all code in the programs, including software that is reused and auto generated code.

Five categories of risk within each factor were identified based on input from software developers from all NASA centers. Values 1, 2, 4, 8, 16 were assigned to each category.

Factors contributing to probability of software failure	**Un-weighted probability of failure score**					**Weighting Factor**	**Likely-hood of failure rating**
	1	2	4	8	16		
Software team complexity	Up to 5 people at one location	Up to 10 people at one location	Up to 20 people at one location or 10 people with external support	Up to 50 people at one location or 20 people with external support	More than 50 people at one location or 20 people with external support	X2	
Contractor Support	None	Contractor with minor tasks		Contractor with major tasks	Contractor with major tasks critical to project success	X2	
Organization Complexity*	One location	Two locations but same reporting chain	Multiple locations but same reporting chain	Multiple providers with prime sub relationship	Multiple providers with associate relationship	X1	
Schedule Pressure**	No deadline		Deadline is negotiable		Non-negotiable deadline	X2	
Process Maturity of Software Provider	Independent assessment of Capability Maturity Model (CMM) Level 4, 5	Independent assessment of CMM Level 3	Independent assessment of CMM Level 2	CMM Level 1 with record of repeated mission success	CMM Level 1 or equivalent	X2	
Degree of Innovation	Proven and accepted		Proven but new to the development organization		Cutting edge	X1	
Level of Integration	Simple - Stand alone				Extensive Integration Required	X2	
Requirement Maturity	Well defined objectives - No unknowns	Well defined objectives - Few unknowns		Preliminary objectives	Changing, ambiguous, or untestable objectives	X2	
Software Lines of Code***	Less than 50K		Over 500K		Over 1000K	X2	
Total							

Table 1 Likelihood of Failures Based on Software Environment

Finally, a weighting factor of 1 or 2 was identified for each factor. This information is shown in Table 1.

To apply the criteria, the project manager identifies the category of risk for each factor and multiples the appropriate value (1, 2, 4, 8, or 16) times the weighting factor of 1 or 2. The sum for all factors yields an initial numerical representation of the project software development risk. For example, a project might rate the following:

- Software team complexity – "up to 20 people at one location" = 4 * 2 = 8
- Contractor support – "with minor tasks" = 2 * 2 = 4
- Organizational complexity – "two locations but with same reporting chain" = 2 * 1 = 2
- Schedule pressure – "non-negotiable" = 16 * 2 = 32
- Process maturity – " CMM Level 1 but with a successful history" = 8 * 2 = 16
- Innovation – between proven but new and cutting edge = 8 * 1 = 8
- Integration – almost stand alone = 2 * 2 = 4
- Requirement maturity – "preliminary objectives" = 8 * 2 = 16
- Lines of code = ~ 300K = 2 * 2 = 4

$\therefore$ TOTAL = 8+4+2+32+16+8+4 +16+4= 94

This risk information must now be combined with the consequence evaluation.

Consequence Evaluation

In general, the consequences of a software failure can be derived from the purpose of the software: i.e., what does the software control; what do we depend on it to do. NASA has many types of software, including flight software which is launched and contains mission critical functional, ground system that sends commands, scientific software for the experiments, and just about all other types of software imaginable. Factors which can be used to categorize software based on its intended function as well as the level of effort expended to produce the software are defined as follows:

a. Potential for loss of life - Is the software the primary means of controlling or monitoring systems that have the potential to cause the death of an operator, crewmember, support personnel, or bystander? The presence of manual overrides and failsafe devices are not to be considered. This is considered a binary rating: responses must be either yes or no.
b. Potential for serious injury - Serious injury is defined as loss of digit, limb, or sight in one or both eyes, sudden loss of hearing, or exposure to substance or radiation that could result in long term illness. This rating is also binary. This rating considers only those cases where the software is the primary mechanism for controlling or monitoring the system. The presence of manual overrides and failsafe devices are not to be considered.
c. Potential for catastrophic mission failure - Can a problem in the software result in a catastrophic failure of the mission? This is a binary rating.
d. Potential for partial mission failure - Can a problem in the software result in a failure to meet some of the overall mission objectives? This is a binary rating.
e. Potential for loss of equipment - This is a measure of the cost (in dollars) of physical resources that are placed at risk due to a software failure. Potential collateral damage is to be included. This is exclusive of mission failure.

f. Potential for waste of software resource investment- -This is a measure or projection of the effort (in work-years, civil service, contractor, etc.) invested in the software. This shows the level of effort that could potentially be wasted if the software doesn't meet requirements.
g. Potential for adverse visibility - This is a measure of the potential for negative political and public image impacts stemming from a failure of the system as a result of software failure. The unit of measure is the geographical or political level at which the failure will be common knowledge—specifically: local (Center), Agency, national, international. The potential for adverse visibility is evaluated based on the history of similar efforts.
h. Potential effect on routine operations - This is a measure of the potential to interrupt business. There are two major components of this rating factor: scope and impact. Scope refers to who is affected. The choices are Center and Agency. The choices for impact are inconvenience and work stoppage.

Now the Potential for failure had to be quantified. Four ratings were chosen: Grave, Substantial, Marginal and Insignificant. Each of the factors above were quantified for each rating. If *any* of the conditions are met, the software is considered to reside in that category. The categories are defined as follows:

- *Grave*
  Potential for loss of life - Yes
  Potential for loss of equipment – Greater than $100,000,000
  Potential for waste of resource investment – Greater than 200 work-years on software
  Potential for adverse visibility - International
- *Substantial*
  Potential for serious injury – Yes
  Potential for catastrophic mission failure – Yes
  Potential for loss of equipment – Greater than $20,000,000
  Potential for waste of resource investment – Greater than 100 work-years on software
  Potential for adverse visibility - National
  Potential effect on routine operations – Agency work stoppage
- *Marginal*
  Potential for partial mission failure - Yes
  Potential for loss of equipment – Greater than $2,000,000
  Potential for waste of resource investment – Greater than 20 work-years on software
  Potential for adverse visibility - Agency
  Potential effect on routine operations – Center work stoppage or Agency inconvenience
- *Insignificant*
  Potential for loss of life - No
  Potential for serious injury – No
  Potential for catastrophic mission failure – No
  Potential for partial mission failure – No
  Potential for loss of equipment – Less than $2,000,000
  Potential for waste of resource investment – Less than 20 work-years on software
  Potential for adverse visibility – No more than local visibility
  Potential effect on routine operations – No more than a Center inconvenience

For example, the project described below would determine the consequence as demonstrated.

- Software controls life-support systems (potential loss of life)
- There is no potential for loss of equipment
- There is potential for adverse visibility at the Agency level
- There would be no effect on routine operations

∴ This software would rate the consequence of failure as "GRAVE" because it met one of the conditions – loss of life support. Software consequence is determined by meeting any of the conditions within that category.

Combining the software the probability of failure rating and the consequences of failure yields a risk assessment, which can be used to identify the need for IV&V Application of these criteria only determines that a project is a candidate for IV&V – not the level of IV&V nor the resources associated with the IV&V effort. These must be determined as a result of discussions between the project and the IV&V Facility.

Figure 1 shows a dark region of high risk where software consequences, likelihood of failure, or both are high. Projects having software that falls into this high-risk area shall undergo IV&V. The exception is those projects which have already done hardware/software integration. An IV&V would not be productive that late in the development cycle. The gray regions of intermediate risk. Projects having software that falls into these areas shall undergo an evaluation to determine if IV&V is warranted.

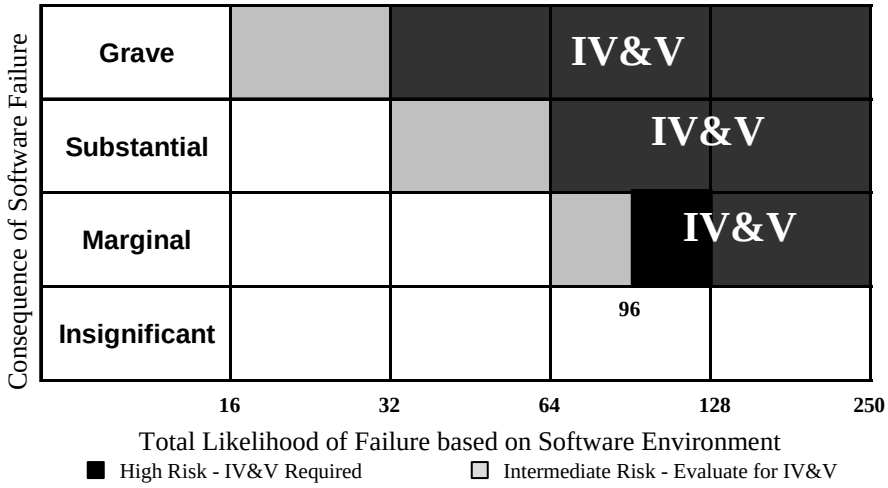


Figure 1: Probability & Consequence = Risk

Using the previous project example, a probability of 94 score means that they must perform IV&V if the consequence of failure is Substantial or Grave. Since the project's consequence was determined to be Grave, they must perform IV&V under the NASA IV&V Facility.

Step 3 - Project implementation

Prior to implementing IV&V, a company has to know about the software it develops. At NASA, software development is done at all 10 Centers spread from California, to Florida, Texas to Ohio, and Alabama to Maryland, and with Universities and industries. NASA projects are initially funded through multiple Enterprises at Headquarters. Each Enterprise was requested to compile a list of their projects. Since all projects have a software component, some very small and some mission critical, this list was used as a starting point for investigating the application of IV&V.

Approximately 100 projects were identified across all NASA Centers. Project managers were sent a request from NASA's Chief Engineer to apply the criteria to determine if their project was a candidate for IV&V. If they fit the category of potentially requiring IV&V, they were then ask to apply the criteria, estimating the probability of failure and the consequence. This information was sent to the NASA IV&V Facility where personnel evaluated the results and looked for inconsistencies in the data. For example, a project that indicated they were at a CMM Level 5, the contractor was identified, since very few companies are at Level 5. If a project marked their deadline was negotiable, the innovation was proven and accepted and the requirement maturity was well defined, with no unknowns, indicating a project doing nothing new and in no hurry to get there. This is not usually the case for NASA software. In many cases, the person completing the information did not understand a factor, such as what CMM is. In a few cases, the project chose the lowest possible risk to prevent the need for IV&V. (All data was scanned to identify those projects who checked the least risk category for each factor.)

Projects were given one week to complete the criteria information. Projects not completing the data were sent a second letter from NASA Headquarters with notification to appropriate personnel at the Center. Center Directors were eventually notified of the projects that did not respond (only two), and the data was promptly received. In total, data was received on approximately 100 projects. After discussions with projects to clarify the information and correct erroneous data, approximately seventy projects were identified as candidates for IV&V.

Applying IV&V to seventy projects immediately however, is an impossibility. The Facility currently does not have the resources to accommodate this many projects and this much work at this time. Using the data from the criteria (consequence and probability) and discussions with the project managers, the following guidelines to focus the IV&V efforts were implemented.

1. All project currently receiving IV&V will continue.
2. All projects classified as "GRAVE" should be addressed first for IV&V with the highest priority applied to those closest to operational date as a general rule, with some attention applied to why it is classified as "GRAVE".
3. All projects classified as "SUBSTANTIAL" and needing IV&V based on the risk probability value greater than 32 that are in the requirements or design phase.
4. All remaining projects classified as "SUBSTANTIAL" and needing IV&V based on the risk probability value greater than 32.

5. All remaining projects classified as “MARGINAL” and needing IV&V based on the risk probability value greater than 96, prioritized based upon how close to starting they are.

4 RESULTS

The application of IV&V on NASA projects has shown some very positive results and prevented costly errors. In one project, the IV&V activity identified design flaws in the Command and Control system which had it not been corrected, would have resulted in a catastrophic hazard. This critical piece of software sends commands to hardware elements, and if not working properly, could fail to send emergency response commands leading to the loss of attitude control, rapid depressurization, and other hazardous conditions. Using a code analysis, IV&V identified an error which would eliminate the vital command link between the ground control system and the satellite. A special software patch was generated for the on-orbit software to correct the problem. IV&V developed policy criteria used by one Program for non-flight software in integrated tests. This policy was key for insuring testing integrity while making it possible to keep tight development schedules.

IV&V activities have benefited many different domains of NASA’s software. In the Manned Space flight domain, over 4,000 problems were identified, 10 of the highest criticality, those that could result in loss of mission or loss of life. For Experimental Flight Vehicles, IV&V identified over 300 requirements and design problems. For Ground systems, over 250 legacy system requirements and mitigation problems were identified.

5 CONCLUSION

The results of NASA’s investigation have shown that in this environment, IV&V has been cost effective. Companies large and small, in today’s competitive world cannot afford software that is unreliable. To be effective however, IV&V must be applied effectively and the independence must not be lost. Project managers must understand the benefits above the cost, looking at the whole development, not just the current state.

NASA has recognized the value of IV&V and has taken steps to implement IV&V on all software projects where warranted. The decision to implement IV&V is no longer solely the decision of the project manager, but through an independent evaluation, the risk of the project is evaluated, and the need for IV&V is determined. Although the policy and criteria in this paper were written for NASA projects, they are applicable with minor modification to any software development.

ACKNOWLEDGEMENTS

Much of the work presented in this paper was a combined effort of NASA IV&V Facility personnel (Judy Bruner, John Hinkle, and Ken McGill), Goddard Space Flight Center personnel (Charles Vanek, John Dalton, Linda Rosenberg), and the members of the Software Working Group under Pat Schuller, Langley Research Center.

REFERENCES

[1] IEEE 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology , Institute of Electrical and Electronics Engineers, Inc.

[2] *NASA IV&V Business Plan*, Office of System Safety and Mission Assurance, NASA Goddard Space Flight Center, MD , June 2000, http:// ivvplan.gsfc.nasa.gov.

[3] *NASA Policy Guideline (NPG) 7120.5 "NASA Program and Project Management Processes and Requirements Highlight Code"*, Office of Chief Engineer, NASA Headquarters, Washington, DC.

QW2001 Keynote 5P2

Dr. Dalibor Vrsalovic
(Intel Corporation)

Issues in Design and Validation of Modern eBusiness Systems

Presentation Abstract

Silicon economics helps in reducing cost of building modern eBusiness systems. However, year after year, the ongoing operations and software maintenance cost is killing many of the Internet based business models. This is due to two major facts: high complexity of new eBusiness systems, and low level of design, verification and management tools involved. In other words there is too little automation, and too much manual labor involved.

This talk will discuss modern trends in Internet systems architecture. It shows how higher level system abstractions could save both time and cost during the design, building, validation and operations of eBusiness systems. Author will also discuss a concrete example of a high level system model and its benefits.

About the Author

Dalibor F. Vrsalovic is a Vice President and Chief Technology Officer of Intel's New Business Group. In this position, Dr. Vrsalovic has responsibility for the long-range development of the architecture for Internet services and systems and for working with internal and external technology vendors to ensure a continuing stream of innovation for Intel's Internet platforms and services.

Prior to joining Intel, Dr. Vrsalovic was Vice President of Internet Technology at AT&T, where he and his team developed Internet service platforms and additional technologies and components supporting Internet telephony and messaging. Prior to this assignment, Dr. Vrsalovic was the Advanced Technology Vice President, AT&T Labs. Before joining AT&T, he was Chief Scientist at Sun Microsystems/ SunSoft, where he managed the Advanced Technology Group. He was also the Vice President of Engineering at Ready Systems, where he directed worldwide R&D, product engineering, quality assurance, customer support, and manufacturing.

Dr. Vrsalovic was a member of the faculty at Carnegie-Mellon University and the University of Zagreb, has consulted to various governments and companies worldwide, and helped design several formative data networks. He is a member of various boards, including the Purdue University Advisory Board, Carnegie-Mellon University EDRC, Tripwire Security Inc. Board, and the Board of Manufacturing and Engineering Design of the National Academy of Engineering.

Dr. Vrsalovic holds a B.S. in Electrical Engineering, an M.S. in Computer Engineering, and a Ph.D. in Computer Sciences, all from the University of Zagreb. During his studies, Dr. Vrsalovic was also an accomplished athlete as a member of the national water polo team and the European Water Polo Cup winning club.

The Experts!

Cem Kaner

Bill Howden

Kris Mohan

Elisabeth Hendrickson

Linda Rosenberg

Nick Borelli

QW2001 Panel 8P

Thursday, 31 May 2001
5:00 - 6:30

Mr. Nick Borelli
(Microsoft Corporation)

Stump the Quality Experts If You Can! QW2001 Advisory Board Members Will Answer All Questions!

Post Your Questions LIVE on the Web!
(Live question collection and voting provided by Microsoft Corporation)

How The *Ask The Quality Experts* Panel Works

This special QW2001 panel session works **interactively** with you to get your key questions answered! If you have a burning question about any aspect of Software or Internet Quality, click on the Ask The Quality Experts! page.

You'll see the current set of questions posed to the Panel Of Experts, rank ordered based on the number of votes each question has received.

Are The Questions Moderated?

Yes, the questions posted are moderated. From time to time the Ask The Quality

Experts! Panel Moderator, Nick Borelli, will review the current set of questions. He'll remove off-topic questions, consolidate obvious duplicates, and make other necessary corrections. If there are high-scoring questions that seem to be outside the range of the experts currently on the panel we will **add more experts to the panel**.

About The Panelists

The panelists are chosen from among the Quality Week Advisory Board. You can see who the panelists are on the current Ask The Quality Experts! page.

How Often Can I Vote?

You can vote as often as you like, but, **please** we ask that you only vote for your favorite question(s). P.S. QW2001 will have Web workstations available where you can vote on-site.

How Will I Get Answers to My Question?

During this special session the Advisory Board Experts will answer the top ranking questions -- using the data from the web as of Noon on Thursday 31 May 2001. QW2001 will have Web workstations available where you can vote on-site before then.

Brief summaries of the answers will be posted on the Web shortly after the conference is over.

About the Moderator

Nick Borelli is currently a Group Test Manager at Microsoft Corporation and is responsible for the World-Wide releases of the award-winning application, Microsoft Word.

Nick has over 15 years experience in both Software Testing and Development and has worked in both small start-ups such as Pensoft, Go and EO, as well as working at Triad Systems, Apple and Software Publishing Corporation.

QW2001 Keynote 10P1

Ms. Lisa Crispin (Tensegrent)

The Need For Speed: Acceptance Test Automation In an Extreme Programming Environment (QW2000 Best Presentation)

Key Points

- Why Testing for XP is Different, challenging conventional wisdom
- How to educate yourself in the eXtreme Programming (XP) methodology
- How to automate functional tests quickly and leverage them to save time

Presentation Abstract

In my two and a half years working in a web environment, where quality and time to market are both essential to success, I've been frustrated by the difficulty in combining these traits within traditional software process. After reading Kent Beck's book, eXtreme Programming Explained, I couldn't wait to try this methodology to enable small teams to deal with short timeframes and changing requirements while still producing high quality software.

Testing in a Web environment can feel like leaping out of a plane. Testing in an XP environment feels like competing in a sky-surfing competition. You have to be better than everyone else, but you don't have much time. You can only hope for a soft landing. While the eXtreme Programming literature (including Ron Jeffries' book, eXtreme Programming Installed), centers around unit and integration testing as part of the XP core process, I felt that functional/acceptance testing from the customer perspective was incompletely defined. The role of the tester in XP is clearly defined - to help the customer choose and write functional tests and to make sure those tests run successfully. The question is, how to do this when the ratio of developers to testers is quite high (8 - 1 is recommended, and we are in a more extreme ratio than that) and the development iterations are so short.

Like an extreme-sports competitor, the XP tester needs courage, speed, stamina and creativity. Working with the developers and with input from an automated test tool vendor, I have developed an approach to designing modularized, self-verifying tests that can be quickly developed and easily maintained. I'll present my basic design and give some examples. I used the test tool WebART, but this methodology should be applicable to any au tomated tool that includes a scripting language.

About the Speaker

Lisa Crispin is a Senior Quality Engineer at Tensegrent (http://www.tensegrent.com/), a very different type of software development company built around the streamlined eXtreme Programming (XP) methodology. Consistent with the values of XP, Tensegrent focuses on delivering high quality software that provides immediate business value, while remaining responsive to changing requirements.

Lisa has managed to enjoy her work during almost 20 years of non-XP software development by demonstrating the flexibility needed to dodge the boring projects and grab the cool ones. Her most recent fun job before joining Tensegrent was as Quality Boss of TRIP.com, where she embraced the challenge of bringing QA to a chaotic web startup. While TRIP.com basked in much success - 4.5 stars from BizRate, in the top 5 of the Keynote Top 40 for performance, sale to Galileo International for $326M - Lisa was continually frustrated by the difficulty of finding a process to deliver high quality software quickly in a dynamic and competitive industry.

Since leaping into the unknown world of XP last June, Lisa has figured out the role of the XP tester and, with lots of help, developed a test automation methodology which can keep up with the pace of XP. She's happy to report that XP practices, well understood and diligently followed, really work! However, she is still trying to overcome the inherent disadvantages of not being good at Foosball.

Lisa also enjoys wine, horses and testing web applications which contain no Javascript.

The Need for Speed: Acceptance Test Automation in an Extreme Programming Environment

Lisa Crispin, Senior Quality Engineer,
Tensegrent

TENSEGRENT

What is XP?

XP Values:

- Simplicity
- Communication
- Feedback
- Courage

TENSEGRENT

What Makes it Extreme?

Commonsense Practices to Extreme Levels

- If testing is good, everybody tests all the time
- If code reviews are good, pair program
- If design is good, refactor every day
- Extremely Practical
- Extremely Productive!

TENSEGRENT

3

XP Overview

Commonsense Practices to Extreme Levels

- 4 variables: cost, quality, time and scope
- Planning Game
  - Developers estimate stories and velocity
  - Customer chooses stories
- Iteration Planning: Write, estimate and accept tasks
- One- to three-week iterations

TENSEGRENT

4

XP Overview

Commonsense Practices to Extreme Levels

- Acceptance tests document customer needs
- Testing concurrent with development
- Test result reports help steer
- Defects may become stories for future iterations

TENSEGRENT

5

Automated Testing

This Presentation Will Give Tips On:

- How Testing in XP is Different
- Tools for Writing Acceptance Tests
- Tools for Automating Acceptance Tests
- Tools for Reporting Results
- Lightweight Automated Test Design

TENSEGRENT

6

XP Practices

A few XP practices used at Tensegrent:

- Pair Programming
- Test First, Then Code
- Do the Simplest Thing that Works
- 40-Hour Week
- Refactoring
- Coding Standards
- Small Releases
- Incremental planning

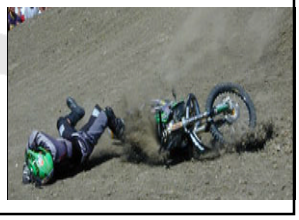
TENSEGRENT

7

XP Differences

How is Testing Different with XP?

- Customer can change mind anytime
- Lack of written documents UP FRONT
- Short cycles: 1 - 3 weeks
- High developer/tester ratio



TENSEGRENT

XP Differences

How is This OK?

- Unit and integration tests must pass 100%
- Continual customer involvement
- Development team assists with test automation
- Tester is part of development team
- Tester advocates customer viewpoint

TENSEGRENT

9

XP Test Automation

So, how do we get there?

TENSEGRENT

10

XP Test Automation

XP Test Practices:

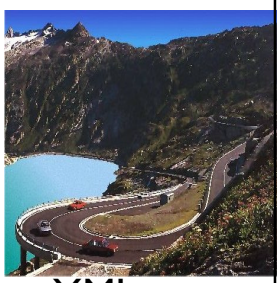
- Pair test
- Continually refactor
- Verify critical functionality first
- Add automated tests as budget permits
- Use / develop lightweight test tools

TENSEGRENT

11

XP Test Automation

Effective XP Acceptance Tests



- Test cases and data in spreadsheet or XML format
- Data and actions separated
- Granular enough to show progress
- Include "nasty path" or "Soap Opera" tests
- Include load, performance criteria

TENSEGRENT

12

Acceptance Test Automation

Basic User Scenario

Action	*Minimum Passing Criteria*
Go to Login Page	Challenges for Authentication
Login	Verifies valid userid and password; resulting page contains correct forms and links (list)
Search for Valid Category for Denver area	Table of appropriate search results
Logout	Login Page

TENSEGRENT

13

Acceptance Test Creation

Spreadsheet Template for Customer Tests

Actions:

1. Enter login name and password
2. Click Submit

Data:

Login ID	**Password**	**Expected Result**
Testy	tester	Login successful
jim-bob	11111	Login successful
empty	(spaces)	Invalid login and/or password

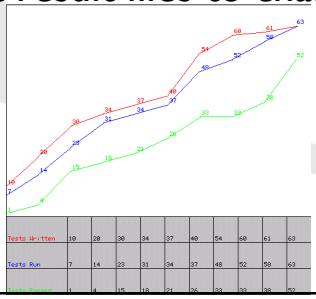
TENSEGRENT

14

Tensegrent Test Tools

Automated Testing for Web Applications

- In-house tool to convert test case data for input to automated tests
- HTTP-based tool - WebART
- Tool inputs test cases to WebART
- In-house tool to convert result files to charts and graphs



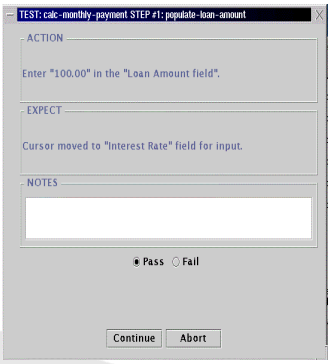
TENSEGRENT

15

Tensegrent Test Tools

Automated Testing for GUI Applications

- In-house tool "TestFactor-e"
- Prompts user for repeatable test
- Records pass or fail, comments
- Detail and summary reports
- Extending to input automatically



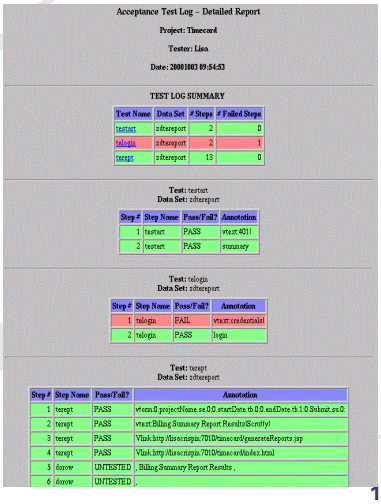
TENSEGRENT

16

Tensegrent Test Tools

Acceptance Test Reports

- Visual feedback
- Drill down for detail
- Post as "mile marker"
- Helps team steer
- Promotes change



Acceptance Test Log – Detailed Report

Project: Timecard

Tester: Lisa

Date: 20001003 09:54:53

TEST LOG SUMMARY

Test Name	Data Set	# Steps	# Failed Steps
testart	zdtexport	2	0
telogin	zdtexport	2	1
texrpt	zdtexport	13	0

Test: testart
Data Set: zdtexport

Step #	Step Name	Pass/Fail?	Annotation
1	testart	PASS	vtest:4011
2	testart	PASS	summary

Test: telogin
Data Set: zdtexport

Step #	Step Name	Pass/Fail?	Annotation
1	telogin	FAIL	vtest:credentials!
2	telogin	PASS	login

Test: texrpt
Data Set: zdtexport

Step #	Step Name	Pass/Fail?	Annotation
1	texrpt	PASS	vtest.0.projectName.se.0.0.startDate.th.0.0.endDate.th.1.0.Submit.se.0
2	texrpt	PASS	vtest:Billing Summary Report Results/Sconfty!
3	texrpt	PASS	Vlink:http://lisacrispin:7010/timecard/generateReports.jsp
4	texrpt	PASS	Vlink:http://lisacrispin:7010/timecard/index.html
5	dorow	UNTESTED	, Billing Summary Report Results ,
6	dorow	UNTESTED	,

TENSEGRENT

17

Tensegrent Test Design

Tests must:

- Be modular and self-verifying
- Verify the minimum success criteria
- Contain no duplicate code
- Do the simplest thing that works
- Feature reusable modules

TENSEGRENT

18

Tensegrent Test Design

Creating Test Scripts

- Create templates for all modules
- Record scenario with capture tool
- Cut and paste into template code
- Use utility/validation modules
- One function per module

TENSEGRENT

19

Tensegrent Test Design

Modules

- Main Script - calls supporting modules
- Interface Modules - one function per module, validates system responses
- Validation Modules - check for specific conditions, return pass/fail
- Utility Modules - track execution, log results in XML format for reporting tools
- http://www.tensegrent.com for sample scripts

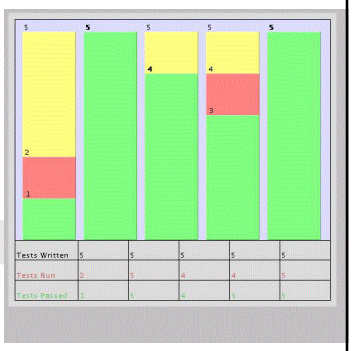
TENSEGRENT

20

Automated Testing

Rewards

- Keep team at maximum safe speed
- Navigate the curves
- Help make needed corrections
- Watch for landmarks
- Keep team informed of progress



TENSEGRENT

21

The Need for Speed:
Acceptance Test Automation in an Extreme Programming Environment

Lisa Crispin, Senior Test Engineer, Tensegrent
Contributors: Carol Wade, TRIP.com; Tip House, OCLC.org

"Extreme Programming, or XP, is a lightweight discipline of software development based on principles of simplicity, communication, feedback, and courage. XP is designed for use with small teams who need to develop software quickly in an environment of rapidly changing requirements." Ron Jeffries, http://www.xprogramming.com.

What makes XP Extreme?

As Kent Beck says in Extreme Programming Explained, XP takes commonsense principles and practices to extreme levels. For example: if testing is good, everybody will test all the time (unit testing), even the customers (acceptance testing). Taking anything to extremes can feel scary. While you or I might happily go skiing, we're not likely to ski off the side of a cliff in the manner of Warren Miller. Extreme Programming isn't about taking risks - it's about reducing risks and having fun. It takes courage, but the rewards are immediate.

The XP practices we follow at Tensegrent include:

- pair programming
- test first, then code
- do the simplest thing that works (NOT the *coolest* thing that works!)
- 40-hour week
- refactoring
- coding standards
- small releases
- play the planning game

How is Testing in XP Different?

How does acceptance testing in an XP environment deviate from traditional software testing? First of all, let's look at acceptance testing. Acceptance tests may include load and performance tests as well as functional and system tests. Acceptance tests prove that the application works as the customer wishes. Acceptance tests give customers, managers and developers confidence that the whole product is progressing in the right direction. Acceptance tests check each increment in the XP cycle to verify that business value is present. Acceptance tests, the responsibility of the tester and the customer, are end-to-end tests from the *customer* perspective, not trying to test every possible path through the code (the unit tests take care of that), but demonstrating the business value of the application.

Should I strap on a helmet and elbow pads?

Testing in an XP environment feels like a run through a half-pipe when you first try it, turning the software development model on its head. The customer is allowed to change her mind anytime. The XP techniques make sure the cost of making changes remain constant throughout the life of project.

Testers may be dismayed at first by the lack of formal written requirements and specifications. To produce small releases very quickly, XP minimizes written documentation. The system is documented through the unit tests, acceptance tests and the code itself. Customers may create mockups of screens and sample reports, but no traditional specifications are written. Design is done primarily with a whiteboard. Collective ownership, promoted by pair programming, reduces the need for written documentation (Which usually is immediately out of date anyway!)

Question: How do you write acceptance test cases without documents?
Answer: According to the XP books, the customer writes the acceptance tests, assisted by the tester. In my own experience, I've found that for various reasons, customers are not about to sit down and write out their acceptance tests. However, they will tell me what acceptance tests they want. I write them down and go over them with the customer, changing and modifying them until she is satisfied that they will show whether the software meets her requirements.

Other differences between traditional and XP development are more subtle. It's really a matter of degree. XP projects move fast even when compared with the pace at the Web startup where I used to work. It's like running a motocross race when you're accustomed to a street bike. A new iteration of the software, implementing new customer "stories", is released every one to three weeks. The tester and customer must start writing acceptance tests at the beginning of each iteration, as these are the only written "specifications" available. Acceptance tests should run along with unit tests after each integration - which could be several times a day.

From a tester's point of view, the developer to tester ratio in XP looks about as comfortable as street luge. According to Kent Beck, there should be one tester for each eight-developer team. At Tensegrent, the ratio is even higher.

Eeek! Are you SURE protective armor is not required?

Fear not! XP builds in checks and balances that enable a small percentage of test specialists to do an adequate job of controlling quality.

- Becuse the developers write so many unit tests , which they must write before they begin coding - the tester doesn't need to verify every possible path through the code.
- The developers are responsible for integration testing and must run every unit test each time they check in code. Integration problems are manifested before acceptance tests are run.
- The customer provides assistance with writing the acceptance tests and is responsible for deciding when they are complete. The customer usually also helps to execute the acceptance tests.

- The entire development team, not just the tester, is responsible for automating acceptance tests. Developers also may help the tester produce reports of test results so that everyone feels confident about the way the project is progressing.

The roles of the players on an XP team are quite blurred compared with those in a traditional software development process. Thus our Tensegrent XP ("TXP") philosophy is "*specialization is for insects"*. Here are some of the tasks I perform as a tester:

- Help the customer write stories
- Help break stories into tasks and estimate time needed to complete them
- Help clarify issues for design
- Team with the customer to write acceptance tests
- Pair with the developers to code the application and the test tools
- Pair with the developers to code automated test scripts

Question: Wait a minute. The whole concept of pair programming sounds weird enough. How can a tester pair with a programmer?
Answer: I'm not a Java programmer and our developers don't know the WebART scripting language, but we can still pair program. The partner who is not doing the actual typing contributes by thinking strategically, spotting typos and even serving as a sounding board for the coder. This is a fabulous way for developers and testers to understand and work together better. It also gives the tester *much* more insight into the system being coded. Pairing may not be an everyday occurrence for the tester as it is for the developers, but it happens whenever it's needed.

Once you've mustered the courage to jump in to XP, the water's great.

How do I Educate Myself About XP?

Just as you wouldn't attempt to climb Mount Everest without preparing yourself with months of intense training. the XP team needs good training to start off on the right path and stay on it.

Start by reading the XP books. The first book on to be written on XP is Extreme Programming Explained, by Kent Beck. It's a fascinating and quick read. Two new books will be published in the fall of 2000, Extreme Programming Installed, by Ron Jeffries, Ann Anderson, and Chet Hendrickson; and Planning Extreme Programming, by Kent Beck and Martin Fowler.

You can get an overview and extra insight into XP and similar lightweight disciplines from the many XP-related websites, including:
http://www.xprogramming.com
http://www.extremeprogramming.org
http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap
http://www.martinfowler.com

When we at Tensegrent had assembled our first team of eight developers and a tester, we

got together and went through Extreme Programming Explained and Extreme Programming Installed as a group, discussing each XP principle, recording our questions (many of them on testing) and deciding how we thought we would implement each principle. This took several hours but put us all on common ground and made us feel more secure in our understanding of the concepts.

Once your team has read and discussed the XP literature, it's time to get professional training. We hired Bob Martin of ObjectMentor, a consulting company with much XP expertise, for two days of intense training (see www.objectmentor.com for more information). After Bob answered all our questions, we felt much more confident about areas that had previously been difficult for us to understand, such as the planning game, automated unit testing and acceptance testing.

Don't stop there. Talk to XP experts. Look at the Wiki pages and sign up for the egroups. If no XP user group has been formed in your city, start one.

Automating Acceptance Tests

What can you automate?

According to Ron Jeffries, author of XP Installed, successful acceptance tests are customer-owned, comprehensive, repeatable, automatic, timely and produce results that are known to everyone. The "automatic" criterion has given us trouble in some cases, although our goal is to automate whenever it makes sense. Sometimes a mountain bike is the best way up the hill; other times it's easier to get off and walk your bike. For example, we haven't found a cost-effective way to automate Javascript testing. Also we're struggling with how to automate non-Web GUI testing in an acceptable timeframe. If we're just doing a 6 week project for a customer, the customer may not wish to pay for automation. Even if we can't automate a test right away, we *can* make it comprehensive, repeatable and timely, and we can publish our results.

Principles of TXP Test Automation

For ease of development and maintenance, automated test scripts should meet the following criteria:

- **Modular and self-verifying** to keep up with the pace of development.
- **Verify the minimum criteria for success.** Because the unit tests are comprehensive, we don't need to duplicate them or try to hit every path through the code with our acceptance tests. Demonstrate the business value, but don't do more than you need.
- **Perform each function in one and only one place** to minimize maintenance time.
- **Contain modules that can be reused,** even for unrelated projects
- **Do the simplest thing that works.** This XP value applies as much to testing as to coding.

In addition, the developers try to design the software with testability in mind. This might mean building hooks into the application to help automate acceptance tests. Push as

much functionality as possible to the backend, because it is much easier to automate tests against a backend than through a user interface.

Follow these **XP Test Practices** to help keep up with the pace of XP iterations:

- **Continually refactor**
- **Verify most critical functionality** first with a "smoke test"
- **Add to automated tests** as long as maintenance resources not exceeded
- **Pair test,** especially for defining test cases and automated test scripts.

TXP Automated Test Design

Appendix A describes a lightweight automated test design that works with XP projects. We use WebART (see the Tools section below) to create and run the scripts. However, this design should work with any method of automation that permits modularization of scripts. Please see Appendix A for the details of this test design.

Who automates the acceptance tests?

Some sports appear to be individual, when in actuality, they involve a team. Winners of the Tour de France get all the glory, but their victory represents a team effort. Similarly, the XP team may have only one tester, but the entire team contributes to automating acceptance tests. If tools are needed to help with acceptance testing in an XP project, write stories for those tools and include them in the planning game with all the other stories. You'll probably need to budget at least a couple of weeks for creating test tools for a moderately size project.

In the early days of Tensegrent, we initiated a project for the specific purpose of developing automated test tools. This had several advantages, in addition actually producing the tools:

- **Practice with XP** writing stories, playing the planning game, estimating. This gave us confidence in our XP skills that served us future projects.
- **Practice with development technologies.** Developers could experiment with different approaches and get experience with new tools. For example, the developers investigated in advance the advantages of using a dom versus a sax parser on the XML files containing customer test data. Doing this in advance gave us more time to experiment and research technologies than we might have had later with a client project.
- **Mutual understanding.** The team tasked with producing an acceptance test driver consisted of only four members and me, so I was called on to pair program. This exercise gave me insight into how tough it is to write unit tests, write code and refactor the code. The developers gave a lot of thought to acceptance testing and we had long discussions about what the best practices would be. This is a great foundation for any XP team.

Tools

Sky surfers don't leap out of the plane wearing any old parachutes purchased from a discount store. They look for state-of-the-art harness and container systems, main and reserve canopies, helmets and goggles, even altimeters, all designed with their particular needs in mind. XP testers need a good toolbox too, one containing tools designed specifically for speed, flexibility and low overhead.

I've asked several XP gurus, including Kent Beck, Ward Cunningham and Bob Martin, the following question: "What commercial tools do you use to automate acceptance testing?" Their answers were uniform: "Grow your own". Our team extensively researched this area. Our experience has been that we are able to use a third-party tool for Web application test automation, but we need homegrown tools for other purposes.

For **unit testing**, we use a framework called jUnit, which is available free from http://www.junit.org. (There are flavors available for other languages besides Java.) It does an outstanding job with unit tests. Even though I am not a Java programmer, I can run the tests with jUnit's TestRunner and can even understand the test code well enough to add tests of my own. It's possible to do some functional tests with jUnit. Some XP teams use this tool for automating acceptance tests, but it cannot test the user interface. We didn't find it to be a good choice for end-to-end acceptance testing.

Tools for Creating Acceptance Tests

Some XP pros such as Ward Cunningham advocate the use of spreadsheets for driving acceptance tests. This isn't a new idea. We want to make it easy for the customer to write the tests, and most are comfortable with entering data in a spreadsheet. Spreadsheets can be exported to text format, so that you and/or your development team can write scripts or programs to read the spreadsheet data and feed it into the objects in the application. In the case of financial applications, the calculations and formulas your customer puts into the spreadsheet communicate to the developers how the code they produce should work.

At Tensegrent, we generally use a spreadsheet format for both acceptance tests – descriptions of commands, actions, input data and expected results – and for the test case data which will drive the tests. Most people are familiar with spreadsheets and comfortable working in that format. If we're using a test tool that requires another format such as XML, we do it that way, or convert the spreadsheet data to an XML format. See Appendix B for a sample acceptance test spreadsheet template.

Appendix C shows a ***partial*** excerpt of a sample XML file used for acceptance test cases that will be run with our homegrown test tool. We enter a description of the test, data and expected output, steps with actions to be performed and expected results.

Automated Testing for Web Applications

Test automation is relatively straightforward for Web applications. The challenge is creating the automated scripts quickly enough to keep pace with the rapid iterations in an XP project. Like a motocross racer, I'm zipping down hills and slogging through mud, trying to keep up with the pack of developers. For that extra burst of speed, I use WebART (http://www.oclc.org/webart), an inexpensive HTTP-based tool with a powerful scripting language. WebART enables me to create modularized test scripts, creating many reusable parts in a short enough timeframe to keep up with the pace of development. Javascript testing presents a bigger obstacle. We test it manually and carefully control our Javascript libraries to minimize changes and thus the required retesting. Meanwhile, we continue to research ways of automating Javascript testing.

Our developers wrote a tool to convert test data provided in spreadsheet or XML format into a format that can be read by WebART test scripts so that we can automate Web application testing. Even small efforts like this can help you gain that competitive edge in the speedy XP environment.

Automated Testing for GUI Applications

Test automation for non-HTTP GUI applications has been more of an uphill climb. You can travel faster in a helicopter than a mountain bike, but it takes a long time to learn to fly a helicopter; they cost a lot more than a bicycle and ou may not find a place to land. Similarly, the commercial GUI automated test tools we've seen require a lot of resources to learn and implement. They're budget breakers for a small shop such as ours. We searched far and wide but could not come up with a WebART equivalent in the GUI test world. JDK 1.3 comes with a robot that lets you automate testing of GUI events with Java, but it's based on the actual position of components on the screen. Scripts based on screen content and location are inflexible and expensive to maintain. We need tests that give the developers confidence to change the application, knowing that the tests will find any problems they introduce. Tests that need updating after each application change could cause us to lose the race.

We felt that the most important criteria for acceptance tests is that they be repeatable, because they have to be run for each integration. We decided to start by developing our own tool, "TestFactor-e", that will help customers and testers run manual tests consistently. It will also record the results. We're now enhancing this tool to feed the test data and actions directly into application backends in order to automate the tests.

Reports

Getting feedback is one of the four XP values. Beck says that concrete feedback about the current state of the system is priceless. An extreme skier constantly monitors snow conditions, the course, his speed, the state of his equipment, all while keeping an ear out for the avalanche that may be coming along behind him. He accommodates these factors with changes in speed, trajectory and position. The XP team needs a constant flow of

information to steer the project, making corrections in mid-course just as the skier would. The team's continual small adjustments keep the project on course, on time and on budget. Unit tests give programmers minute-by-minute feedback. Acceptance test results provide feedback about the "Big Picture" for the customer and the development team.

Reports don't need to be fancy, just easy to read at a glance. A graph showing the number of acceptance tests written, the number currently running and the number currently succeeding should be prominently posted on the wall. You can find examples of these in the XP books. Our development team wrote tools to read result logs from both automated tests and manual tests run with "TestFactor-e". These tools produce easy-to-read detail and summary reports in HTML and chart format.

With all this feedback, you'll confidently deliver high-quality software in time to beat your competition. You'll meet the challenges of 21st century software development!

References

- Beck, Kent, Extreme Programming Installed., Addison-Wesley, 2000
- Jeffries, Ron et al, Extreme Programming Installed, Addison-Wesley, 2001

Appendix A: Test Design Description

XP Automated Test Design

The sample scripts used to illustrate the test design are written with a test tool called WebART (http://www.oclc.org/webart). Any test tool which permits modularization and paramterization of the scripts should support this design. To download a soft copy of the sample scripts, go to http://www.tensegrent.com and click on the "Sample WebART Scripts" link.

Appendix 1 contains a diagram with an overview of the test design.

The Sample Application

Our sample application is a telephone directory lookup website, http://www.qwestdex.com. This is certainly not intended as an endorsement of Qwest and we have no connection with them, it was just a handy public application with characteristics that allow us to illustrate the tests.

End User Scenario

Our test will emulate a basic end user path through the code which, in our view, tests the most critical functionality of the site. Here's the basic scenario we want to test:

Action	***Minimum Passing Criteria***
Go to login page	Application challenges for authentication
Login	Valid login name and password brings up profile page
Search for valid	Table of results

Action	*Minimum Passing Criteria*
category in specified city	
Logout	Login page

The Test Components

Main Script

The **main module** calls the supporting module to perform a typical user scenario and to validate the system responses at strategic points along the way. A basic user scenario is created for each customer story. The **supporting modules** are divided into several groups based on their function. These modules are described below.
The main module passes to the supporting modules test cases from tables of test data along with other parameters.

Interface Modules

Called by the main script module (and possibly other interface modules) to perform user functions and to validate the correct system response. Some of these modules such as start, login and exit can be used by multiple tests for the same application. The main module passes **parameters** to the interface modules as follows:

page loaded - An *output* parameter; it receives the value of the page loaded by the module. For example, the login module loads the “pQwestDex” page.

page used - An *input* parameter; it is the handle of the page used by the interface module to know what page to load. For example, the ???? page is passed to the login module.

test case data - An *input* parameter; it is data to be parsed by the interface module and used for input or validation. For example, for a login module, the test case consists of an *userid* and *password* .

outcome - An *output* parameter that receives a value of **PASS** or **FAIL** indicating the overall outcome of the call.

Additional parameters may be used if needed (e.g. more than one page needs to be loaded.)

The script "qwmain" in the sample scripts is an example of a main script. Here's a snippet of how it calls a supporting module:

```
zzLogin = zzLoginTable[$user];
zzPassword = zzPasswordTable[$user];
qwlogin(pQwestDex,http://${zzh_host}/cgi/tools/fcg?form==login&from==, zzLogin, zzPassword,
outcome);
```

The main module passes to the supporting modules test cases from tables of test data along with other parameters. In this example, the script obtains a login and password from the test case data tables and passes them along to the supporting login module along with the page loaded, the page used (in this case supplied as a URL), and a field to return the result.

If the login is successful, the qwmain script then proceeds to call another supporting module to perform a search, passing the page handle in pQwestDex, and receiving the handle for the resulting page in pResults:

qwsrch (pResults, pQwestDex, zzSearchCat, STD, outcome);

Validation Modules

Called by interface modules to check for specific conditions in a system response and return a pass or fail condition. The validation modules in turn call **utility** modules to record the results. Parameters are:
results - An *output* parameter which returns **PASS** or **FAIL** to the calling module

controls - An *input* containing values that control how the validation is done, specific to each validation module.

response or handle - An *input* parameter containing either the system response to be validated or the handle of the page into which the response was loaded.

Currently, there are three validation modules that can be used by any test:
vtext validates that a response contains specified text. Parameters are:

result - PASS or FAIL

text - the text that must be present for the validation to pass

response - the system response to check for the text string.

vlink validates that a page contains a specific link. Parameters are:

result , *urlMatch* - the value which must exist in a link in the page whose handle is in pPage

pPage - the **page handle** of the page being validated.

vform validates that a page contains a specified HTML form. Parameters are:

result , *formvars* - one or more required variables for the form - if any are missing, the validation fails.

pPage - the **page handle** of the page being validated.

Here's an example from the login supporting module, "zsqwlogin" in the sample scripts. It calls vtext to determine if two text strings (delimited by vertical bars) appear on the page. If one or both is not present, vtext will return 'FAIL' in the zzResult variable. The zzrespzz variable contains the system response that needs to be checked.

vtext (zzResult, Welcome|Edit My Profile|, zzrespzz, qwlogin);

Utility Modules

Currently there are two utility modules which can be used by any test:
trace - Displays execution tracing information in the WebART execution window. Called by interface and log modules. Without going into detail of the many parameters, they reveal where it was called from/ and what happened.
log - Records validation outcomes in a log file. Parameters are:

type - detail or summary, *outcome* - PASS or FAIL

validation - describes the validation performed.

The "zslog" module in the sample scripts writes test results out in XML format. An in-house tool from Tensegrent called TestFactor-e builds an HTML page from this log file showing the results with color-coding for pass, not run and fail. See Appendix B for an example.

Providing Test Case Data

Test case data can be input to the scripts two ways: 1) at compile time, through the data tables (eg, zdqwmain); 2) at runtime, through the use of parameters. For example, if you want to be able to run your script against a test machine, then against a production machine, and compare the results, you can make the host name a parameter and change it at execution time. Here's how it's done in the sample WebART script "qwmain":
!param zzh_host(enter host to run against:) = "qwestdex.com"
When you execute this script, you are prompted to enter a host name, with qwestdex.com as the default.

Creating the Scripts

Creating the first set of scripts is the hard work. Once you have a working set of modules, you can reuse entire modules in some cases or turn them into templates in other cases. Here are the steps I use (preferably as part of a pair) to create test scripts:

1. Capture a session for the scenario I want to test. See "capqwest" in the sample scripts as an example.

2. Copy "qwmain", "zsqwlogin" and the other supporting modules that I already have to new names. Strip out the code that was specific to that application.

3. Paste in the code specific to the scenario I want to test, copying from the captured script into the newly created "templates". Use XP principles here: work in small increments, make sure your scripts work before you go on. For example, first see if you can get the login to work. Then add the search. Then add the logic for switching depending on the pass/fail outcome. Remember to do the simplest thing that works and add complexity only as you need it.

Appendix B: Sample Acceptance Test Spreadsheet Template

	A	B	C	D	E
1		**Ref #:**	Template for acceptance test: Timecard/QA/AcceptanceTest.sdc		
2		**Iteration:**			
3		**Functionality proven by this test case * is / is not * critical**	**What does this test?** *Example: Tests to make sure that when a time entry record is input, it is saved to the database and the report generated correctly.*		
4		**What to do:**	*Examples given in italics*		
5	**Step**	**Command/URL**	**Action**	**Input Data**	**Expected Output**
6	1	*Access wwwltimecardladdEntry in browser*	*Select a project, release, iteration, task, and enter duration, date and comments*	*Row 1, columns Project, Release, Iteration, Task, Duration, Date, Comments*	*Selections displaying on screen*
7	2	*Still on wwwltimecardladdEntry*	*Click the save button*		*Message that data was saved to database*
8	3	*Access wwwltimecardlgenerateReports in browser*	*Select a project, start date and end date*	*Row 1, columns Project, Start Date, End Date*	*Report generated – data matches row 1 columns Project, Release, Iteration, Duration, Cost and Total Cost*
9	4	*Repeat steps 1 – 3 with each row in the test case spreadsheet*			

Appendix C: Partial Excerpt of XML Template for Acceptance Test Cases

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE at-test SYSTEM "at-test.dtd" [
  <!ELEMENT input ANY >
  <!ELEMENT loan-amount ANY >
  <!ELEMENT interest-rate ANY >
  <!ELEMENT term-of-loan ANY >
  <!ELEMENT output ANY >
  <!ELEMENT monthly-payment ANY >
]>

<at-test name="calc-monthly-payment" version="1.0" severity="CRITICAL">

  <at-project>mortgage-calc</at-project>

  <at-description>
    Enter loan amount, interest rate, term of loan (in months)
    to calculate monthly payment.
  </at-description>

  <at-data-sets>
    <at-struct id="values">
```

```
      <input>
      <loan-amount>1000000000.00</loan-amount>
      <interest-rate>0.5</interest-rate>
      <term-of-loan>1200</term-of-loan>
    </input>
    <output>
      <monthly-payment>A big, fat wad of dough!</monthly-payment>
    </output>
      </at-struct>
 </at-data-sets>

 <at-plan>

  <at-step name="populate-loan-amount">
    <at-action>
      <at-text>Enter "{0}" in the "Loan Amount field".</at-text>
      <at-value dset="values" select="/input[2]/loan-amount"/>
    </at-action>
    <at-expect>
      <at-text>Cursor moved to "Interest Rate" field for input.</at-text>
    </at-expect>
  </at-step>

  </at-plan>

</at-test>
```

QW2001 Keynote 10P2

Mr. David Lilly
(siteROCK Corporation)

Internet Quality of Service (QoS): The State Of The Practice

Key Points

- There is a "Crisis of Quality" in the Internet Economy
- No major corporation in the world today operates without some reliance on the Internet: for communications with & application delivery to employees, customers, suppliers, and industry partners; the sale of products and services; or after-the-sale support.
- Given this reliance on the web and Internet technology to support their internal IT systems - A Web site or IT infrastructure that is down or performing poorly can cost company productivity, revenues, and worse, can damage the relationship between a business and its customers or partners.
- Dave Lilly will demonstrate how the innovative monitoring, measurement and management services delivered by siteROCK Corporation are enabling the enterprise to address this quality challenge head on.

Presentation Abstract

In a world increasingly dependent on information technology, there is a growing "crisis of quality." Every major corporation in the world relies on IT infrastructure and Internet connectivity for critical aspects of its communication with employees, customers, and industry partners; development and sale of products and services; and after-the-sale support. Given this reliance, infrastructure that is down or performing poorly can cost company productivity, revenues, and worse, can permanently damage the relationship between a business and its customers or partners: this is the crisis of quality.

Dave Lilly will address how IT best practices can help corporations meet this challenge and avert potential crisis. Today, business success is built on the success of the IT enterprise and its integration with the Internet and other external systems. Lilly will demonstrate that best practices are the cornerstones of a strong IT foundation, and he will describe five specific best practices for efficient monitoring, measurement and management that can help build quality into the IT process.

About the Author

Dave Lilly, Chief Operating Officer of siteROCK Corporation, has over 20 years of technical-based project and account management experience, including heading

projects for major industry leaders such as IBM, Sun Microsystems, General Electric, DEC, and General Motors. Dave served most recently as the COO of NONSTOP Solutions, Inc., a supply-chain technology provider. Additionally, he founded and managed the largest division of Integrated Automation and was president of Teknekron Customer Information Solutions, an imaging and call center service provider. Dave holds a Bachelor of Science from the U.S. Naval Academy and an MBA from Rutgers University School of Business.

Internet Quality of Service (QoS): the State of the Practice

Mr. Dave Lilly, Chief Operating Officer, siteROCK Corporation

Abstract

In a world increasingly dependent on information technology, there is a growing "crisis of quality." Every major corporation in the world relies on IT infrastructure and Internet connectivity for critical aspects of its communication with employees, customers, and industry partners; development and sale of products and services; and after-the-sale support. Given this reliance, infrastructure that is down or performing poorly can cost company productivity, revenues, and worse, can permanently damage the relationship between a business and its customers or partners: this is the crisis of quality.

Dave Lilly will address how IT best practices can help corporations meet this challenge and avert potential crisis. Today, business success is built on the success of the IT enterprise and its integration with the Internet and other external systems. Lilly will demonstrate that best practices are the cornerstones of a strong IT foundation, and he will describe five specific best practices for efficient monitoring, measurement and management that can help build quality into the IT process.

Biography

Dave Lilly, Chief Operating Officer of siteROCK Corporation, has over 20 years of technical-based project and account management experience, including heading projects for major industry leaders such as IBM, Sun Microsystems, General Electric, DEC, and General Motors. Dave served most recently as the COO of NONSTOP Solutions, Inc., a supply-chain technology provider. Additionally, he founded and managed the largest division of Integrated Automation and was president of Teknekron Customer Information Solutions, an imaging and call center service provider. Dave holds a Bachelor of Science from the U.S. Naval Academy and an MBA from Rutgers University School of Business.

QW2001 Keynote 10P3

Mr. Thomas Drake
(Integrated Computer Concepts, Inc.)

Riding The Wave -- The Future For Software Quality

Key Points

- Quality for network centric systems
- Software quality engineering
- Future of quality and testing

Presentation Abstract

How will the software quality market evolve over the next few years? What is at stake? What is it going to take?

Internet, the Web, and e-Business are increasingly demanding higher and higher levels of quality for network-based software systems with less and less mean time between failures.

Why? The impact of poor quality and less than robust systems increasingly affect the bottomline for many businesses. A lot has happened to improve the situation and new methodologies and new tools for improving software quality have emerged in the last few years, but I would suggest that radical new approaches and initiatives must be created and adopted if the desirable quality levels to support the e-Economy are to be truly realized and especially at the Internet and global level.

What would that look like? It will take a combination of component-based development, design by contract, specification-based testing, and statistical process control. It will require and even demand much higher levels of predictive and profiling analysis. It will also demand enterprise level and even systems thinking as more and more complex "web-enabled" applications are deployed into and amongst various market spaces. So we have the twin challenges of faster and faster delivery and deployment times requiring higher and higher levels of quality. All of these changes are placing great pressure on the traditional ways of testing and viewing quality. The 'target" customer is no longer just the QA or test group. Increasingly what is demanded is a business and enterprise level focus in addition to the technical.

Moreover, these challenges are not only technical but also cultural in nature and it may be useful to describe at a survey level this "new" future in terms of the new initiatives that are now increasingly required including component-based

development and applied software engineering, specification-based testing, test coverage and analysis technology, design for test principles, various predictive software and profiling analysis techniques and approaches, and full life-cycle application testing activities and content quality and even customer “experience” methodologies as well as a concentrated focus and emphasis on the various quality and testing “states” as part of these initiatives. The future of quality demands nothing less.

About the Author

Mr. Drake is a software systems quality specialist and management and information technology consultant for Integrated Computer Concepts, Inc. (ICCI) in the United States. He also consults to industry and government on quality management and software quality engineering and code development issues. As part of an industry and government outreach/partnership program, he holds frequent seminars and tutorials covering code analysis, software metrics, OO analysis for C++ and Java, coding practice, testing, best current practices in software development, the business case for software engineering, software quality engineering practices and principles, quality and test architecture development and deployment, project management, organizational dynamics and change management, and the people side of information technology.

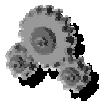
He is the principal author of a chapter on “Metrics Used for Object-Oriented Software Quality” for a CRC Press Object Technology Handbook published in December of 1998. In addition, Mr. Drake is the author of a theme article entitled: “Measuring Software Quality: A Case Study” published in the November 1996 issue of IEEE Computer. He also had the lead, front page article published in late 1999 for Software Tech News by the US Department of Defense Data & Analysis Center for Software (DACS) entitled: “Testing Software Based Systems: The Final Frontier.” He is also one of the featured leading computer scientists interviewed in the textbook entitled: Problem Solving, Abstraction, & Design Using C++, Third Edition, 2000 by Friedman and Koffman from Addison Wesley Longman. Mr. Drake is a member of IEEE and an affiliate member of the IEEE Computer Society. He is also a Certified Software Test Engineer (CSTE) from the Quality Assurance Institute (QAI).

- Riding the Wave -
The Future for Software Quality

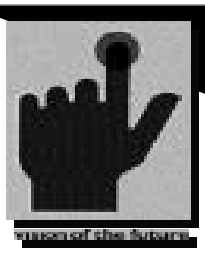
Thomas A. Drake
Quality Architect/Software Anthropologist
Enterprise Management and Information Technology Consulting
Certified Software Test Engineer (CSTE)

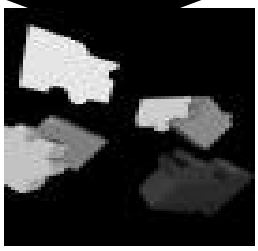
International Internet & Software Quality Week 2001 - San Francisco

Integrated Computer Concepts, Inc. (ICCI)
http://www.integratedcc.com
thomas.drake@integratedcc.com

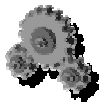


Facing the Real Quality Challenge...





Occam's Razor - ***With all things being equal, the simplest explanation tends to be the right one.***



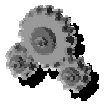
Is the Surf Up or Down, Dude?!

What is going on?

So what difference does software quality REALLY make?

Just make sure you have a real business plan, first!

Survey...



An Historical Perspective...

"The conveniences and comforts of humanity in general will be linked up by one mechanism, which will produce comforts and conveniences beyond human imagination. But the smallest mistake will bring the whole mechanism to a certain collapse."

- Pir-o-Murshid Inayat Khan, 1922 (Tasawuuf leader)

Running Rampant in Cyberspace

Missing the Obvious

Forgetting the Fundamentals

Shortened Product Release Schedules

"Mass Customization"

Testing Never Stops when Content is Continuously Changing

Web Time is ALL the Time!

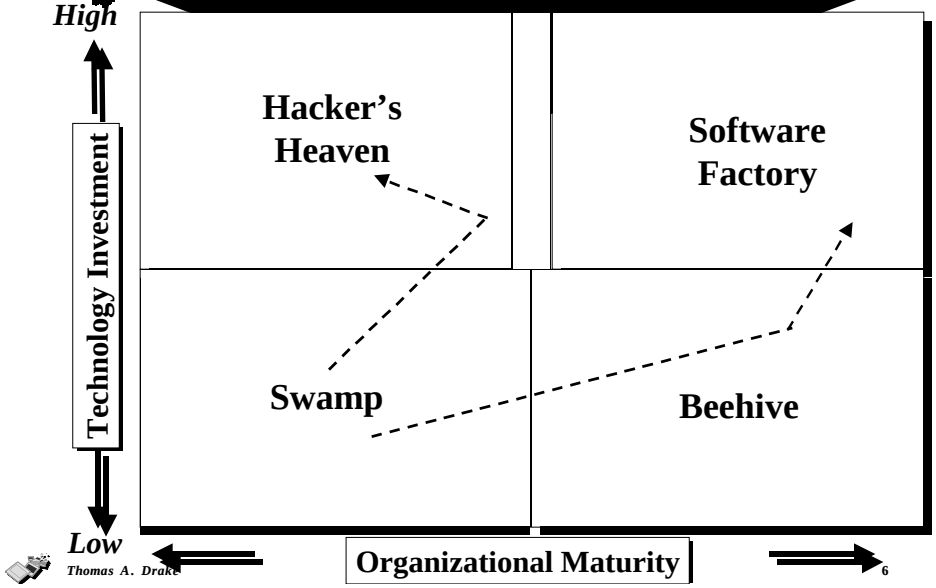
What's Missing "Big Time" ?

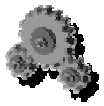
Systems Thinking and Perspective!

Where am I from?!

My Request of You...

The Software Development Challenge: "Easy to Hack Out - Easy to Hack In..."

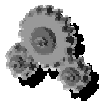




Software - NOT a Naturally Occurring Phenomenon!

If you exclude the time it takes to learn, the money that it take to train, the elusive reuse benefits, the resistance to change, the constantly arising trouble spots, the frequent upgrades, the long lead times required to build applications from scratch, the complex programming languages, the lack of scalability, the shortage of talent, the performance penalties, the deployment challenges, the heavy maintenance burdens, the difficulty in comprehension, and the expense of manually reapplying one's customization, then software technology is quite beneficial. :-)

Completely dependent on who we are as human beings! After all it is us humans doing the creating! A product of our fertile imaginations and intellect!

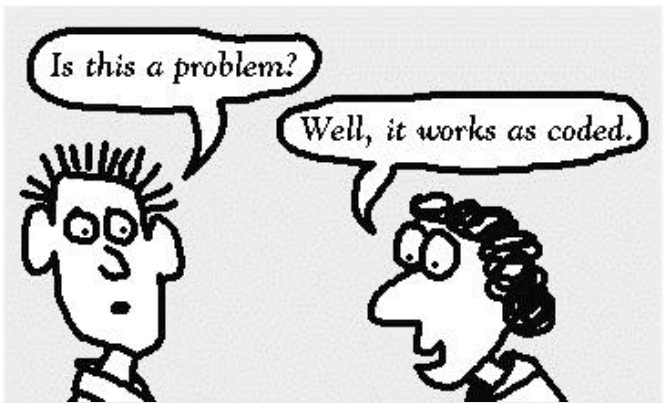


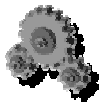
Sobering Numbers

- ✓ **Standish Group Study…**
  - ✓ **Over $250 billion spent annually on IT application development**
  - ✓ **31% of all projects are cancelled before completion**
  - ✓ **88% of all projects are over schedule, over budget, or both**
  - ✓ **For every 100 projects started, there are 94 restarts**
  - ✓ **Average cost overrun is 189% of original estimates**
  - ✓ **Average time overrun is 222% of original estimates**
- ✓ **Even with strong technical skills many project managers and project team members find themselves in over their heads**
  - ✓ **On projects out of control**
  - ✓ **Without the necessary business, organizational and political skills**
- ✓ ***Faith is not a management method*! *Oh, DUHHH*!!**

Lack Of Quality - The Epidemic Of Buggy Software

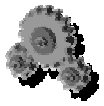
- ✓ ***The Software Conspiracy*** **by Mark Minasi**
- ✓ **We are all guilty! Would you unplug your automated toaster after every 6 slices of bread just to reset the internal software?!**
  - ✓ **The myth that bug-free code is not possible**
- ✓ **Software publishers/contractors STILL aren't generally interested in producing stable, functionally fit, error free software**
  - ✓ **It is features, not quality, that sells! And we buy!**
- ✓ **By hiring the "best and the brightest" we may actually be sabotaging our own efforts - it's embedded in the culture**
  - ✓ **It is the boring but absolutely necessary work that does it!**
- ✓ **Time to emphasize quality - Remember the car industry in the USA in the late 50s, 60s, and early 70s??**
  - ✓ **Also see Jeremy Main's book -** ***Quality Wars***
- ✓ **And it's the business side of software quality that gets us in trouble!**





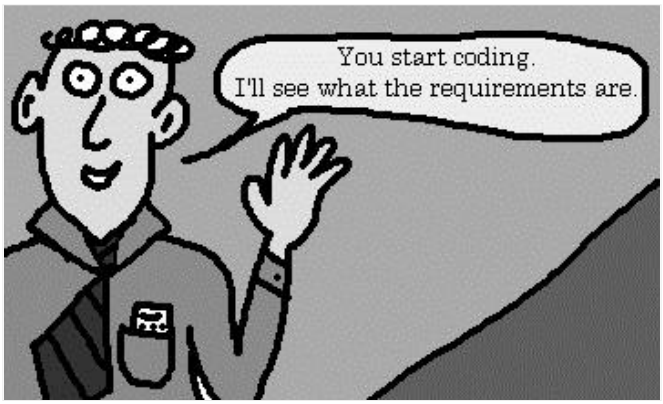
Conventional Wisdom For Software Quality Engineering :-)

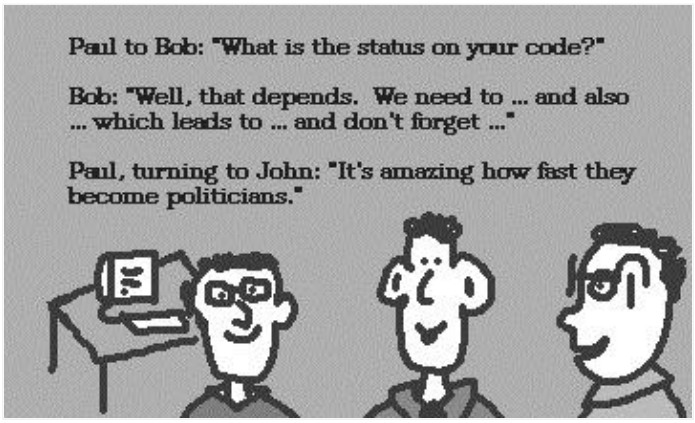
- ✓ **Software Engineering Guidebook on Terminology, v2.0n**
- ✓ **NEW: Different colors from previous version.**
- ✓ **ALL NEW: Software is not compatible with previous version.**
- ✓ **UNMATCHED: Almost as good as the competition.**
- ✓ **ADVANCED DESIGN: Upper management doesn't understand it.**
- ✓ **NO MAINTENANCE: Impossible to fix.**
- ✓ **BREAKTHROUGH: It finally booted on the first try.**
- ✓ **DESIGN SIMPLICITY: Developed on a shoe-string budget.**
- ✓ **UPGRADED: Did not work the first time.**
- ✓ **UPGRADED AND IMPROVED: Did not work the second time.**

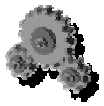


Living in a state of constant ambiguity...

Software Failures Can Lead to Financial Catastrophe


You start coding.
I'll see what the requirements are.

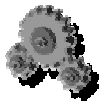

Paul to Bob: "What is the status on your code?"
Bob: "Well, that depends. We need to ... and also ... which leads to ... and don't forget ..."
Paul, turning to John: "It's amazing how fast they become politicians."



What's Happening?

- ✓ **Dynamic and New Information Sources**
  - ✓ **Global Information Network**
  - ✓ **Diversity of Telecommunication Alliances and Global Arrangements/Alignments**
  - ✓ **New Telecommunication Technologies**
  - ✓ **Explosion of Wireless, IP Telephony, Virtual Numbers and more!**
  - ✓ **Internet and Beyond (Dynamic Roaming)**
- ✓ **New Information Technologies**
- ✓ **New Protocol Structures**
- ✓ **Massive Interdependencies**

- ✓ **Software can be technically correct, but still not succeed**
- ✓ ***All place HUGE demands and MASSIVE Strain on Quality!***



Where Are We Going?

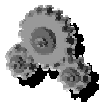
- ✓ **Integrated Component Design and Code**
- ✓ **Software Reuse (domain/pattern level)**
- ✓ **Network Common Services**
- ✓ **XML & Web-Enabled Technologies**
- ✓ **Intelligent Agents**
- ✓ **Data Visualization - (2-D versus 3-D)**
- ✓ **Dynamic Security/Roaming Accounts**
- ✓ **Private/Public Network/P2P**
- ✓ **Web-based Workflow/Content Updates**

- ✓ **Extreme Programming/Rapid Application Network Testing(RANT)!**
- ✓ **AutoCode Generation/Design Logic Engines**
- ✓ **Continuous Testing/Experiential-Based/Event-Based Testing**
- ✓ ***The "Old" QA Paradigms Come Up Way Short!***

Differentiator for quality is less the technology than the difference the technology makes!

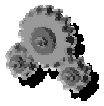
And Only Getting MORE Connected

- ✓ **The future is not what it used to be!**
- ✓ **Pace of change (Default Standard)**
- ✓ **Time, communications, space (compressing)**
- ✓ **Implication - Speed of light access and impact**
- ✓ **All linked together in one dense, interconnected web of information and data!**
- ✓ ***Places HUGE demands on QUALITY!***



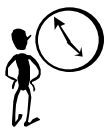
Internet/Cyberspace Quality

- ✓ **Competing for shelf space versus competing for Web space**
  - ✓ **Producing a shift toward the Web and more and more testing of internet-based/network-based software**
- ✓ **More complex programs and applications using Java and Shockwave and Flash are emerging**
- ✓ **Testing on multiple platforms and operating systems**
  - ✓ **Different Internet service providers and methods of connecting to the Internet**
  - ✓ **Can't afford to put out a shoddy product on the Web**
- ✓ **Quality is STILL quality and even more so on the Web**
  - ✓ **Stakes are much higher in this kind of "operating" environment and bugs/problems/defects are much more visible**



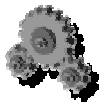
Quality Makes a Difference!

- ✓ **Issue of market cycle time and shelf life**
- ✓ **Problem of code entropy**



- ✓ **Impact of incremental patching and upgrading**
- ✓ **Doing it right the first time - interface design is everything**
- ✓ **Simple and elegant solutions - stand the test of time and the marketplace**

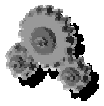
- ✓ ***Quality is, as quality does!***



Best Practices - The Use of Domain/Design Patterns (1)

- ✓ **Set of objects with certain known roles and responsibilities**
  - ✓ **Relationship to each other**
  - ✓ **Common usage**
  - ✓ **Prerequisites**
  - ✓ **Cataloged/documented**
  - ✓ **Refinement/updates/extensibility**

- ✓ **Emerging due to Internet time/intense schedules**
- ✓ **Program structures are fundamental**
  - ✓ **Where are the execution cycles**
  - ✓ **"The most efficient instruction set is the one that's never executed!"**
  - ✓ **Provides for the abstraction that provides summary/overview**

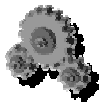


Best Practices - The Use of Domain/Design Patterns (2)

- ✓ **Patterns should also identify the intended use audience**
  - ✓ **Provide for the external and internal assumptions**
- ✓ **Document/Document/Document! (1)**
  - ✓ **Name, problem, context, constraints, trade-offs, static relationships and dynamic rules/behavior, variants/specializations**

  - ✓ **Examples - sample implementations**
  - ✓ **Known uses - describes known occurrences of the pattern and its relationship and application within existing systems**
- ✓ **Generalize! - How can the program be developed such that it minimizes the code interdependencies among the various subsystems?**

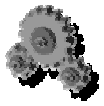
(1) Source: Patterns and Software: Essential Concepts and Terminology - Brad Appleton (www.enteract.com/~bradapp/docs/patterns-intro.html)



The Power of Patterns

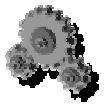
- ✓ **Patterns...**
  - ✓ **Patterns provide a relationship between actions and then "embody" it in a design artifact**
  - ✓ **Ultimately facilitates the reuse of code and design**
  - ✓ **If clean design is the most difficult and important step in software engineering, there could be real benefits in adopting**

- ✓ **Where do patterns come from?**
  - ✓ **Tangible: Architecture/materials**
    Intangible: process/procedures
- ✓ **A domain pattern "sensitivity" problem...**
  - ✓ **Software development culture is still dominated by "bottom-up" thinkers (trees/forests)**
  - ✓ **For patterns, you need to think "top down" and not just on the bit level but the aggregate of bits (atoms/molecules)**



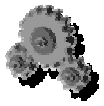
So What Is Software Quality Engineering Really All About??

- ✓ **"...Too many organizations have spent too much time obsessing on the information they want their networks to carry and far too little time on the effective *relationships* those networks should create and support. This is a grave strategic error."**
  - ✓ **Michael Schrage, MIT Media Lab Research Associate, in a white paper of The Merrill Lynch Forum**
- ✓ **The language is not THE answer! But the language is a primary means of communication with its own syntax, structure, rules and meaning.**
  - ✓ **Design by Contract - Precise definitions/relationships**
  - ✓ **And it shows in the Code!**
  - ✓ **Rigor and discipline are fundamental**
  - ✓ **And it is the quality of the software "experience" that may be the real measure of quality!**
- ✓ **Huh? What is he talking about? Listen to your customers!**



Quality Communication
It's All About People!

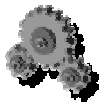
Customer, Customer, Customer!



The Future of Quality
Into the Looking Glass! (1)

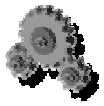
- ✓ **Moving away from large development programs**
  - ✓ **Fixed set of ideas, very difficult to maintain or modify**
- ✓ **Virtual computing -- 24/7, 365 & around the world!**
- ✓ **Component-based development**

- ✓ **Design logic to code!**
  - ✓ **An increasing trend (code engines)**
  - ✓ **Solves many of the "hand" created complexity problems**
- ✓ **Contract-based/specification-based outcomes**
- ✓ **Test and Quality By Design Principles (Design by Contract)**



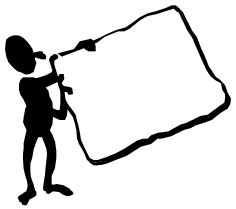
The Future of Quality
Into the Looking Glass! (2)

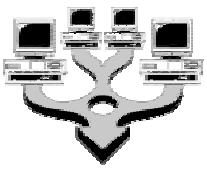
- ✓ **Do not be overly transfixed by C++ or OO or Java or even the very latest and greatest! Still way too many buzz words and ambiguity!**
- ✓ **Real-time operations across multi-distributed and heterogeneous environments**
- ✓ **The network IS increasingly becoming THE program - just get it!**
- ✓ **Key to software development innovation is business savvy, smart developers, and high quality products that meet and exceed customer expectations and still meet the bottom line!**
- ✓ **Managing in this environment requires a "paradigm shift"**

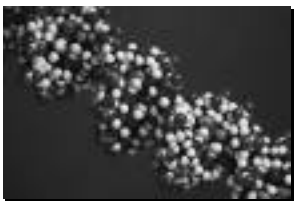


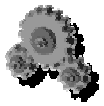
The "Genetic"
Heart and Core of Quality

- ✓ **Design is fundamental!**
- ✓ **Business Rule Algorithms**
- ✓ **Behavioral modeling**
- ✓ **Attributes - The key for quality**



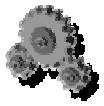




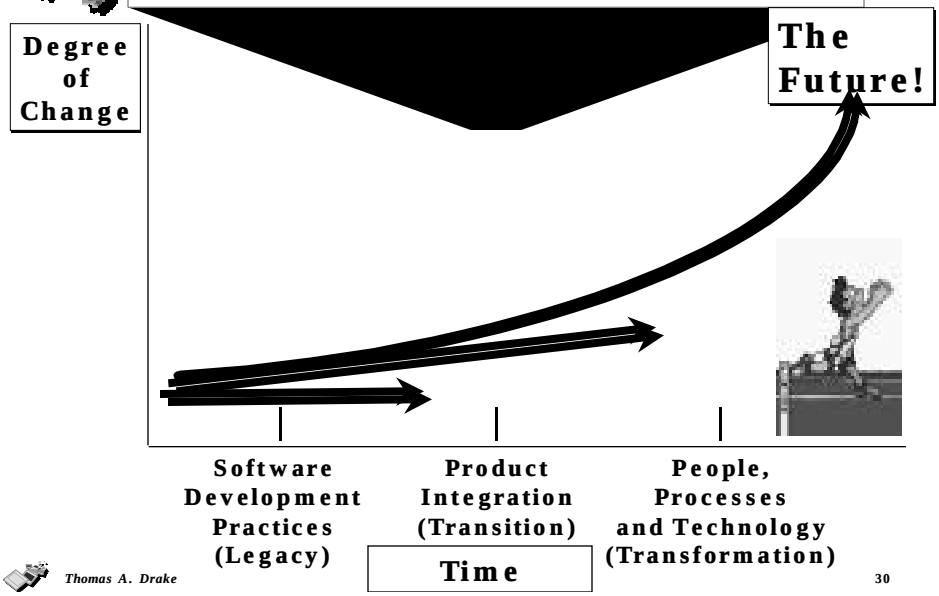


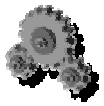
The Continuing Software Quality Challenge

- ✓ Taming the "uncontrolled" distribution of data
- ✓ In software, data is an abstraction
- ✓ Software's strength IS abstraction!
- ✓ Concrete representation vice the abstract notion
  - ✓ How do you build software??
  - ✓ Concept by concept or brick by brick??
- ✓ Agree on the concept!
- ✓ Information hiding is critical
  - ✓ Each module must only access the information it needs
  - ✓ And every software element must have a specification
  - ✓ Use Design by Contract - architecture is critical!

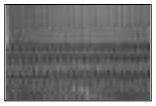


Quality Paradigm Shift Where Do You Live on the Continuum?!

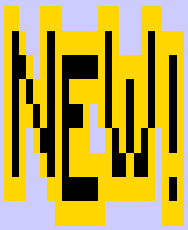




Some Final Thoughts About Quality and Software



- ✓ **Dealing with multiple levels of complexity**
  - ✓ **Quality enemy #1 - orders of magnitude greater**
- ✓ **Nth Order Computing (modeled after cell life)**
  - ✓ **Computer is 1st order - computational!**
  - ✓ **Autonomous objects/components are 2nd order**
  - ✓ **Interaction of objects is 3rd order - no object alone**
  - ✓ **Result/outcome/state transition is 4th order**
  - ✓ **Nth Order is shifting function onto structure and code onto data - *Built in dynamic living networks - It's alive!***
- ✓ ***Computers as machines containing cyberspace containing machines* - (Bandwidth, Turnaround Time, and Complexity)**
- ✓ **So... Are you now ready for this brave new world?! :-)**



QW2001 Paper 6V1

Ms. Lauri MacKinnon
(Vanteon)

Three Customer Case Studies of Performance Testing using Segue SilkPerformer

Key Points

- Techniques for capturing, storing, and using multiple dynamically created Session IDs and other keys
- A method of customizing IP spoofing
- Overcoming difficulties encountered with implementing secure transactions
- Challenges of recording and replaying encrypted, multi-port TCP/IP traffic
- A method for sending load traffic directly to a database server

Presentation Abstract

Three customer case studies of performance testing using Segue SilkPerformer will be presented. Challenges presented for each case study, the techniques used to meet the challenges, and actual test results and analysis will be reviewed. The presentation will highlight our solutions in to challenges involving dynamic key parsing, handling multiple session IDs, directing traffic with IP spoofing, adding secure transactions and more. Result charts will help provide analysis the circumstances when the customer web sites and applications were put to the test. Participants will see a variety of scripting techniques and real world examples of test results and analysis using Segue SilkPerformer.

About the Author

Lauri MacKinnon is a performance and load testing specialist in Vanteon's automation group. She is a Segue-certified eConfidence Performance Consultant and has been using Segue SilkPerformer for a year and half. She has used other automation tools (including QA Partner and SilkTest) for eight years and has been in the QA industry for twelve years. Lauri has a Master's degree in Computer Science and applies her programming background toward automation scripting for Vanteon's clients.

SilkPerformer Case Studies

Presented To

Quality Week 2001
San Francisco, CA
May 31, 2001

Presented By

Lauri MacKinnon
QA Advanced Development Engineer,
Vanteon

VANTEON

Delivering Innovative Engineering Solutions

Vanteon Overview

- A full service development and Quality Assurance consulting company
- Vanteon specializes in Quality Assurance performance testing services
- Vanteon is partnered with Segue to create performance and load tests for clients using Segue SilkPerformer

VANTEON

Delivering Innovative Engineering Solutions

Case Studies

- Review three case studies of performance testing engagements with Segue SilkPerformer
- Identify challenges presented by each case study
- Show techniques developed to solve each challenge
- Review actual test results from Segue SilkPerformer

VANTEON *Delivering Innovative Engineering Solutions*

Case Study #1

- A web–based brokerage and investment solution for the financial community. The site is intended for use by brokers to manage client accounts and portfolios.
- This was a first prototype of the site, with no real-time trading incorporated.

VANTEON *Delivering Innovative Engineering Solutions*

Goals

- Determine scalability of the site with up to 1000 concurrent users
- Create a matrix of resource requirements for different levels of response times and numbers of users
- Make recommendations for changes to the application infrastructure and hardware requirements
- Determine performance of Oracle database
- Establish a testing, feedback, and site enhancement sequence

VANTEON

Delivering Innovative Engineering Solutions

Requirements

- Ensure correct flow of transaction execution by users on the site for meaningful results
- Run scripts to clear out the database between load tests
- Create additional custom timers for measuring load times for specific pages on the web site
- Check performance of secure areas of the site (using SSL and 128 bit encryption)
- Additional testing requests: Error Recovery, Uptime/Availability, Load Balancing Effectiveness, Distributed Object Performance, Adequate Bandwidth, End-to-End Functionality, and Competitive Positioning
- Client confidentiality is critical in the financial industry

VANTEON

Delivering Innovative Engineering Solutions

Challenge – Unique Logins and Session IDs

- Create unique login accounts for each user
- Capture dynamically generated session IDs
- Recapture Session IDs as they change during the user's session

Technique – Building a Unique Login

- For each user, the scripts needed to:
  - log in as a unique broker (broker ID is read in from a spreadsheet file)
  - create a new client in the system, using a unique client name (generated at runtime by a random variable) and a unique Social Security Number (read in from a spreadsheet file)
- Social Security Numbers used must not already exist in the system or be in use by manual testers

Technique – Capturing Session IDs

Session IDs for each user were tricky to capture:

- IDs were not visible in SilkPerformer record or replay log files
- IDs were dynamically created at runtime by a function on a page in the web site

Technique – Capturing Session IDs

Parsed the source code, log files, and output files:

- Captured IDs as they were passed from the web server to the client browser when brokers log into the system
- Recaptured in later transactions as they changed value (even during the same browser instance) and printed to output files
- Compared values against those in log files to find changes
- Isolated which lines of the script caused the new value for the ID to be sent from the web server to the client machine
- Used parsing functions to get the ID, store it in a variable, and substitute in URLs and web forms

Challenge – Dynamic Keys

- Each simulated user needed to: log in as a broker, create an account for a client, view the client's details, edit the client's profile, add goals for the client, create portfolios and scenarios for the client, assign holdings into the portfolios, and run portfolio analysis
- The load testing scripts needed to capture, parse, store, and use several dynamically-created keys while executing these transactions for each unique user:
  - Broker keys
  - Asset keys
  - Account keys
  - Allocation keys
  - Portfolio keys
  - Client keys
  - Stock keys
  - Scenario keys

VANTEON

Delivering Innovative Engineering Solutions

Challenge – Dynamic Keys

Complications:

- Several of the keys' values are interdependent:
  - Portfolio keys are dependent on client keys and broker keys
  - Allocation keys are dependent on stock keys and asset keys
- Portfolios could contain several assets, and assets could be contained in multiple portfolios

VANTEON

Delivering Innovative Engineering Solutions

Technique – Capturing Dynamic Keys

- Scan through SilkPerformer's record and runtime log files and site page source code for dynamically generated keys.
- Use Silk Performer's WebParseResponseHeader and WebParseResponseData functions to capture data coming from the web server in response to WebFormGet, WebFormPostEx, and WebURL requests.
- Use SilkPerformer's SubString and StringSearchDelimited functions to parse the keys out of the response string and store them in variables. Check boolean responses to verify we had correctly captured the keys.
- Substitute variables for hard-coded keys in web forms and URLs. There were 187 occurrences of using the parsing functions and 104 occurrences of substituting keys in forms in this script.

Complications

- Midway through scripting the site was moved to a production server
  - Modified scripts to run against the new server address
  - Discovered that only 3 of 10 assets in each portfolio were now being created. The relationship between three of the keys had been changed…
- Also midway through scripting the client requested some changes:
  - Increase the transactions that each user executes from 6 to 39
  - Create 5 portfolios for each client instead of 1
  - Further diversify the assets assigned to the portfolios
- The format required for asset keys in web forms was different than the format they were captured in. String manipulation functions were used to concatenate keys into the correct format.
- Delivered scripts to the customer's developers to review for correct flow and to indicate any other keys we needed to capture.

VANTEON™ *Delivering Innovative Engineering Solutions*

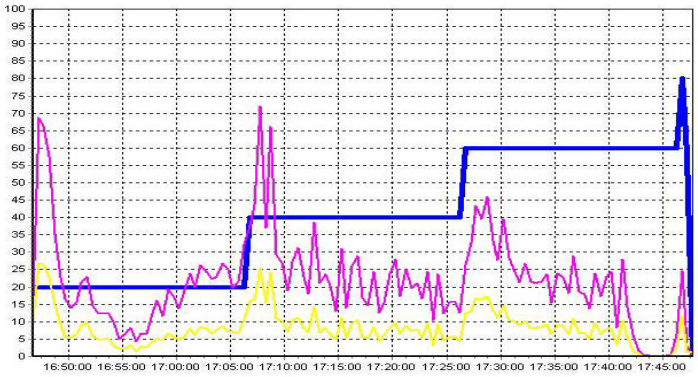
Analysis: 60 Concurrent User Tests

- Ramp up to 60 concurrent users
- Request Data Sent spikes as new users are added
- HTTP Hits spike as new users get pages not in the cache, and falls as they get pages already in the cache
- Graph illustrates good behavior of the site

VANTEON

Delivering Innovative Engineering Solutions

60 Concurrent User Test



	Group	Name	Type	Factor	Source	Unit	Count	Min	Avg	Max	StdDev	Int.
—	Controller	Controller	Active users	1	Average		123	0.000	40.146	60.000	16.994	30
—	Controller	Controller	Request data sent[kb] / sec	10	Sum		123	0.000	2.154	7.200	-	30
—	Controller	Controller	Http hits / sec	1	Sum		123	0.000	8.048	26.800	-	30

VANTEON

Delivering Innovative Engineering Solutions

Analysis: 100 Concurrent User Tests

- Initial 100 User Load Test:
  - Drop off in activity between 25 and 32-minute marks
  - Errors start appearing after response data throughput dropped to 0
  - Transactions start to time out at 32-minute mark
  - Manual observation – individual page loads took between 30 seconds to 2 minutes
- Another 100 User Load Test:
  - Transaction completion times increased significantly after 50 users. With a 50 user load the transaction cycle to 10 minutes to complete. With a 100 user load it took 30 minutes to complete.
  - The site did not crash

VANTEON *Delivering Innovative Engineering Solutions*

Analysis: 200 Concurrent User Test

- Application server died at 29 minutes (with 140 concurrent users). Indications:
  - Errors started appearing
  - Transactions started to time out
- Web Server did not crash

VANTEON *Delivering Innovative Engineering Solutions*

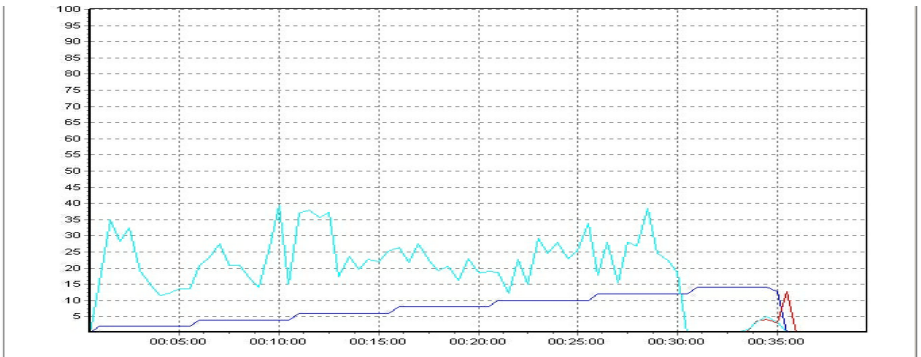
Analysis: 140 Concurrent User Test

- To isolate the problem noted with 140 users we ran an additional test with a more gradual ramp-up to the peak load
- Users were increased from 20 to 140 in increments of 20
- An error spike occurred just before the end of the test
- The response data received drops at the last increase in the number of users (to 140)

VANTEON

Delivering Innovative Engineering Solutions

140 Concurrent User Test



	Group	Name	Type	Factor	Source	Count	Min	Avg	Max	StdDev	Int
	Controller	Controller	Active users	0.1	Average	70	0.000	77.800	140.000	40.174	30
	Controller	Controller	Transactions	1	Sum	70	0.000	0.000	0.000	-	30
	Controller	Controller	Errors	0.1	Sum	70	0.000	3.063	126.000	-	30
	Controller	Controller	Response data received[kb]	0.01	Sum	70	0.000	1748.291	3935.000	-	30

generated at 2000-06-05 12:09:32
(c) Segue Software Inc.

VANTEON

Delivering Innovative Engineering Solutions

Case Study #1 Summary

- Ran and refined load tests over a two-week period
- The client made progress in fixing areas of the site based on our feedback
- The company was purchased and stopped doing performance testing

VANTEON

Delivering Innovative Engineering Solutions

Case Study #2

- A web–based solution for submitting insurance claim forms for the commercial trucking industry that allowed clients to submit insurance claims and review their current safety statistics
- The goal: to determine the scalability of the new web site with up to 250 concurrent users
- Requirement - only a limited number of connections could be made for each IP address hitting the web server, so each virtual user needed to have a unique IP address.
- Site contains both non–secure and secure features and uses SSL3, HTTP and HTTPS protocols
- New and existing users need to perform transactions on different areas of the site simultaneously during the simulation to balance secure and non-secure traffic loads

VANTEON

Delivering Innovative Engineering Solutions

Challenge – Multiple IP Addresses

- Normally when load testing from Silk Performer traffic is funneled through one port using the IP address of the agent machine.
- The web server under test was configured (for security) with a limited number connections allowed from each IP address. When the limit to the number of connections was reached, the web server would shut down or time out.
- Each simulated user was required to have a unique IP address.
- IP Spoofing was used to distribute unused IP addresses from our network to each simulated user.

VANTEON

Delivering Innovative Engineering Solutions

Technique – IP Spoofing

- One method - use multiple machines, each running a SilkPerformer agent, to distribute traffic across the agent machines and hit the web server with multiple IP addresses. Note that the connection limit on the web server can still be reached with high numbers of simulated users.
- IP Spoofing method:
  - Identified IP addresses available for use with IT department. The range had gaps for reserved IP addresses which changed daily.
  - Configured SilkPerformer's System Configuration tool with unused IP addresses
  - Manually modified scripts to read in available IP addresses from a spreadsheet and assign an IP address to each user at runtime

VANTEON

Delivering Innovative Engineering Solutions

Challenge – Changing Site Security

- Handling implementation of certificates, SSL3 protocol and HTTPS protocol midway through the scripting process
- Change in some of the traffic used in several transactions from HTTP to secure HTTPS mode

VANTEON

Delivering Innovative Engineering Solutions

Technique – Adding Secure Transactions

- Users start hitting secure areas of site midway through transactions
- Modified and appended scripts to handle HTTPS and SSL3 traffic and multiple certificates. Rerecorded transactions that introduced new security certificates and changes from HTTP to HTTPS mode for all of the new secure pages in the transactions
- Randomized transactions to mix up secure and non-secure site access
- Monitored performance hit incurred from adding security to the site. Observation: during an initial 50 user test, two of the secure transactions averaged very high average response times of 179 seconds and 128 seconds

VANTEON

Delivering Innovative Engineering Solutions

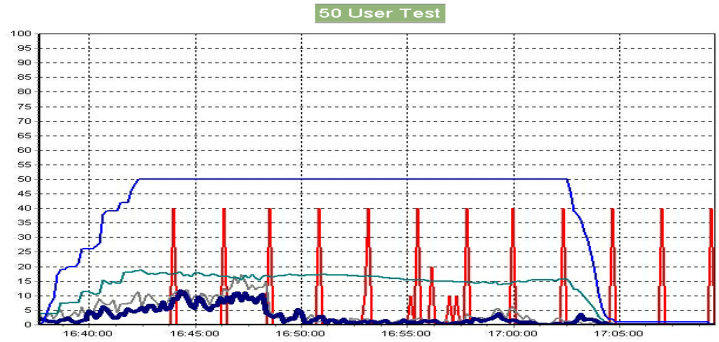
Analysis: 50 Concurrent User Test

- Started with 5 users and ramped up 5 users per minute, as indicated by the blue line
- Timeouts from the web server began to occur at a 50-user load, as indicated in red
- Following the error spike, HTTP hits, concurrent connections, and successful connections began to decline
- Web server CPU peaked at 100% usage and stayed between 60-90% for most of the run
- Transactions took four times as long to execute during the 50-user load test than during a two-user load test
- Note: this test was run against a single processor server

VANTEON

Delivering Innovative Engineering Solutions

50 Concurrent User Test



	Group	Name	Type	Factor	Source	Unit	Count	Min	Avg	Max	StdDev	Int
	Summary General	---	Errors	10	Average		192	0.000	0.281	4.000	0.981	10
	Summary Internet	---	Connects successful / sec	1	Sum		192	0.000	3.765	17.200	-	10
	Summary Internet	---	Concurrent Connections	0.1	Average	state	192	0.000	122.870	188.000	64.039	10
	Summary Web	---	Http hits	0.01	Average		192	0.000	254.333	1155.000	295.114	10
	Summary General	---	Active users	1	Average	state	191	0.000	37.398	50.000	19.343	10

generated at 2000-12-07 10:02:12
(c) Segue Software Inc.

VANTEON

Delivering Innovative Engineering Solutions

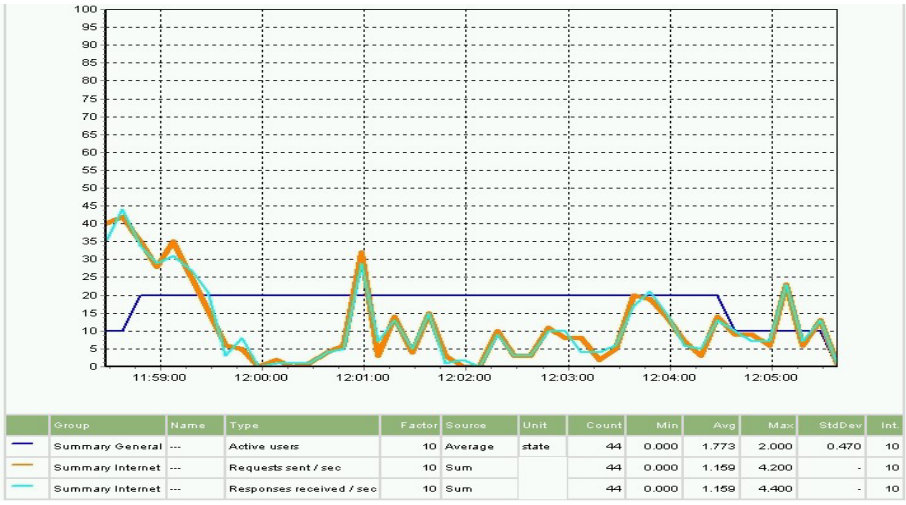
Analysis: 20 Concurrent User Test

- After some changes were made to the site, a smaller 20 concurrent-user test was run.
- This graph show some good, expected behavior during a test:
  - 1 to 1 correlation between requests sent and received from Silk Performer, shown by the light blue and orange lines
  - The web server adequately handled the number requests it received

VANTEON

Delivering Innovative Engineering Solutions

20 Concurrent User Test



	Group	Name	Type	Factor	Source	Unit	Count	Min	Avg	Max	StdDev	Int
	Summary General	---	Active users	10	Average	state	44	0.000	1.773	2.000	0.470	10
	Summary Internet	---	Requests sent / sec	10	Sum		44	0.000	1.159	4.200	-	10
	Summary Internet	---	Responses received / sec	10	Sum		44	0.000	1.159	4.400	-	10

VANTEON

Delivering Innovative Engineering Solutions

Analysis: 250 Concurrent User Test

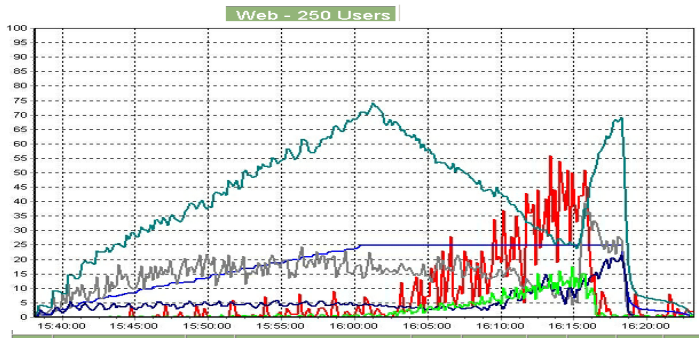
Changes since last run:

- Second processor added to the web server and the database server
- MS IIS web server was tweaked for performance
- Site caching was implemented

VANTEON

Delivering Innovative Engineering Solutions

250 Concurrent User Test



	Group	Name	Type	Factor	Source	Unit	Count	Min	Avg	Max	StdDev	Int
	Summary General	---	Errors	1	Average		400	0.000	5.185	56.000	10.858	10
	Summary General	---	Active users	0.1	Average	state	399	0.000	113.619	249.000	105.785	10
	Summary Internet	---	Connects successful / sec	1	Sum		399	0.000	9.350	39.600	-	10
	Summary Internet	---	Concurrent Connections	0.1	Average	state	399	0.000	267.764	741.000	248.532	10
	Summary Web	---	Http hits	0.01	Average		399	0.000	335.907	2166.000	404.223	10
	Summary Internet	---	Connects failed / sec	1	Sum		370	0.000	1.586	17.500	-	10

generated at 2001-01-11 09:41:15
(c) Segue Software Inc.

VANTEON

Delivering Innovative Engineering Solutions

Analysis: 250 Concurrent User Test

New results:

- Some connection timeout errors occurred at peak load of 250 users
- Successful connections (silver line) stayed fairly consistent
- Web server CPU usage stayed below 30% of the combined processors during the test run
- SQL server maintained around 50% usage of the combined processors during the test run
- Connection errors indicate that not all requests were getting through to the web server. A bandwidth limit was reached on the T1 line. Testing should be moved to a LAN at this point.

Case Study #2 Summary

- The client continued to make enhancements to the site
- The site was launched a month later

Case Study #3

- Client/server solution to enable, expedite and track customer Internet, e-mail and telephone transactions for large scale customer service clients.
- Client requested simultaneous playback of client/server, web, email, telephone, and database transactions.
- Client had experienced slow performance with a low number of users. They requested load testing to help them isolate the problems and fine-tune their setup.

VANTEON

Delivering Innovative Engineering Solutions

Testing Requests

Web and E-mail Contact Performance and Scalability Testing

- Load the system with web and e-mail traffic at the same time and watch the response times
- E-mail load generated by Mustang software
- Execute several distinct load tests by keeping one load variable constant while increasing the other

VANTEON

Delivering Innovative Engineering Solutions

Testing Requests

Internal Client-Server testing

- Executed a Proof of Concept phase to verify that the proprietary protocols used by the client applications (a Java Console and a Remedy Forms Maker client) could be recorded by SilkPerformer
- Resulting scripts were to be used during Internal Client-Server load testing and also in Direct Database testing phase

Testing Requests

Direct Database testing

- Measure response times of transactions run directly against the Oracle database server
- Repeat tests on additional hardware configurations to determine if the database should be moved to a Unix server platform

Testing Requests

- Simultaneous execution of email/web scripts and internal client-server scripts. This was difficult because:
  - SilkPerformer 3.5 could only run one script at a time
  - Only one SilkPerformer MMC / agent machine was available for travel to the client site at the time
- SilkPerformer 4 now supports running user groups from multiple scripts at the same time

VANTEON

Delivering Innovative Engineering Solutions

Other Testing Requests

- Make recommendations for server hardware (CPUs, memory, etc.)
- Computer Telephony (CTI) user simulation during load runs
- Delivery of a formal schedule, plan, and timeline to incorporate into their developers' MS project
- Purchase, configure, train, and use monitoring tools for their Oracle and application servers
- Purchase site licenses for Oracle and Remedy users

VANTEON

Delivering Innovative Engineering Solutions

Challenges

- Record and generate traffic directly to an Oracle database from an application server with no GUI
- Record and play back TCP/IP traffic over multiple ports simultaneously
- Simulate users with different logins in TCP/IP traffic

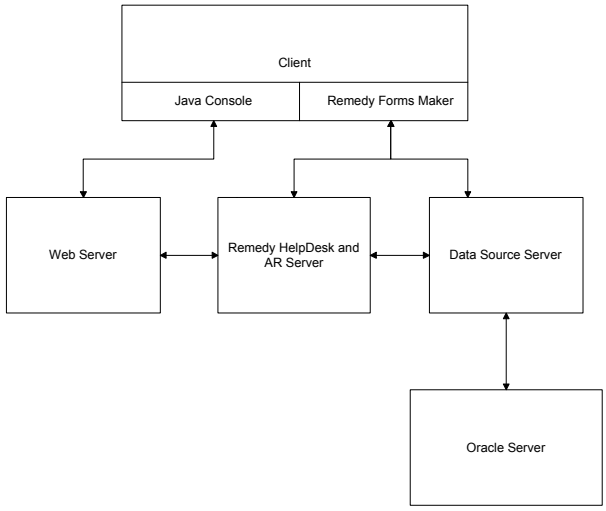
Technique – Direct Database Traffic

Need to force traffic directly to an Oracle server

- Recorded scripts on a client machine (with a GUI)
- Played back traffic from port on a data source server directly to the Oracle server
- Note: replaying the traffic directly to the database server would have resulted in non-realistic response times – the play back would use only one server hop instead of two (client –> application server –> database server)
- The traffic put a heavy load directly on the Oracle server for isolated performance testing

VANTEON
Delivering Innovative Engineering Solutions

Architecture



VANTEON

Delivering Innovative Engineering Solutions

Technique – Direct Database Traffic

- Set up SilkPerformer to use the correct port on intermediary data source server as a proxy.
- Determined that scripts created during internal client-server testing could be reused in direct database testing. This eliminated the need to create new scripts using SQL commands, proprietary DLL calls, and/or API calls to generate ODBC traffic.
- Next step: create scripts to load 300,000 records into the Oracle 8 database prior to testing. Client provided a database template with sample records and an architecture document (E/R diagram).

VANTEON

Delivering Innovative Engineering Solutions

Technique – Multiple Port Traffic

Recording and playing back TCP/IP traffic from multiple ports to several servers simultaneously:

- Verified with Segue that TCP/IP traffic could be recorded from two ports simultaneously in one script
- Tested in a Proof of Concept phase
- Proof of Concept also used to determine if traffic needed to be manually merged in the script to synchronize the actions initiated from the client applications
- Determined that the applications on the client machine coordinated the traffic themselves before sending traffic out through either port – it was not necessary to synchronize actions manually in the scripts

Technique – Multiple Logins in TCP/IP

- For the internal client/server test scripts we needed to simulate multiple users logging in with unique login accounts and using different sets of data records at the same time
- Minimized the dependencies between user interactions by having each user work with a different set of data records

Technique – Multiple Logins in TCP/IP

- No web forms available to customize
- Could not use random or increasing value variables to parameterize accounts because:
  - Traffic was TCP/IP only
  - TCP/IP traffic was encrypted and no decryption formulas were available from the client for use in encoding new login names and decrypting traffic
  - Client could not turn off encryption (Note: turning off encryption would have affected server response times)
- Workaround:
  - Re-recorded transactions several times (once per user group)
  - Set up each user group to run a different set of transactions
  - Executed load tests with one user of each user group type to simulate different logins

Case Study #3 Summary

- Successfully completed first phase of the engagement, verifying that:
  - Traffic can be recorded from the client machine (with a GUI) and played back directly to the database server from the data source server
  - Traffic can be recorded using multiple ports, synchronized, and replayed simultaneously
  - Transactions can be created to simulate multiple login accounts with TCP/IP traffic
- The client company was purchased. The second phase of this engagement is currently on hold

Summary of Techniques Used

- Dynamic key parsing
- Handling multiple session IDs
- IP spoofing
- Adding secure transactions
- Implementing direct database transactions (without a GUI)
- Multiple port recording
- Customizing TCP/IP with unique login IDs

VANTEON

Delivering Innovative Engineering Solutions

Summary of Techniques Used

Questions
&
Answers

VANTEON

Delivering Innovative Engineering Solutions

QW2001 Paper 6V2

Mr. Larry Markosian
(Reasoning)

Improving Software Quality & Delivery Schedules
Through Automated Inspection

Key Points

- The kinds of defects that are detected by InstantQA.
- The benefits of the InstantQA service, and particularly for embedded applications.
- The underlying technology.

Presentation Abstract

Automated source code inspection tools have been available for decades. Only recently, however, has the underlying technology matured enough to pinpoint serious defects-defects that cause an application to crash or corrupt data-without burying these “nuggets” under reams of false positives and low-interest code problems. The technology has been developed by Reasoning to focus on defects such as memory leaks, NULL pointer dereferences, out of bounds array accesses, and other serious defects that delay functional testing and often escape into the deployed application. Especially in the case of mission-critical applications and embedded applications, such defects are a significant cause of delayed releases and expensive failures in the field.

Reasoning’s InstantQA is a software defect detection service based on advanced source code analysis technology that can pinpoint critical defects during development, when they are easiest and least expensive to fix. Early discovery means that the testing cycle is not interrupted to deal with application crashes, unpredictable results and other delays caused by these hard-to-identify bugs. Also, identification by source code analysis provides precise information about where the bug is located, what type of bug it is, and under what conditions will trigger it. This information is usually adequate for even junior level developers with limited knowledge of the application to implement a fix. This is contrasted with the results of testing, where only the symptom of the failure typically can be reported, and long hours on the part of experienced developers may be spent tracking the runtime error to its source.

About the Author

Lawrence Markosian, a founder of Reasoning, Inc., is product manager for InstantQA, Reasoning's automated source code defect inspection service. Prior to

joining Reasoning, Lawrence was a Research Associate at Stanford University, where he specialized in models of mathematical and logical inference and learning. Lawrence is the author of numerous articles on software reengineering, reverse engineering and defect detection, including articles in Communications of the ACM and Java Developers Journal. His email address is zaven@reasoning.com.

Reasoning

Automated Software Inspection

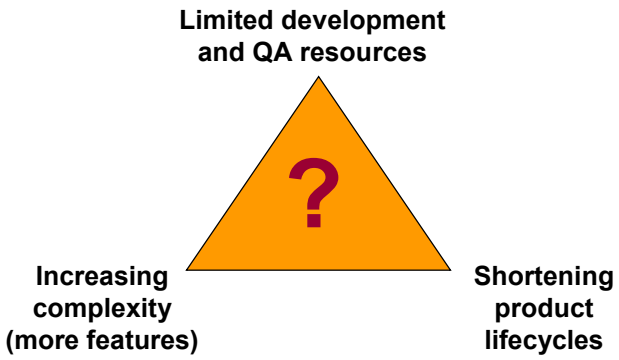
Who is Reasoning?

- **We provide an automated inspection service for companies that develop high-reliability software.**
- **We make them more competitive, enabling them to build better software in less time and at lower cost.**

- Located in Mountain View, CA
- Founded in 1986
- Inspected over 1B LOC

The Problem

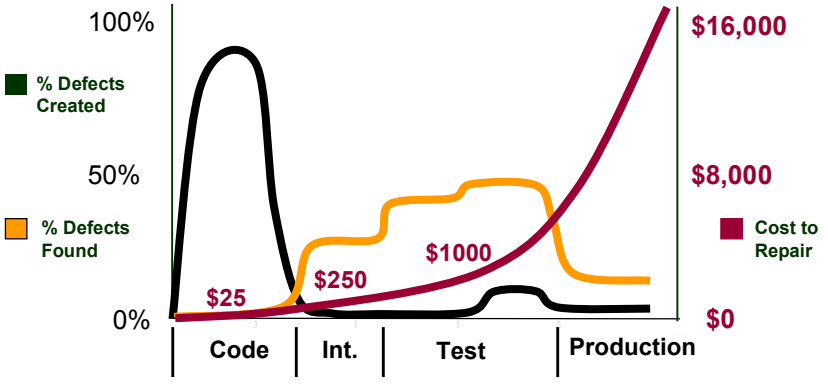
Releasing Reliable Software on Schedule is Almost Impossible



 3

The Problem

Finding Bugs Late is Expensive and Time-Consuming



Source: Capers Jones

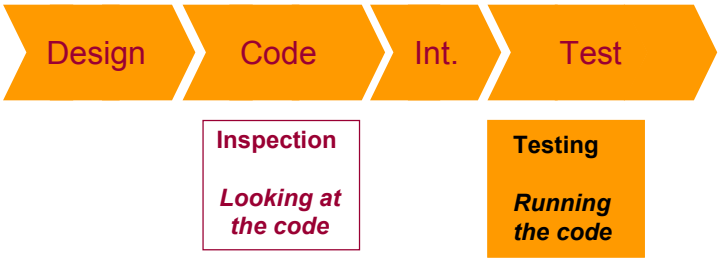
Need to Find Bugs Sooner

 4

The Need

Inspection Finds Bugs Sooner

Software Development Lifecycle

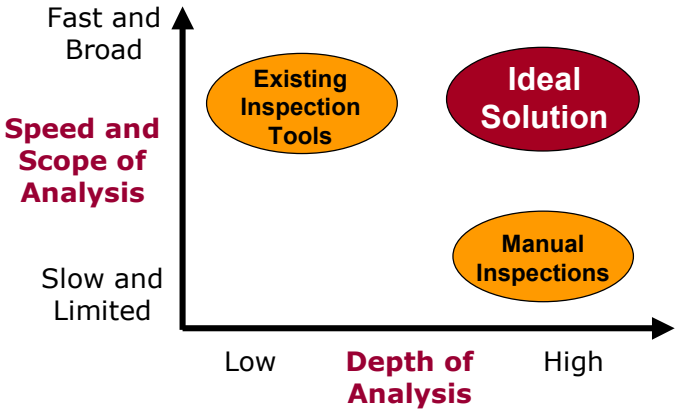


"Inspection is by far the most effective way to remove bugs."

Source: Capers Jones

The Need

A Practical Way to Inspect Software



The Solution

InstantQA:
Automated Software Inspection Service

Finds bugs without testing

- Finds crash-causing bugs
- Works on incomplete code
- Hardware/environment independent
- Does not require test cases

No Development Resources Required

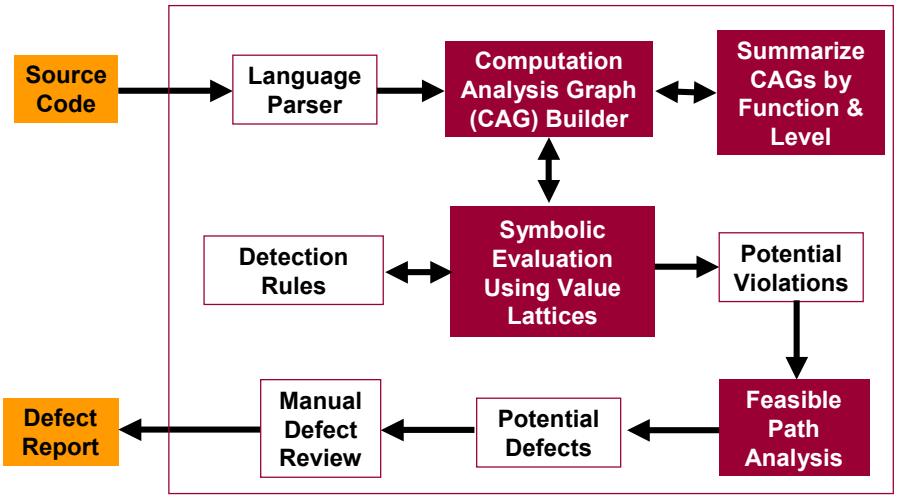
The Technology

Finds Causes of Problems, Not Just the Symptoms

- Based on multiple technical innovations
- Supports C and C++
- Finds five types of defects
  - NULL pointer dereferences
  - Memory leaks
  - Out-of-bounds array access
  - Bad deallocations
  - Uninitialized variables

The Technology

Architecture



 9

= Reasoning Innovations

What's Behind InstantQA?

March 2001 Issue of Dr. Dobb's:

"Value Lattice Static Analysis"

Co-authored by William Brew,
R&D Director, Reasoning

 10

Can You Spot the Bug?

```
void dovec(struct cv *src, struct *cv dst)
{
  char *d, *p;
  ...
  if ((src == NULL || src->nchrs == 0) &&
          dst->nmcces == 0)
      return; /* no src string or no room to copy */
  d = &dst->chrs[dst->nmcce[dst->nmcces - 1]];
  for (p = src->chrs, i = src->nchrs; i > 0; i--)
      *d++ = *p++;
  ...
}
```

TRUE

FALSE

NULL Pointer Dereference

Hint: What happens if *src* == NULL, but *dst->nmcces* != 0?

Sample Defect Report

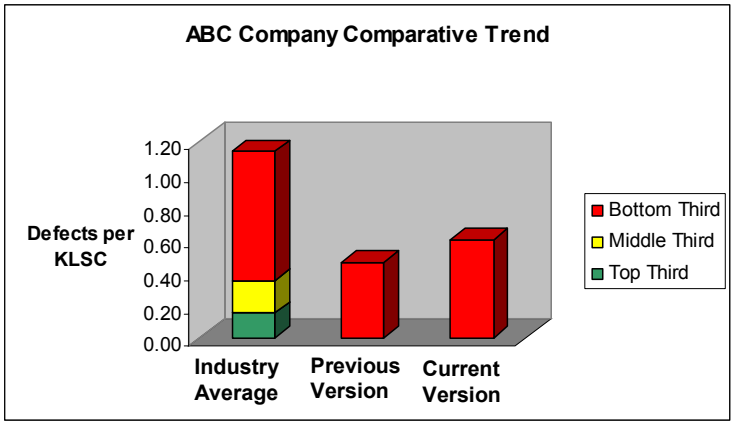
Detailed Defect Report

Defect Class	**Memory Leak**	Risk **Moderate**
Location	**/websrv_1.1/src/os/win32/readdir.c : 43**	
Description	**Pointers to blocks allocated by** malloc() **on lines** 34 **and** 27 **are stored in local variables** dp **and** fspec**. The memory blocks become inaccessible (still allocated, but unreachable) once** dp **and** fspec **go out of scope after line** 43**.**	
Preconditions	**The expression `(errno == ENOENT)` on line** 40 **is false and `((handle = findfirst(fspec, &(dp->fileinfo))) < 0)` on line** 39 **is true**	
Impact	**Memory leaks cause performance degradation of the application, and/or the entire system. Eventually, this may lead to a fatal out-of-memory condition.**	

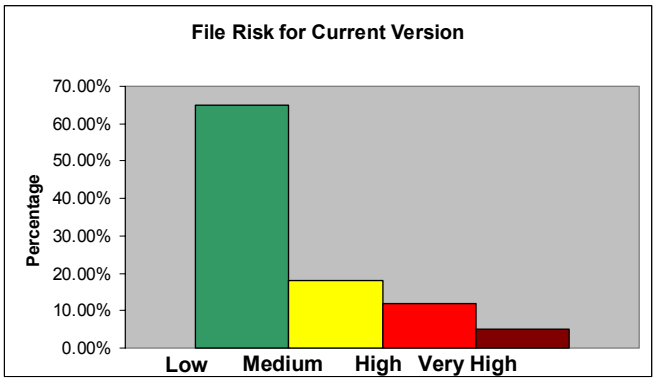
Code Fragment

```
20  API_EXPORT(DIR *) opendir(const char *dir)
21  {
22      DIR *dp;
23      char *fspec;
24      int ix, handle;
26
27      fspec = malloc(strlen(dir) + 2 + 1);
28      strcpy(filespec, dir);
30      if ((ix = strlen(fspec) - 1) >= 0 && (fspec[ix] == '/)
31          fspec[ix] = '\0';
32      strcat(fspec, "/*");
33
34      dp = (DIR *)malloc(sizeof(DIR));
…
39      if ((handle = findfirst(fspec, &(dp->fileinfo))) < 0) {
40          if (errno == ENOENT)
41              dp->finished = 1;
42          else
43              return NULL;
44      }
```

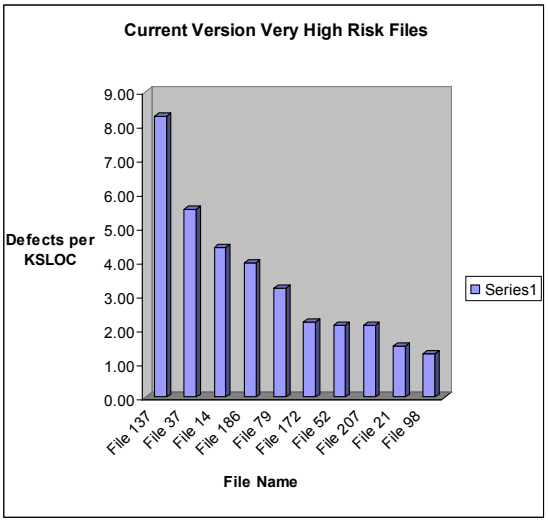
Quality Metrics



Quality Metrics



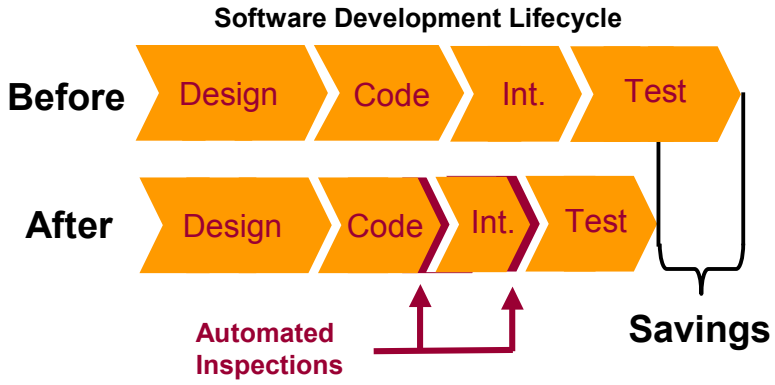
Quality Metrics



 15

The Benefits

Reduced Time and Cost
And Increased Reliability



 16

The Benefits

- Better forecasting
- Tighter scheduling
- Quality tracking
- Highlight training needs

The Proof

Credence Systems – a Significant Savings

Situation
- One million lines of code
- Seven-year-old code base

Benefits
- Eliminated 36 man-months of testing and debugging
- Saved $500K

Results
- Found 331 defects
- 10% were "stop-ship" severity

"It exceeded our expectations by helping our team debug software much faster and at very low cost compared to conventional methods.

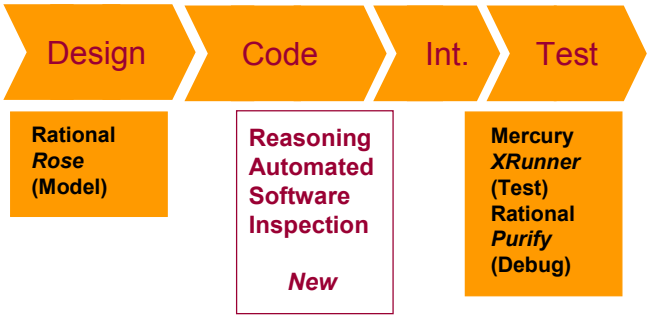
... I highly recommend using the service in any project where quality and highly reliable software are a must."

Jay Bergeron
Director of Product Engineering

InstantQA Completes the Cycle

Complements Design and Testing Tools

Software Development Lifecycle



 19

A Growing Client Base

NEC
HALLIBURTON
Anritsu
ARCHITEL
HITACHI SOFTWARE
hp
JOHNSON CONTROLS
CLEARCOMMERCE
IBM
GENERAL DYNAMICS
ALCATEL
Canon
LARSCOM
credence
VOITH PAPER
VeriFone
MACDONALD DETTWILER
Agilent Technologies
MITEL
OKI
TOSHIBA
NORTEL NETWORKS

 20

QW2001 Paper 7V1

Mr. Rick Banister
(Sesame Technology)

Clothing Optional Relationships With Your Customers--How much should we expose to our customers when it comes to the product improvement process?

Key Points

- Methods for creating limited, direct customer access into your support & quality tracking systems.
- How to turn customer access into big advantage for your product line.
- Learn how far you can go and what methodologies can keep you protected.

Presentation Abstract

No cameras please! This presentation reveals what's underneath the current trend to allow customers limited, direct access into your support and quality tracking systems. What was once considered off-limits is now becoming the norm for progressive product companies and their customer base. Learn how your customers can serve themselves by directly entering and tracking enhancement requests and product defects. Learn what data is best to keep private, and what data keeps customers involved at the appropriate level. See how providing a direct customer interface allows service staff and engineers to work proactively, with fewer interruptions. Lastly, learn how product problems and enhancements can be grouped with other quality issues to create an organized and powerful overview of your product improvement process, ultimately yielding a better product for you and your customers.

About the Author

Rick Banister is the Executive Vice-President and CTO of Sesame Technology. He has developed application systems for over twenty-four years, as a project manager, software developer, systems integrator, database analyst, and business manager. His extensive experience in application development includes Oracle and Java programming, database design, implementation and tuning. Rick is considered by many to be a leading expert in quality and product improvement processes.

QW2001 Paper 7V2

Dr. Edward Miller
(eValid, Inc.)

A Universal Client-Side WebSite Test Engine

Key Points

- Testing WebSites from the Browser has many intrinsic advantages in terms of reality, accessibility, and flexibility.
- The eValid WebSite test engine addresses a wide range of needs for WebSite testing, including functional validation and verification, site timing and page tuning, and complex load imposition.
- This talk describes the architecture, interfaces, and structure of the eValid engine, and provides a number of examples of its operation.

About the Author

Dr. Edward Miller is President of Software Research, Inc., San Francisco, California, where he has been involved with software test tools development and software engineering quality questions. Dr. Miller has worked in the software quality management field for 25 years in a variety of capacities, and has been involved in the development of families of automated software and analysis support tools.

He was chairman of the 1985 1st International Conference on Computer Workstations, and has participated in IEEE conference organizing activities for many years. Miller is the author of *Software Testing and Validation Techniques*, an IEEE Computer Society Press tutorial. Dr. Miller received his Ph.D. (Electrical Engineering) degree from the University of Maryland, an M.S. (Applied Mathematics) degree from the University of Colorado, and a BSEE from Iowa State University.

eValid: The Universal WebSite Test Engine

Dr. Edward Miller

eValid, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA
Email: miller@soft.com

Presentation Outline

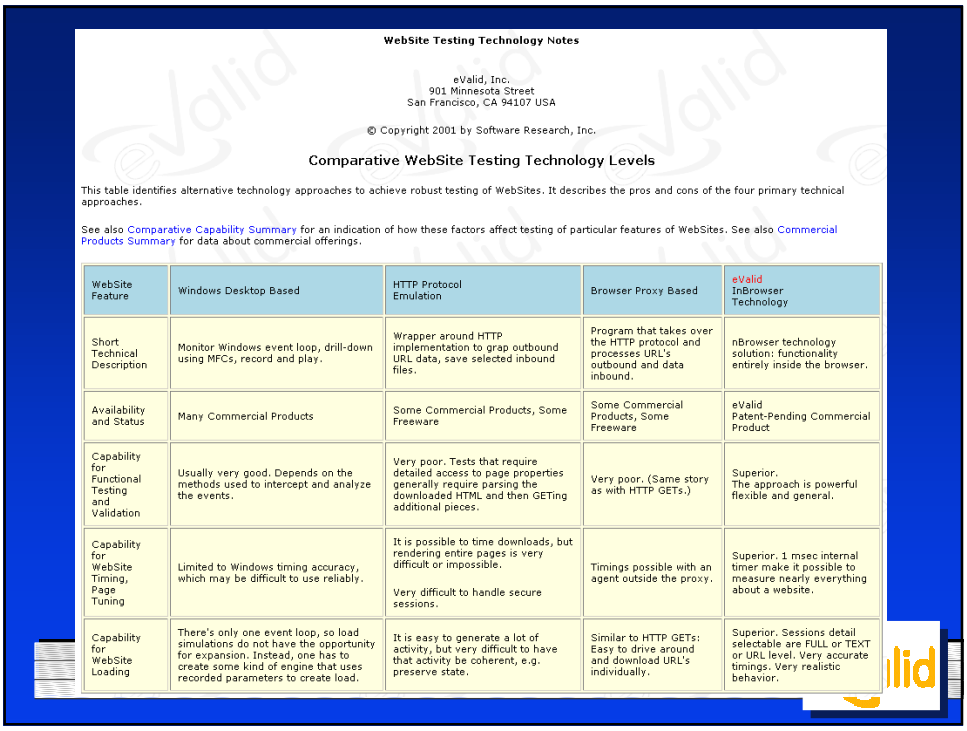
- Overview of Technology
- General Operation Description
- Validation & Verification Modes
- Timing and Tuning
- Load Imposition
- SiteMap Mode
- Performance Considerations

Most Common Problems in WebSites

- Quality/Content
  - Broken Links
  - Missing Components
- Performance
  - Too-Slow Download
- Interaction
  - Failed 1st Layer Transactions
    - Login
    - Specialized Controls
  - Delayed 2nd Tier Transactions
  - Delayed 3rd Tier Transactions

Alternative Technologies

- Windows
  - Client/Server Testing
  - Windows Events
  - Browser is "opaque"
- Unix
  - Client/Server Testing
  - X-Display Events
  - Browser is "opaque"
- Browser
  - Everything is open



WebSite Testing Technology Notes

eValid, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Comparative WebSite Testing Technology Levels

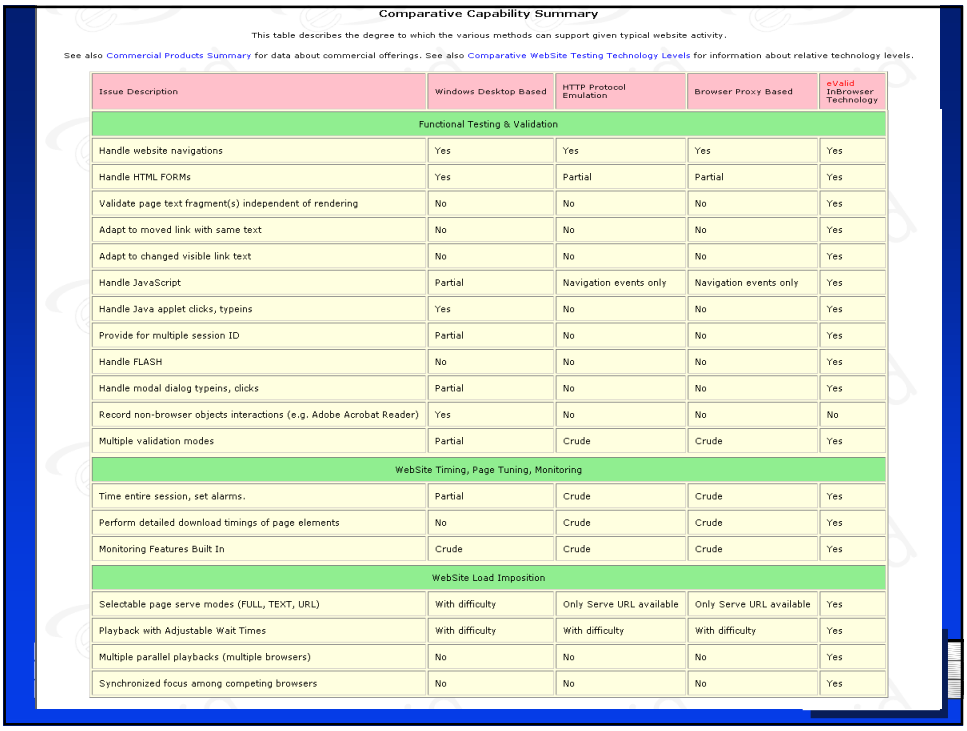
This table identifies alternative technology approaches to achieve robust testing of WebSites. It describes the pros and cons of the four primary technical approaches.

See also Comparative Capability Summary for an indication of how these factors affect testing of particular features of WebSites. See also Commercial Products Summary for data about commercial offerings.

WebSite Feature	Windows Desktop Based	HTTP Protocol Emulation	Browser Proxy Based	eValid InBrowser Technology
Short Technical Description	Monitor Windows event loop, drill-down using MFCs, record and play.	Wrapper around HTTP implementation to grap outbound URL data, save selected inbound files.	Program that takes over the HTTP protocol and processes URL's outbound and data inbound.	nBrowser technology solution: functionality entirely inside the browser.
Availability and Status	Many Commercial Products	Some Commercial Products, Some Freeware	Some Commercial Products, Some Freeware	eValid Patent-Pending Commercial Product
Capability for Functional Testing and Validation	Usually very good. Depends on the methods used to intercept and analyze the events.	Very poor. Tests that require detailed access to page properties generally require parsing the downloaded HTML and then GETing additional pieces.	Very poor. (Same story as with HTTP GETs.)	Superior. The approach is powerful flexible and general.
Capability for WebSite Timing, Page Tuning	Limited to Windows timing accuracy, which may be difficult to use reliably.	It is possible to time downloads, but rendering entire pages is very difficult or impossible. Very difficult to handle secure sessions.	Timings possible with an agent outside the proxy.	Superior. 1 msec internal timer make it possible to measure nearly everything about a website.
Capability for WebSite Loading	There's only one event loop, so load simulations do not have the opportunity for expansion. Instead, one has to create some kind of engine that uses recorded parameters to create load.	It is easy to generate a lot of activity, but very difficult to have that activity be coherent, e.g. preserve state.	Similar to HTTP GETs: Easy to drive around and download URL's individually.	Superior. Sessions detail selectable are FULL or TEXT or URL level. Very accurate timings. Very realistic behavior.

Comparative Technologies

- Windows Desktop
  - Operates from Windows Desktop Event Loop
  - GUI objects partially opaque
- HTTP Protocol
  - Records outbound URLs and response pages
- Browser Proxy
  - Records activity in/out from browser on HTML
- InBrowser Technology
  - Runs inside IE-compatible browser
  - Full context
  - Realistic timings



Comparative Capability Summary

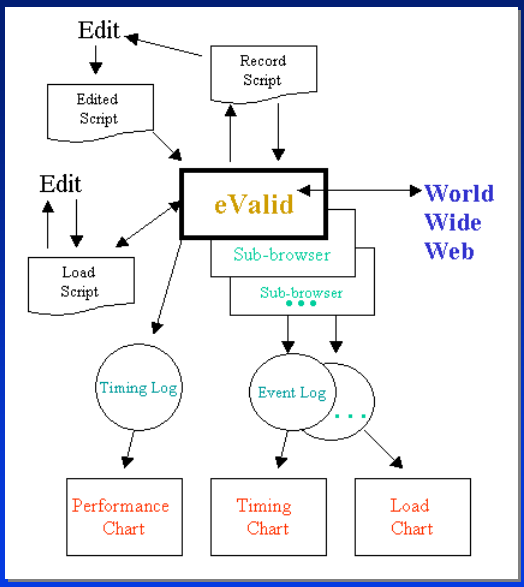
This table describes the degree to which the various methods can support given typical website activity.

See also Commercial Products Summary for data about commercial offerings. See also Comparative WebSite Testing Technology Levels for information about relative technology levels.

Issue Description	Windows Desktop Based	HTTP Protocol Emulation	Browser Proxy Based	eValid InBrowser Technology
Functional Testing & Validation				
Handle website navigations	Yes	Yes	Yes	Yes
Handle HTML FORMs	Yes	Partial	Partial	Yes
Validate page text fragment(s) independent of rendering	No	No	No	Yes
Adapt to moved link with same text	No	No	No	Yes
Adapt to changed visible link text	No	No	No	Yes
Handle JavaScript	Partial	Navigation events only	Navigation events only	Yes
Handle Java applet clicks, typeins	Yes	No	No	Yes
Provide for multiple session ID	Partial	No	No	Yes
Handle FLASH	No	No	No	Yes
Handle modal dialog typeins, clicks	Partial	No	No	Yes
Record non-browser objects interactions (e.g. Adobe Acrobat Reader)	Yes	No	No	No
Multiple validation modes	Partial	Crude	Crude	Yes
WebSite Timing, Page Tuning, Monitoring				
Time entire session, set alarms.	Partial	Crude	Crude	Yes
Perform detailed download timings of page elements	No	Crude	Crude	Yes
Monitoring Features Built In	Crude	Crude	Crude	Yes
WebSite Load Imposition				
Selectable page serve modes (FULL, TEXT, URL)	With difficulty	Only Serve URL available	Only Serve URL available	Yes
Playback with Adjustable Wait Times	With difficulty	With difficulty	With difficulty	Yes
Multiple parallel playbacks (multiple browsers)	No	No	No	Yes
Synchronized focus among competing browsers	No	No	No	Yes

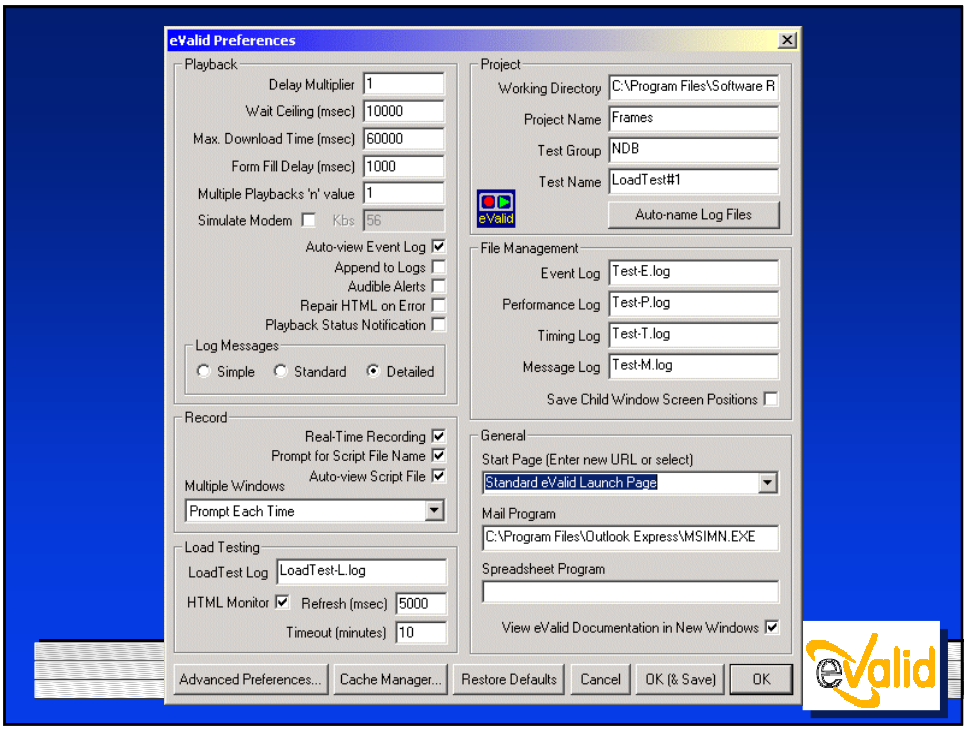
WebBrowser Testing Pros/Cons

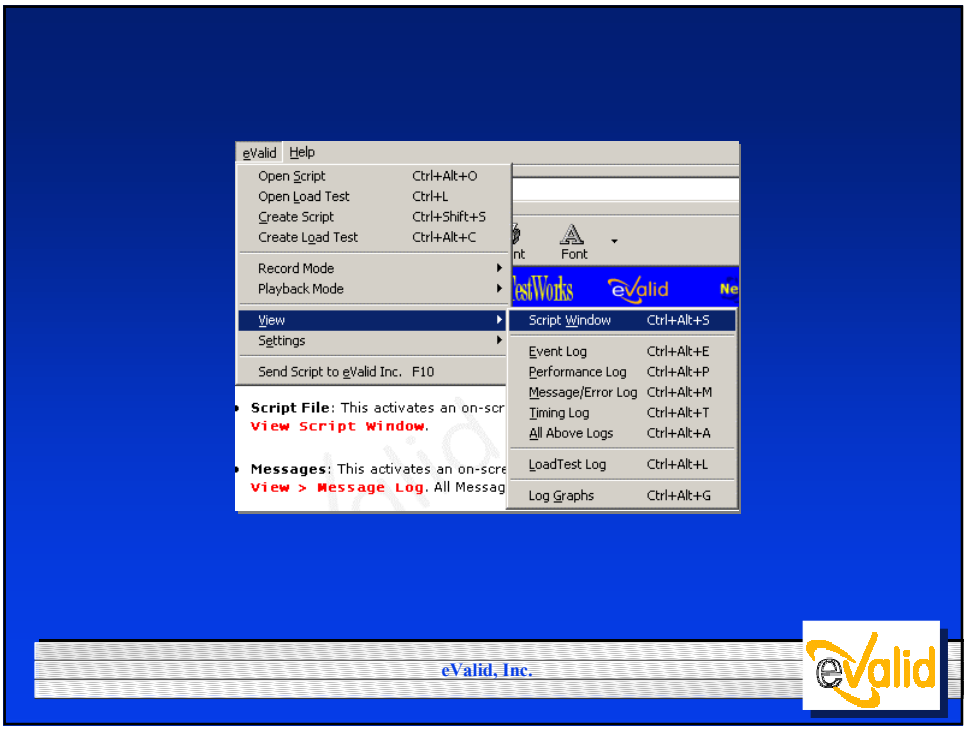
- Pros
  - 100% User View
  - Realistic
  - Natural operation
  - Accurate timings
- Cons
  - Not all browsers are alike
  - UNIX platform support limited

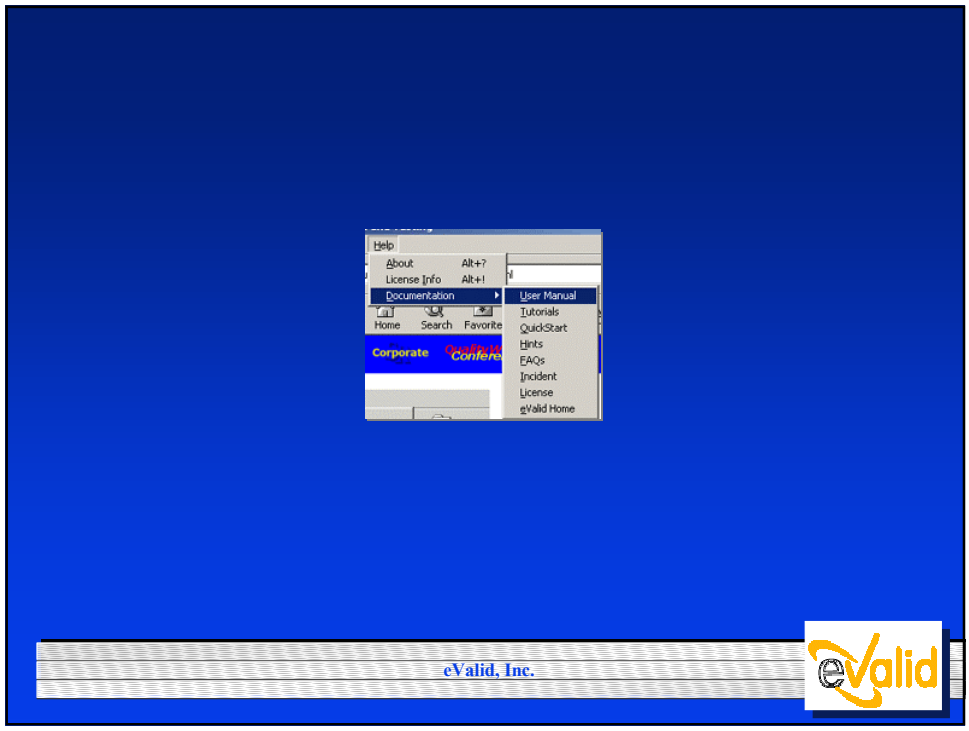


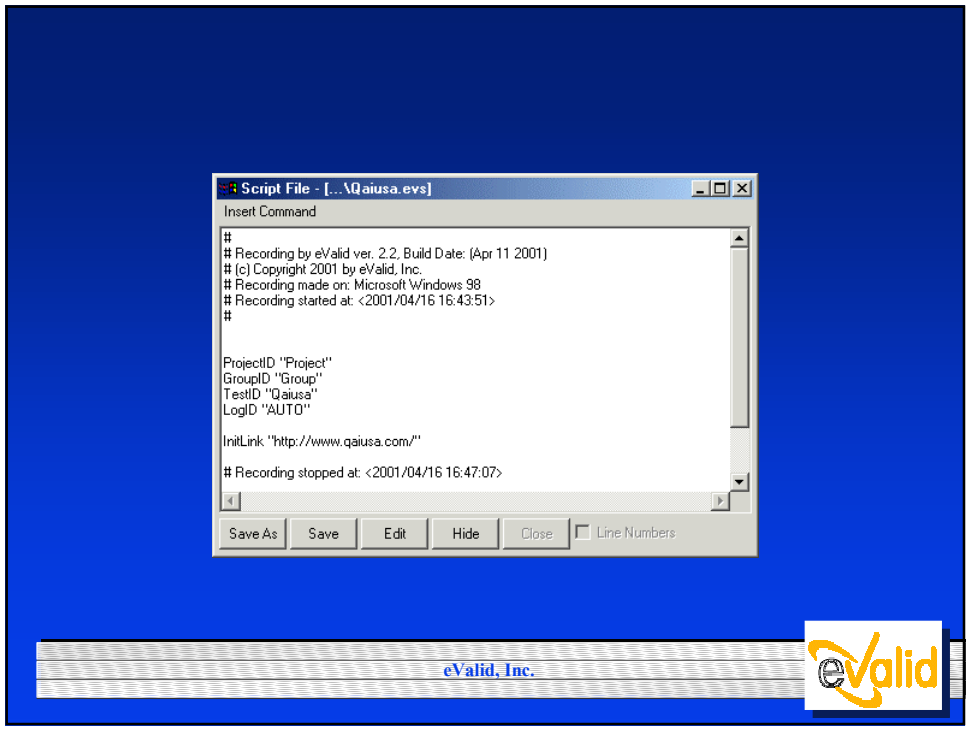
eValid General Features

- IE Base
- Simple Script Language
- Point and Click Interface
- Online Documentation
- Advanced Recording
- Variety of User Options


eValid Preferences
Playback
Delay Multiplier 1
Wait Ceiling (msec) 10000
Max. Download Time (msec) 60000
Form Fill Delay (msec) 1000
Multiple Playbacks 'n' value 1
Simulate Modem Kbs 56
Auto-view Event Log
Append to Logs
Audible Alerts
Repair HTML on Error
Playback Status Notification
Log Messages
Simple Standard Detailed
Record
Real-Time Recording
Prompt for Script File Name
Auto-view Script File
Multiple Windows
Prompt Each Time
Load Testing
LoadTest Log LoadTest-L.log
HTML Monitor Refresh (msec) 5000
Timeout (minutes) 10
Project
Working Directory C:\Program Files\Software R
Project Name Frames
Test Group NDB
Test Name LoadTest#1
Auto-name Log Files
File Management
Event Log Test-E.log
Performance Log Test-P.log
Timing Log Test-T.log
Message Log Test-M.log
Save Child Window Screen Positions
General
Start Page (Enter new URL or select)
Standard eValid Launch Page
Mail Program
C:\Program Files\Outlook Express\MSIMN.EXE
Spreadsheet Program
View eValid Documentation in New Windows
Advanced Preferences...
Cache Manager...
Restore Defaults
Cancel
OK (& Save)
OK
eValid


eValid Help
Open Script Ctrl+Alt+O
Open Load Test Ctrl+L
Create Script Ctrl+Shift+S
Create Load Test Ctrl+Alt+C
Record Mode
Playback Mode
View
Settings
Send Script to eValid Inc. F10
Font
Script Window Ctrl+Alt+S
Event Log Ctrl+Alt+E
Performance Log Ctrl+Alt+P
Message/Error Log Ctrl+Alt+M
Timing Log Ctrl+Alt+T
All Above Logs Ctrl+Alt+A
LoadTest Log Ctrl+Alt+L
Log Graphs Ctrl+Alt+G
Script File: This activates an on-scr
View Script Window.
Messages: This activates an on-scre
View > Message Log. All Messag
eValid, Inc.
eValid


Help
About Alt+?
License Info Alt+I
Documentation
User Manual
Tutorials
QuickStart
Hints
FAQs
Incident
License
eValid Home
Home
Search
Favorite
Corporate
eValid, Inc.

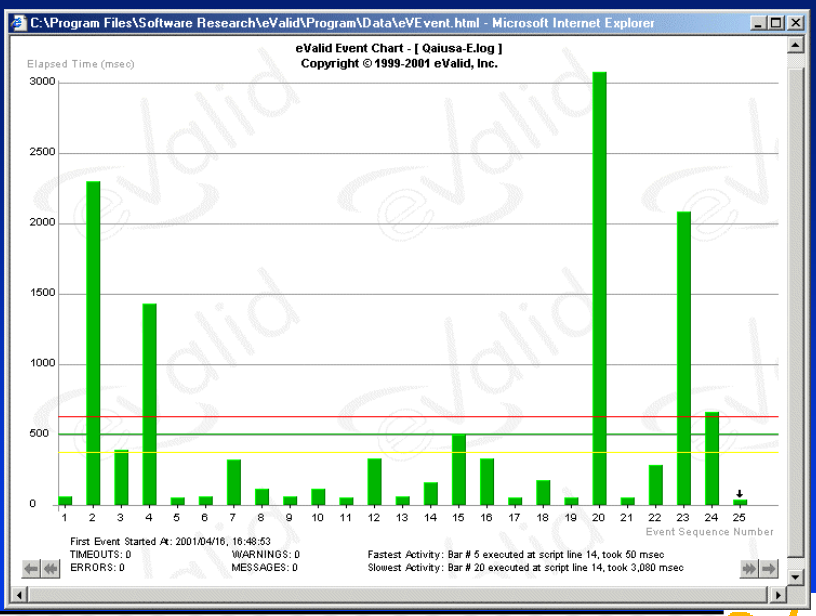

Script File - [...\Qaiusa.evs]
Insert Command
#
Recording by eValid ver. 2.2, Build Date: (Apr 11 2001)
(c) Copyright 2001 by eValid, Inc.
Recording made on: Microsoft Windows 98
Recording started at: <2001/04/16 16:43:51>
#
ProjectID "Project"
GroupID "Group"
TestID "Qaiusa"
LogID "AUTO"
InitLink "http://www.qaiusa.com/"
Recording stopped at: <2001/04/16 16:47:07>
Save As
Save
Edit
Hide
Close
Line Numbers
eValid, Inc.

Dynamic Analysis/Testing Validation

- Text
- Images
- Dialogs
- Sequences
- Other

Dynamic Performance Timing

- Single and Multiple Download Timings
- Overall User-level Response Times
- Perceived User-level Response Times, Thresholds
- Web Effects

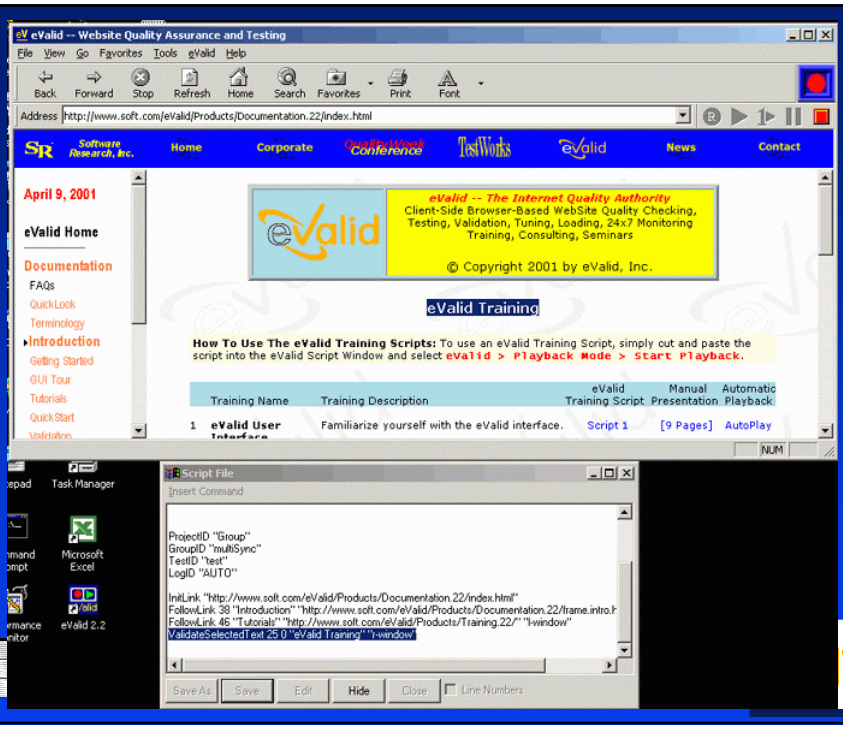


Hurdles for Dynamic Testing

- Repeatability
- Databases that "Remember"
- Fancy Page Effects
- Multi-Media Displays
- Browser Differences
  - Rendering
  - Adaptive Servers

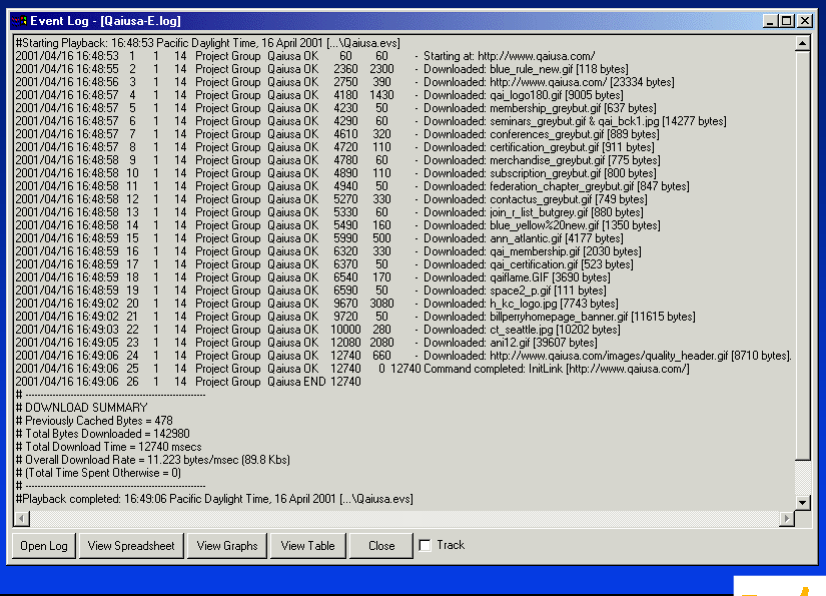
Functional Verification & Validation

- Document
  - URL
  - Elements
  - Size
  - Last-Modified Date
- User-Selected
  - Text
  - Image
  - Table-Cell
- All
  - Images
  - Applets
  - Element IDs



Client-Site Timing

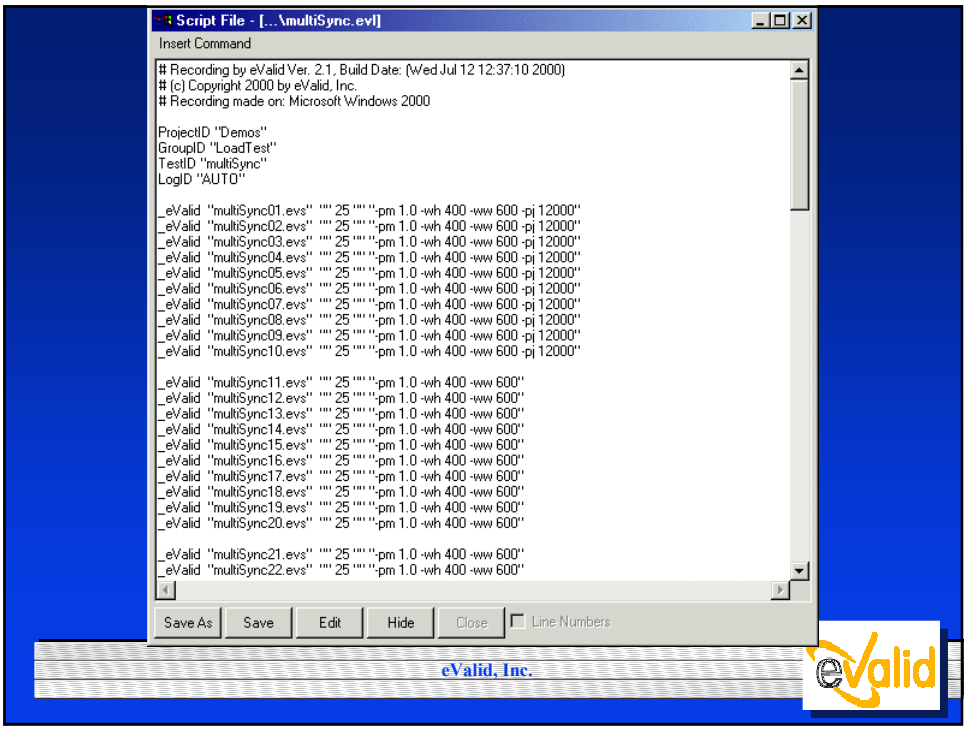
- Overall Timer
  - Total Time
  - Total Bytecounts
- Page Timing
  - Base Page
  - LINKed Files
    - JavaScript (*js)
    - Cascading Style Sheets (*css)
  - Images



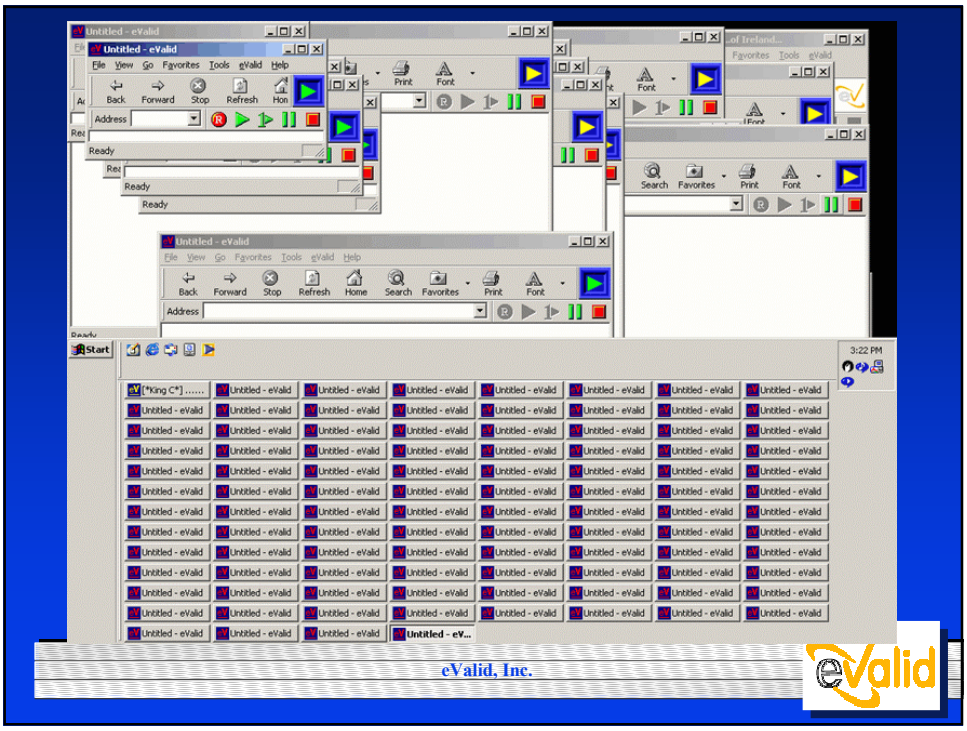
eValid, Inc.

Load and Capacity Checking/Testing

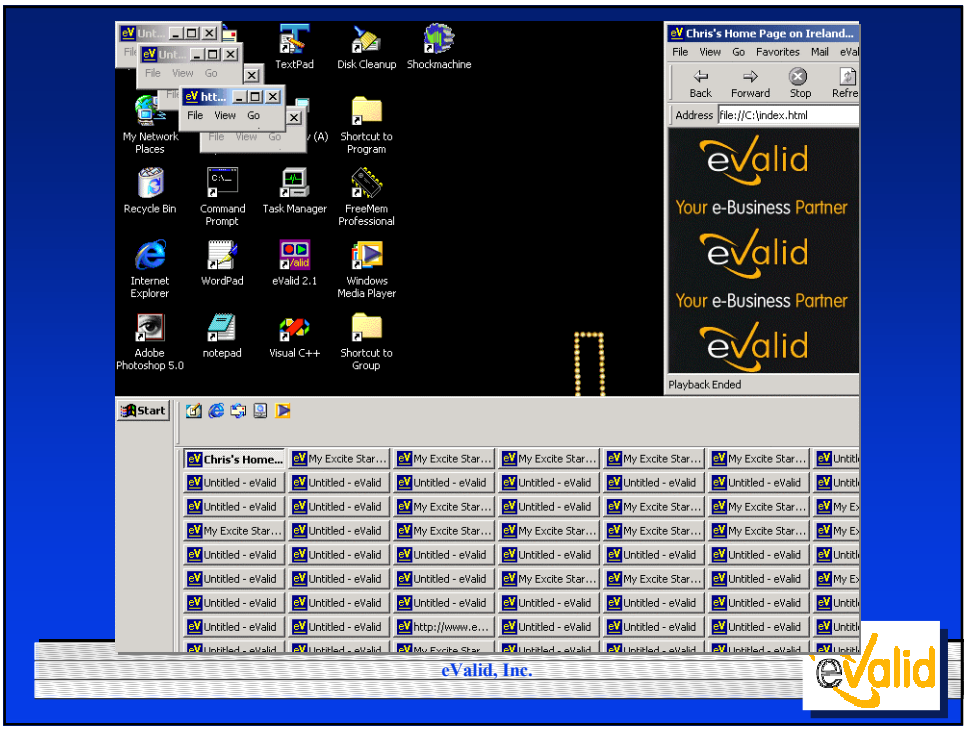
- Load Imposition
- Load Measurement
- User Scenarios
- Realistic vs. Non-Realistic Experiments
- Client-Based vs. Server-Based Experiments
- Web Variability

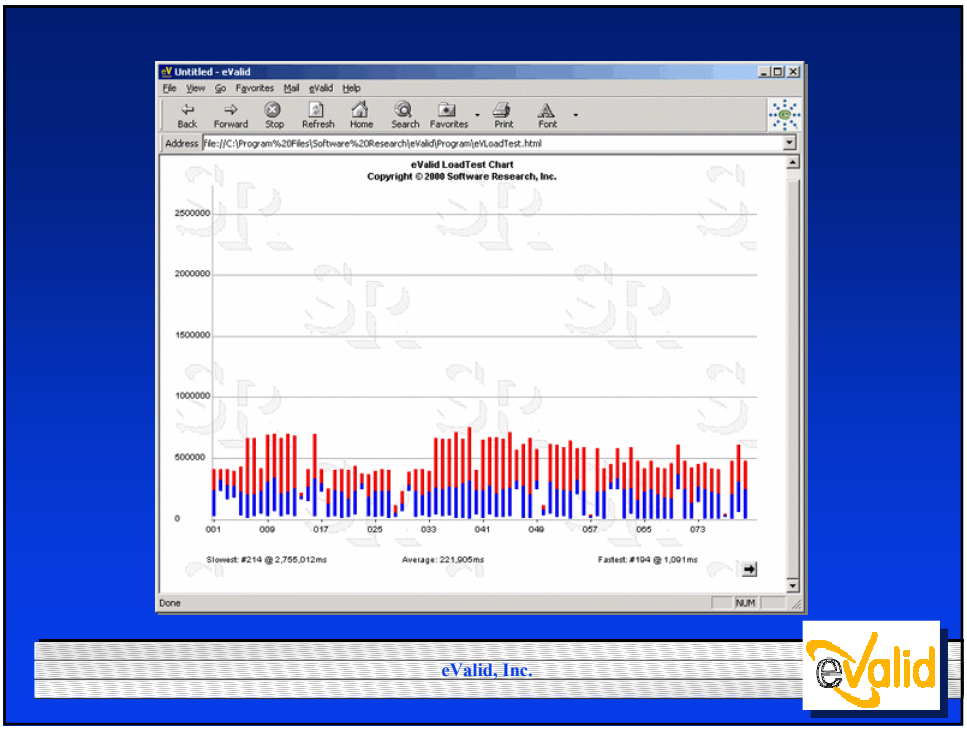


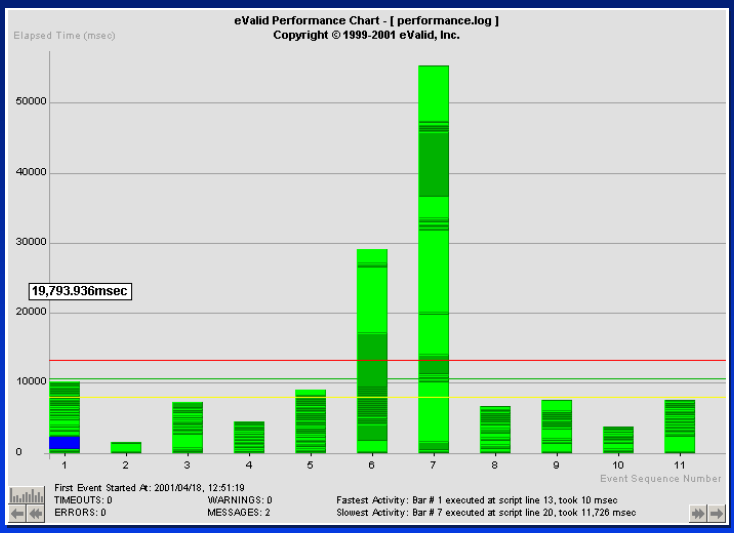
eValid, Inc.


eValid, Inc.


eValid -- Website Quality Assurance and Testing
File View Go Favorites Tools eValid Help
Back Forward Stop Refresh Home Search Favorites Print Font
Address http://www.soft.com/eValid/Products/Documentation.22/index.html
Software Research, Inc.
Home
Corporate
QualityWeek Conference
TestWorks
eValid
News
Contact
April 18, 2001
eValid Home
Documentation
FAQs
QuickLook
Terminology
Latest Version
Introduction
Helpful Hints
LoadTest
Wizards
Advanced
Technical
Feedback
Other
eValid -- The Internet Quality Authority
Client-Side Browser-Based WebSite Quality Checking, Testing, Validation, Tuning, Loading, 24x7 Monitoring Training, Consulting, Seminars
© Copyright 2001 by eValid, Inc.
eValid Ver. 2.2
NOTE: The on-line documentation for eValid is in continuous update status. To avoid confusion when the current version changes the directory that holds the documentation for that version will also change. It is best to use the eValid documentation online and not to print it out as it will be changed without notice.
Release Notes
Please check in the Release Notes for information about features, capabilities, and limitations of this version of eValid.
Current Version
If the version you are using is lower than the one identified above, or if the release date on the version you are using is earlier than the one given, you should download the latest version of
Done
NUM


TextPad
Disk Cleanup
Shockmachine
My Network Places
Shortcut to Program
Recycle Bin
Command Prompt
Task Manager
FreeMem Professional
Internet Explorer
WordPad
eValid 2.1
Windows Media Player
Adobe Photoshop 5.0
notepad
Visual C++
Shortcut to Group
Chris's Home Page on Ireland...
Address file://C:\index.html
Your e-Business Partner
Your e-Business Partner
Playback Ended
Start
eValid, Inc.


Untitled - eValid
Address file://C:\Program%20Files\Software%20Research\eValid\Program\eVLoadTest.html
eValid LoadTest Chart
Copyright © 2000 Software Research, Inc.
2500000
2000000
1500000
1000000
500000
0
001
009
017
025
033
041
049
057
065
073
Slowest: #214 @ 2,755,012ms
Average: 221,905ms
Fastest: #194 @ 1,091ms
Done
NUM
eValid, Inc.



WebSite Testing Technology Notes

eValid, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Comparative Products Summary

This table identifies alternative technology approaches to achieve robust testing of WebSites. It describes the pros and cons of the four primary technical approaches.

See also Comparative Capability Summary for an indication of how these factors affect testing of particular features of WebSites.
See also Comparative WebSite Testing Technology Levels for information about relative technology levels.

WebSite Feature	Windows Desktop Based	HTTP Protocol Emulation	Browser Proxy Based	eValid InBrowser Technology
Example Functional Test Products	SR: CAPBAK/MSW MI: WinRunner Rational: QARobot Segue: SilkTest Compuware: QARun			eValid
Example Timing/Tuning Test Products				eValid
Example Load Test Products	MI: LoadRunner Segue: SilkPerformer Compuware QALoad	Radview: SiteLoad Empirix: eTester MI: Astra	NaviScope Keynote Recorder Atesto ProactiveNet Recorder	eValid

Typical Monitoring Services

- 24x7
- Geographic Coverage
- What is monitored
- Response method:
  - Email
  - Page
  - Direct

QW2001 Paper 8V1

Mr. Steve Smith
(QualityLogic)

Develop Great Stuff, Repeatedly, On Time

Key Points

- Delivering quality products repeatedly requires more than good process. It requires effective strategy and an enabling culture. All three must be present and in balance for success.
- A nine-step formula for delivering good stuff on time…repeatedly.
- Successful quality management the easy way.

Presentation Abstract

Success in building quality products depends on how well a company's management does at setting and enforcing effective corporate Strategy, Process and Culture. Without conscious attention to these factors, some level of quality just "happens" or more likely, "doesn't happen."

No matter how good the processes and tools, if a team does not have a clearly communicated strategy and a disciplined culture to implement that strategy, the probability of delivering good stuff, on time, repeatedly is very small.

QualityLogic has developed the "Quality Success Formula" to work with companies in successfully delivering quality software products that meet and exceed your customers' expectations. The "Quality Success Formula" is a blend of industry best practices and knowledge gained from QualityLogic's years of experience in the software quality business. This includes the management of strategy, process and culture; Information Quality Assurance; advanced quality understanding and philosophy; and product life cycle management.

About the Author

Steve Smith has over 30 years experience in software development and Quality Assurance with companies such as IBM and Mentor Graphics. Steve joined QualityLogic in April of 2000 and has been a key contributor to building the QualityLogic Professional Services Practice and the QualityLogic Quality Assessment Process.

He started his career at IBM. After a tour in the Army he returned to IBM working in a Quality Engineering group testing IBM's then fastest computer, the Model 195. His next position was as Manager of Mission Test for the Physical Design Mission in IBM's Engineering Design System (EDS) Physical Design Mission

developing application software for IBM’s 26 worldwide labs. In this role he was responsible for directing the Quality Assurance and Quality Control requirements of the mission.

Steve moved to Mentor Graphics in the early 1980s as the Manager of Systems Engineering Support in the Board Systems Division. He helped lead Mentor and his division to ISO9001 Certification and conducted SEI assessments. Steve then took on the role of Director of Usability for the PCB Product and then became Director of Engineering for the Division that was approaching $100 million in size.

QualityLogic

QualityLogic Inc.

**"Deliver Good Stuff,
On Time…Repeatedly"**

Quality Week 2001

QualityLogic

QualityLogic

Quality Is Key

QualityLogic

QualityLogic

Quality Doesn't Just Happen

- "Investment in process improvement pays off. Costs are paid back by better quality and shorter development time…in addition to avoiding dissatisfied customers, companies can save time and effort by adopting quality-assurance processes."

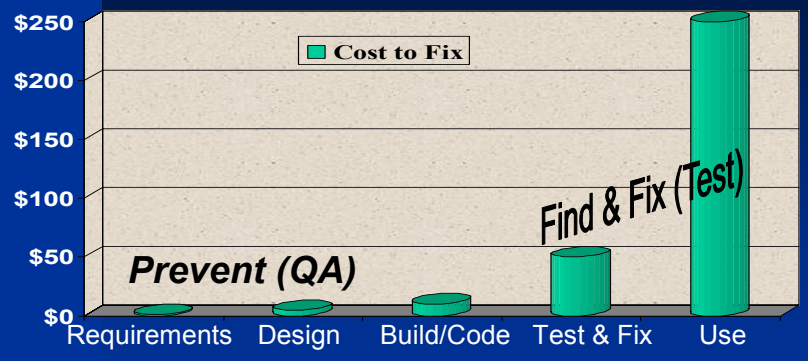
Secrets of Software Success
McKinsey & Company Inc.
Harvard Business School Press, 2000

QualityLogic

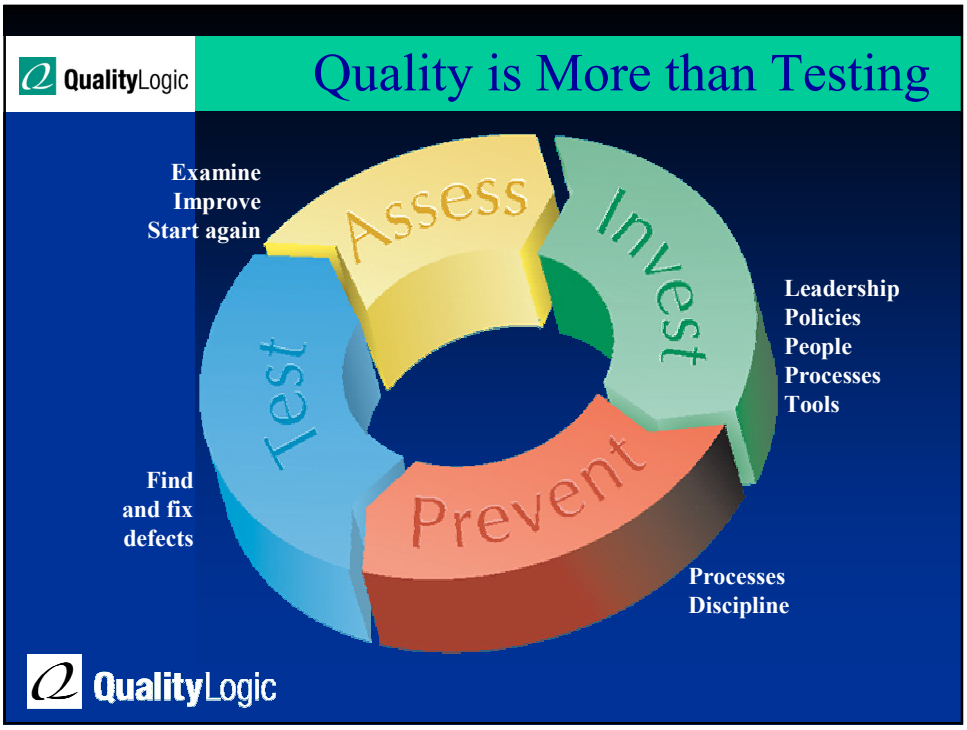
QualityLogic

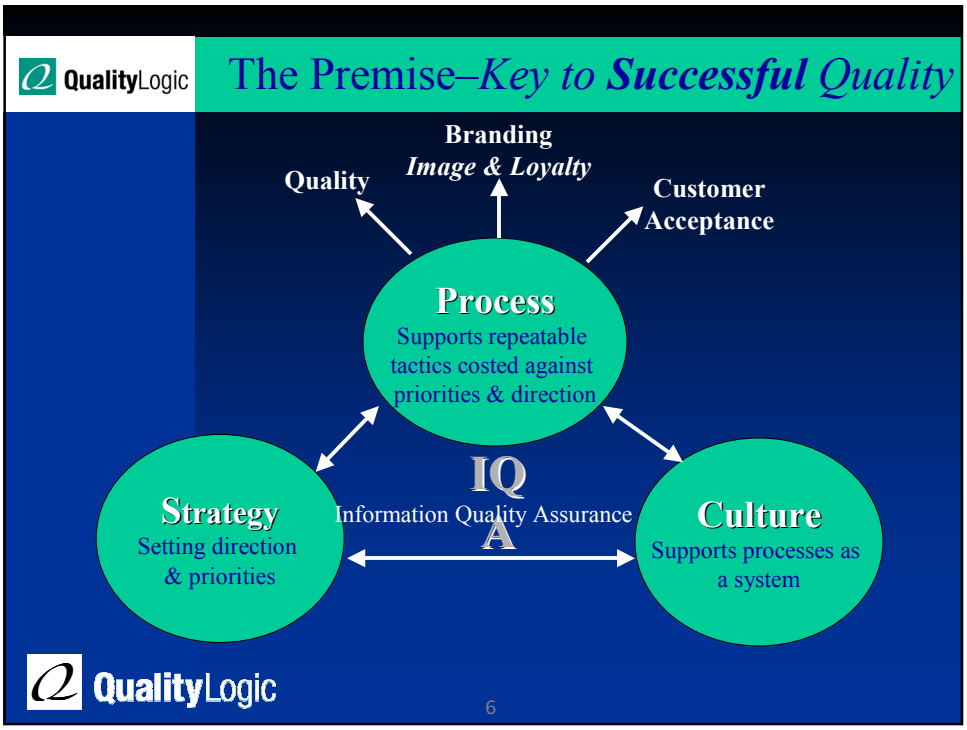
It Pays to Invest Early

- The cost of finding defects too late



QualityLogic


QualityLogic
Quality is More than Testing
Examine
Improve
Start again
Assess
Invest
Test
Prevent
Leadership
Policies
People
Processes
Tools
Find
and fix
defects
Processes
Discipline
QualityLogic


QualityLogic
The Premise–*Key to* ***Successful*** *Quality*
Branding
Image & Loyalty
Quality
Customer
Acceptance
Process
Supports repeatable
tactics costed against
priorities & direction
IQ
Information Quality Assurance
A
Strategy
Setting direction
& priorities
Culture
Supports processes as
a system
QualityLogic

QualityLogic

Quality Definitions

What is quality in the new economy?

"It is about both what you **do** and what you **deliver**. Means and ends. Brilliant leadership, effective strategy, ***unrelenting attention to customers and markets*****,** the recruitment, training and retention of exemplary staff, process superiority, good use of data, and great results."

Business Week April 17, 2000

QualityLogic

QualityLogic

Quality Definitions

Quality is the delivery of expected:

- Functionality
- Usability
- Reliability
- Performance
- Supportability

on time…repeatedly

This creates and sustains brand image and loy

QualityLogic

QualityLogic

Quality Definitions

Quality Assurance vs. Quality Control

Quality Assurance	Quality Control
Strategic	Tactical
Prevention	Detection
Business/Risk Management	Minimize Rework
Process Analysis/Creation	Product & Process Validation
Continuous Improvement	Testing

QualityLogic

QualityLogic

Your Software Quality Resource

- 200+ employees
- Internally funded
- Profitable
- Nationwide
  - Los Angeles, CA
  - Portland, OR
  - Seattle, WA
  - Boise, ID
  - Lexington, KY
  - Sacramento, CA
  - Cupertino, CA

QualityLogic

QualityLogic

Some QualityLogic Clients

QualityLogic

QualityLogic

We Deliver…

- Assurance that product quality issues don't keep you up at night!
- By Solving quality problems at the
  - Enterprise level
  - Department level
  - Project level
- Flexible solutions, tailored for you

QualityLogic

QualityLogic

Summary

- Quality in software matters
- QualityLogic is the partner of choice
- "In rapidly growing a company, you need to outsource everything you can to reliable vendors. QualityLogic leads in the software quality area."

Justin Segal, Co-Founder
Startups.com

QualityLogic

QW2001 Paper 8V2

Mr. Shel Siegel
(Tescom)

Component Based Architecture for Automated Testing

Key Points

- An automation architecture is necessary to optimize productivity and value.
- A component based automation architecture has inherent advantages.
- Re-use of test artifacts may be optimized by using a component based automation architecture.

Presentation Abstract

As test automation tools flourish it is easy to be lulled into thinking that automation will increase productivity or add value to the testing process. In this regard an old saying applies “if you give a fool a tool, the fool will be able to do what he/she does not know how to do ….. faster”. In other words, a fool with a tool is still a fool!

Recently I asked an assembly of more than 350 QA and test managers how many had designed or implemented an architecture for the automated test scripts they were producing. Less than 5% had architected their automation projects. Shocking wouldn’t you say?

If we in the test and software quality fields believe that it is desirable for our developer counterparts to architect and design their software projects why did only 5% of the managers of testing departments architect and design their automation testing projects?

This talk presents a simple and powerful automation architecture for regression test scripts using tools like WinRunner or Silk.

About the Author

I co-founded The Alliance for Software Quality (1991) and co-developed the Quality Optimization (QO) software quality deployment methodology (1993) and the Hierarchical Testing Approach. (1996). I worked with the American Electronics Association (AEA) National TQC Steering Committee to provide “leadership for quality” in the software industry. (1994) I was the technical editor for Software Quality World and I am known as a dynamic presenter at conferences on the subjects of project management, quality engineering, and software testing techniques. I chaired the committee that designed IBM’s System Evaluation

Laboratory at Santa Teresa Labs (1983) and I have built and directed quality departments for large and small software companies since 1981.

COMPONENT BASED ARCHITECTURE for AUTOMATED TESTING

Created by Shel Siegel

Version 2.0

 1

AUTOMATION ADVANTAGAES

- **Practical test environment that is easy to learn and use**
- **Facilitate the reuse of test objects.**
- **Enables greater productivity through the reuse of test objects.**
- **Frees testing resources to do analysis and design**

 2

TEST AUTOMATION VISION

Time spent testing products will be significantly reduced by automating the time consuming manual test procedures.

- Defects will be found faster/earlier after a Dev build.
- Fixes may be tested/re-tested quickly.
- Free-up test people to perform more valuable activities.
- Time to market MAY BE shortened?

 .3

TEST AUTOMATION MISSION

Automate regression testing for all project builds.

- For each milestone Dev. build - execute automated regression test.
- Verify a minimum acceptable level of functionality.
- Focus on high risk areas of the product.

 .4

TEST AUTOMATION OBJECTIVES

1. Create a flexible automation infrastructure with a level of test abstraction.
2. Create an architecture to be used by other projects.
3. Define naming conventions & standards that enable test object identification from the name.
4. Design test objects that can be managed by an inventory tool or version control tool.
5. Create modular & reusable tests.
6. Use existing product test plans as specifications for automated regression tests.
7. Use WinRunner for GUI based automation.

 .5

OBJECTIVE - 1

Objective

Create a flexible automation infrastructure with a level of test abstraction.

Approach

Use a component based architecture that views test artifacts as reusable components.

- **Amount of reuse of test artifacts.**
- **Ability to mix and match WinRunner test steps.**

 .6

OBJECTIVE - 2

Objective

Create an architecture to be used by other projects.

Approach

Design the test infrastructure to be project independent.

- **How easy is it to convert other projects into the automation architecture.**
- **Ratio of reused test objects to number used by this project.**

 .7

OBJECTIVE - 3

Objective

Define naming conventions & standards that enable test object identification from the name.

Approach

Use a simple byte-position coding convention to map key pieces of information about the test object into the name.

- **Number of project test components that meet agreed upon standards.**
- **Ratio of compliant test objects to total objects being produced.**

 .8

OBJECTIVE - 4

Objective

Design test objects that can be managed by an inventory tool or version control tool.

Approach

Encode the type of test object, product/project and a unique identifier into the name of the test object.

Use a standard parsable test header.

- **The names are unique and meet the stated name conventions.**
- **Inspect the test objects for compliance.**

 .9

OBJECTIVE - 5

Objective

Create modular & reusable tests.

Approach

Use a bottom up approach to designing tests.

Create single step tests that correspond to individual user fields on a specific GUI screen.

Document very well.

- **Number of times a test is reused.**

 .10

OBJECTIVE - 6

Objective

Use existing project test plans as specifications for automated regression tests.

Approach

Convert the Test Plans (really Test Specifications!) to the WinRunner Test Header format.

- **Number of conversions per person hour.**
- **Ratio of conversions to total number.**

OBJECTIVE - 7

Objective

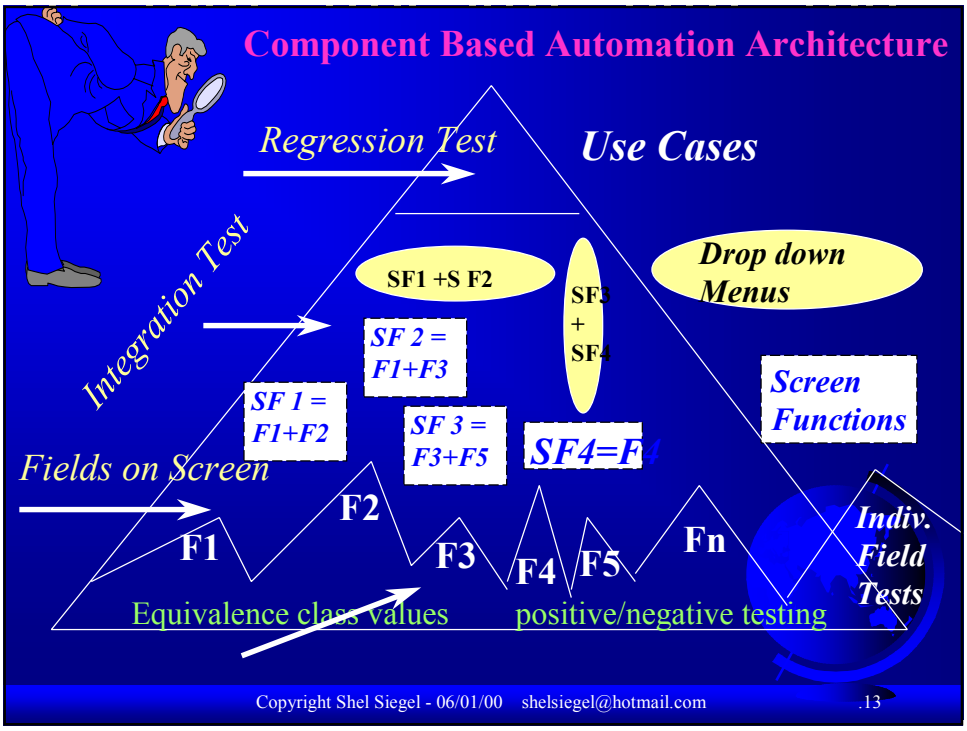
Use WinRunner for GUI based automation.

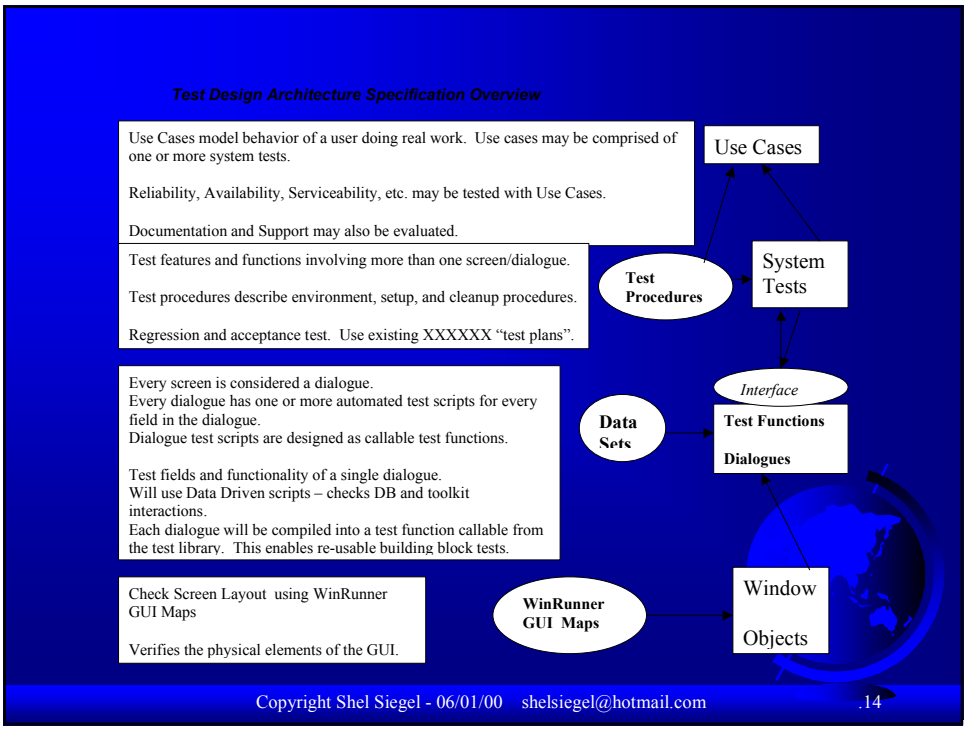
Approach

Provide WinRunner test case templates to the automation team.

Create simple basic WinRunner test cases for each field on a screen.

- **Number of different types of templates needed.**
- **Time to execute individual WinRunner test steps.**


Component Based Automation Architecture
Regression Test
Use Cases
Integration Test
SF1 +S F2
SF3
+
SF4
Drop down
Menus
SF 2 =
F1+F3
SF 1 =
F1+F2
SF 3 =
F3+F5
SF4=F4
Screen
Functions
Fields on Screen
F1
F2
F3
F4
F5
Fn
Indiv.
Field
Tests
Equivalence class values
positive/negative testing
Copyright Shel Siegel - 06/01/00 shelsiegel@hotmail.com .13

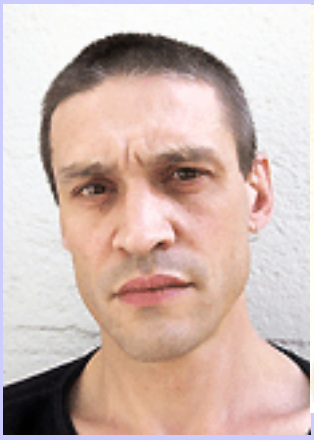

Test Design Architecture Specification Overview
Use Cases model behavior of a user doing real work. Use cases may be comprised of one or more system tests.
Reliability, Availability, Serviceability, etc. may be tested with Use Cases.
Documentation and Support may also be evaluated.
Use Cases
Test features and functions involving more than one screen/dialogue.
Test procedures describe environment, setup, and cleanup procedures.
Regression and acceptance test. Use existing XXXXXX "test plans".
Test Procedures
System Tests
Every screen is considered a dialogue.
Every dialogue has one or more automated test scripts for every field in the dialogue.
Dialogue test scripts are designed as callable test functions.
Test fields and functionality of a single dialogue.
Will use Data Driven scripts – checks DB and toolkit interactions.
Each dialogue will be compiled into a test function callable from the test library. This enables re-usable building block tests.
Data Sets
Interface
Test Functions
Dialogues
Check Screen Layout using WinRunner GUI Maps
Verifies the physical elements of the GUI.
WinRunner GUI Maps
Window Objects
Copyright Shel Siegel - 06/01/00 shelsiegel@hotmail.com .14

Re-Usable Test Components

- Individual Field Tests (Unit)
- Screen Tests (Functional)
- Menu Tests (Integration)
- Regression Test
- Use Cases
- Create Test Data files for each field
- Use Combinations of Field Tests
- Use Combinations of Screen Tests
- Variations & Combinations of Menu Tests
- Different Variations of Menu Test Sequences

AUTOMATION CRITICAL SUCCESS FACTORS

- **Existing manual test bed**
  - **Use existing test specifications.**
- **Re-usable automation architecture**
  - **Testers trained in:**
  - **Consistent conceptual framework.**
  - **Component based architecture.**
- **Automation tool expertise.**
  - **Requires individuals trained on the tool.**
- Project manager uses well defined Rolling Wave SDLC and provides predictable documented builds.



QW2001 Paper 2T1

Mr. Tobias Mayer, Mr. Thomas Stocking
(eValid, Inc., siteROCK)

The Web Site Testing Challenge

Presentation Overview

Web sites are becoming increasingly more complex due to:

1. The inclusion of (e.g.) Flash objects, Java Applets, XML, Javascript.
2. The increased use of Multiple Windows, Secure Log-Ins, Message Pop-ups and Web-launched applications.

Testing these sites requires tools that can intuitively and accurately adapt to such complexity.

The first part of this talk will address many of the difficulties in testing a modern web site from the perspective of a professional web site quality tester. The second part will discuss the 'Browser-centric' test tool as a solution to some of these difficulties.

The main focus will be on the qualities that a test tool needs to meet the myriad of requirements that the web site tester is faced with.

About the Authors

Tobias Mayer is a senior software engineer at Software Research, Inc. He is reponsible for the main design and implementation of the "eValid" Web Test engine. Tobias has a (UK) BSc from South Bank University, London. He is a member of, and OO Metrics consultant to, the Center for Systems & Software Engineering (CSSE) at South Bank University. Tobias has presented and published a number of papers on OO metrics, including papers at IEEE 'TOOLS' 1999 and British Computer Society 'SQM' 1999. During 2000, Tobias presented a number of seminars on Website Testing strategies in the UK. He also presented the "Quickstart - Website Testing" seminar at the 'Quality Week 2000' conference in San Francisco, June 2000.

Thomas Stocking is a systems engineer for SiteROCK Corporation. In this position he is responsible for the implementation of monitoring systems for siteROCK’s enterprise customers. He has had the privilege of working with some of Silicon Valley’s best and brightest in this role over the past two years. Thomas has over 10 years experience as a systems integration consultant, and has worked

on various IT implementation teams. He has a degree in Applied Mathematics from the University of California, and is a native of the San Francisco Bay Area.

The Web Site Testing Challenge

Thomas Stocking
SiteROCK
&
Tobias Mayer
eValid Inc.

Quality Week 2001

The Web Site Testing Challenge

- **Part 1 - Using Web Site Testing Tools**
  - Presented by Thomas Stocking (SiteROCK)
- **Part 2 - The Browser-centric Approach**
  - Presented by Tobias Mayer (eValid, Inc.)

Quality Week 2001

Outline

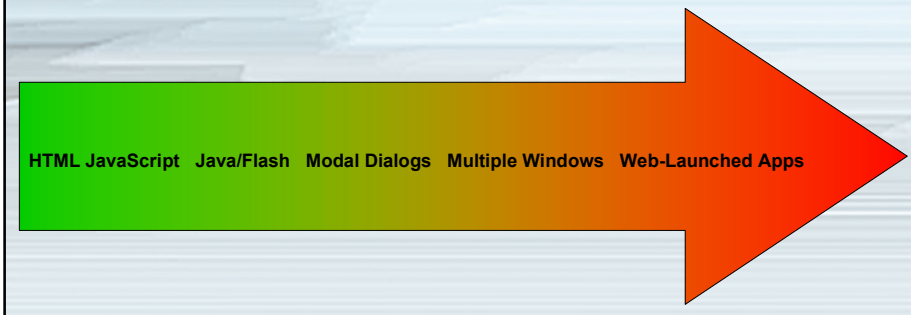
- How siteROCK Tests Sites
- Page Types and Challenges
- Tool Features
- Limits of the Technology
- Examples
- Q&A

How siteROCK Tests Sites

- Availability and Performance Measurement Service
- Monitoring the End User Experience
- Diverse Technologies Tested
- Programmatic Approach – We need good tools!

Page Types and Specific Challenges

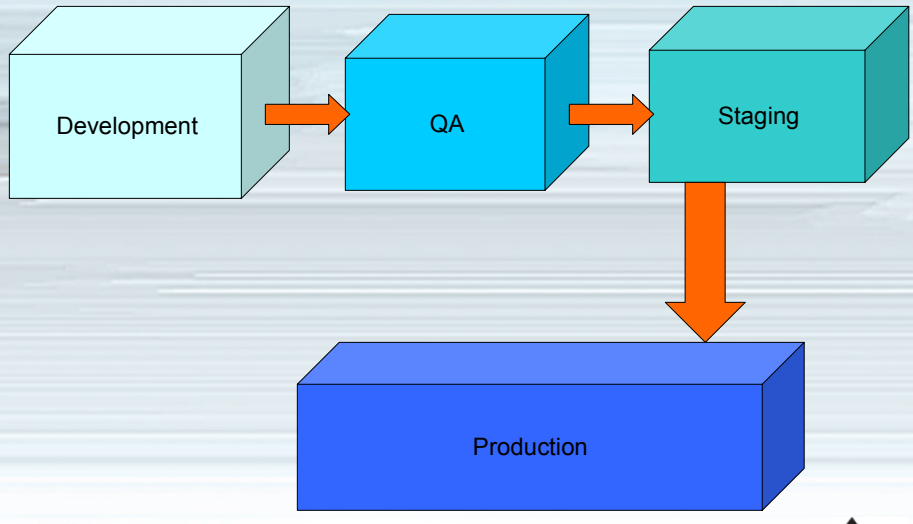
Easy Hard



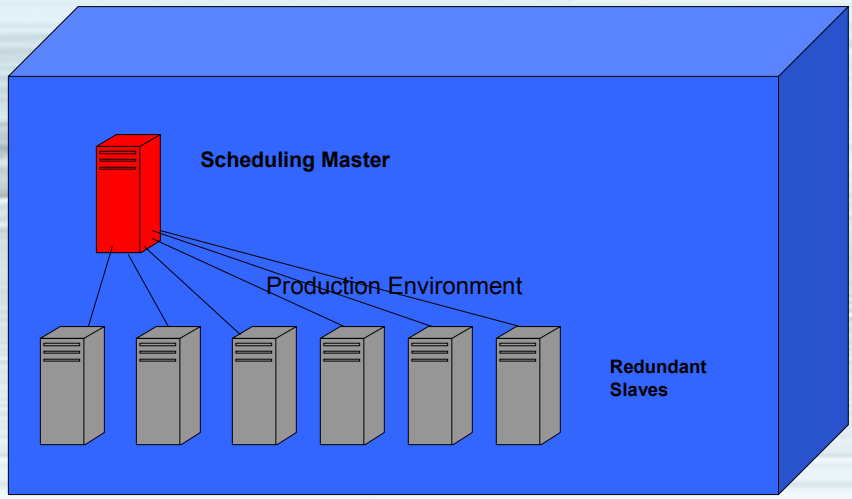
Tool Features and Trade-Offs

- **Interface**
- **Management**
- **Client vs. Server**

Test Tool Management



Test Scheduling



More Tool Features

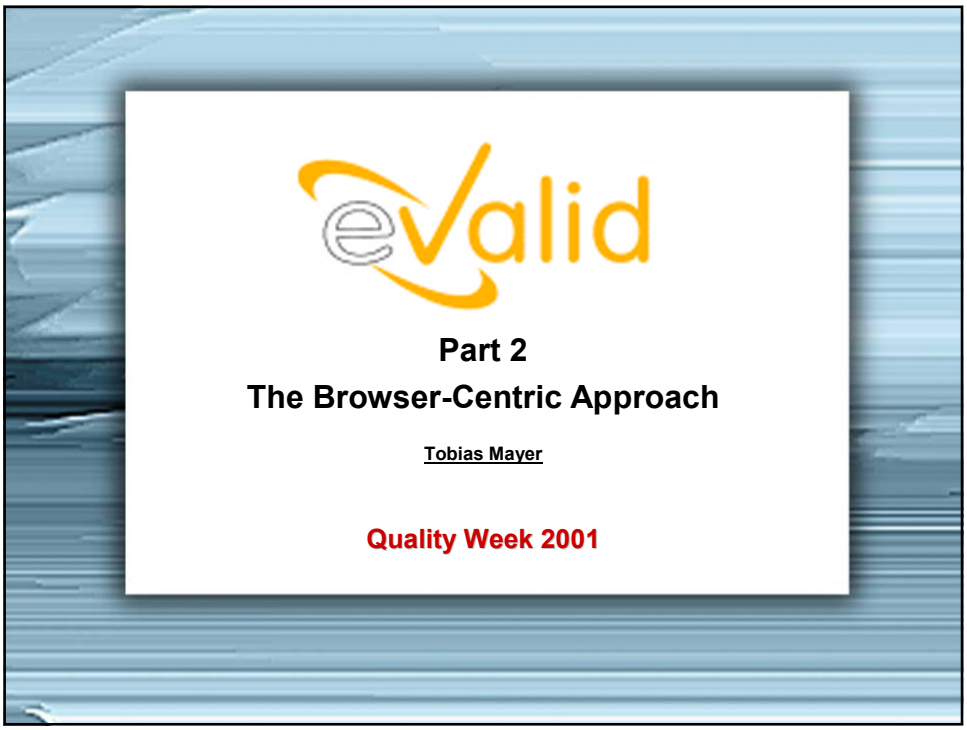
- Implementation models
- Test Management
- Tool Flexibility
- Developer Support vs. Open Source

Getting the Data Out: Reporting

- Availability
  - Reports not critical – up/down status captured by scheduler tool
- Performance
  - Reporting essential – threshold "greening"
- Logging Options
  - Granularity
  - Logging to Text Files
  - Logging to a Database
- Data Presentation
  - Correlation of Availability and Performance Data
  - Graphs

Limits of the Technology

- Stability
  - Tools
  - Networks/Infrastructure
- Poorly Designed Sites
  - Scripting around minor bugs
- Highly Dynamic Sites
  - Scripting around rapidly changing sites
  - Technology/Process Work Breakdown Structure



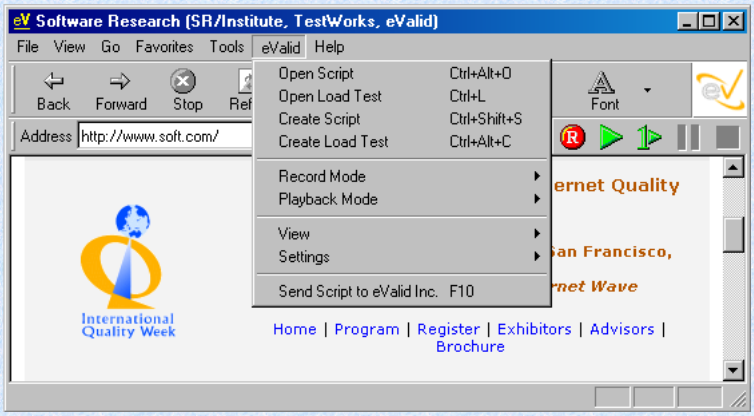
Web Site Regression Test Approaches

Record-Playback of Web Site interaction is generally done at one of three levels:

- **HTTP Messages**
  - **Get, Post, etc.**
- **Event Loop Interception**
  - **Mouse Clicks, Key Events at the 'Desktop' level**
- **Browser Interception** *(a.k.a. 'Browser-centric')*
  - **Direct interfacing with the browser objects**

The Browser-centric Approach

The Test Engine *is* the Browser.



Browser-centric Recording

Test Creation becomes:

- **Simple**
  - **No complex initialization of a separate tool.**
- **Intuitive**
  - **Initialize 'Record Mode' and commence to use the browser in the normal way.**
- **Realistic**
  - **Recorded sessions are precise encapsulations of real user interaction.**

Browser-centric Playback

Test Execution can:

- **Maintain State**
  - **Secure session IDs are always guaranteed to be valid in any playback at any time.**
- **Adapt to dynamically created pages**
  - **No reliance on positioning, size or visibility of the recorded browser objects.**
- **Simulate *Real* Users**
  - **Playbacks faithfully reproduce user interaction.**

Browser-centric Features

Other important features of the Browser-centric approach are:

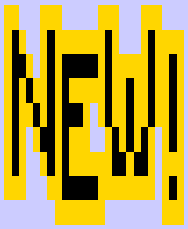
- **Realistic Download Measurements**
  - **Download times are recorded as the user witnesses them, not in some ideal situation**
- **Complete Site Mapping**
  - **Site map of actual links together with realistic download times and selective page filtering.**
- **Page Complexity Measurement**
  - **Access to Browser objects allows detailed complexity metrics to be calculated.**

Conclusion

There are many complex factors to consider when testing a web site.

No single solution has been shown to successfully handle all such factors.

The Browser-centric approach is suggested as an effective, intuitive, accurate and realistic method of testing difficult web sites.



QW2001 Paper 2T2

Dr. Nancy Eickelmann & Mr. Allan Willey
(Motorola Labs)

An Integrated System Test Environment

Key Points

- Automated Test Environments
- COTS tool integration
- System test technologies

Presentation Abstract

Software and system testing is a critical activity in the development of high quality products. When testing is performed manu-ally it is highly error-prone, time consuming and costly. Automated Test Environments (ATEs) overcome the defi-ciencies of manual testing through automating the test pro-cess and integrating testing tools to support a wide range of test capabilities. Industrial use of ATEs could provide signifi-cant benefits by reducing testing costs, improving test accuracy, improving software quality, and providing for test reproducibility. Despite the critical importance of ATEs in the develop-ment of quality products, full life cycle integration of test tools is rarely achieved in practice. To understand what prevents full life cycle integration of tools with respect to data, process, platforms, user interfaces, and control, Motorola Labs is conducting focused R&D initiatives.

The Motorola Automated Test Environment (MATE) initiative provides technology to design, develop, and test high quality software and systems. The Motorola Automated Test Environment initiative addresses a subset of the product development issues focusing on testing, which remains a very costly, time-consuming phase of the product development life cycle.

The Core Process Redesign (CPR) initiative is identifying the necessary integration factors to insert an automated test environment that seamlessly inter-operates with the rest of the product lifecycle. The CPR identifies decision criteria, process input/output pairs, entry/exit criteria and common resource requirements and constraints. The overriding objectives of MATE and CPR are to radically improve time-to-market and predictability in schedules, costs, and quality of product development by:
* Development of a common process that enables consistent practices
* Build a common platform that supports interoperability and reuse
* Increase resource allocation flexibility to facilitate cost effectiveness
MATE and CPR are complementary efforts that focus on standardization, automation and integration of tools, data and processes.

A primary goal of the MATE initiative is to develop a common and automated test environment throughout the corporation. With a common environment the entire corporation would be able to take advantage of optimization improvements made to the environment based on technology advancements external to Motorola and technology acquisition internal to Motorola. Therefore the MATE architecture must accommodate legacy tools, process changes and future needs met by technology insertion. As such the MATE requirements are a negotiated construct among the identified stakeholders.

This paper examines software architectural constraints in relation to software and system test automation environments. The MATE architecture requires a mapping of multiple views of the environment including structure, functionality, process and data. An overview of the environment is represented by mapping the test functionality to hardware and software structures of the environment. An architectural representation provides a foundation for evaluating the impact of architectural and COTS choices on system test engineers and test managers. This paper discusses the integration of COTS tools into MATE including DOORS, Primavera, ClearQuest and ClearCase; the development of standardized test management support TMS; and the automation of the system test process. The MATE architecture is described and a detailed discussion of test automation and tool integration issues is undertaken. Software architectural analysis is used to determine if a specific structural decomposition and the functional allo-cation to system structures supports or impedes certain qualities. Changes to an ATE such as enhancements to system functionality, improvements to performance (space and time), and reuse of components, data representation and changes to processing algorithm are all sensi-tive to system architectural constraints.

About the Author

Dr. Nancy Eickelmann is currently a research scientist for Motorola Labs and is leading the Motorola software and system test process measurement and evaluation research initiative. Prior to joining Motorola she was program manager at the NASA/WVU Software Research Laboratory, her research focused on integrating the Balanced Scorecard into the NASA context to provide a measurement framework for software test technology improvements. Before joining NASA she was a member of the Advanced Programs Research Group at MCC where she developed a measurement framework for guiding the decision-making process in product line development. Dr. Eickelmann began her research career as a member of the technical staff at Hughes Research Laboratory (HRL) in Malibu, California while completing her doctorate at the University of California, Irvine. She was named a Hughes Doctoral Fellow while working at HRL and received several research awards while working with Dr. Debra Richardson's Formal Methods and Software Testing Group at UCI. Dr. Eickelmann has collaborated internationally on research projects for defense systems, space station applications, space shuttle and global software.

Allan Willey is a Member of Technical Staff at Motorola Labs in the Software and

System Engineering Lab (SSERL). Allan leads the "Motorola Automated Test Environments" (MATE) Team. These applied researchers are developing techniques to improve the capabilities of software development groups to test new products. Projects using various advanced statistical analysis techniques, formal modeling, and simulation techniques are being carried out to assess their value for improving delivered product quality, as well as their impact on productivity and testing time. The MATE Team collaborates closely with development organizations in various Motorola product groups to transition successful results. Allan holds an AB in philosophy from the College of William and Mary, and both a BS in mathematics and an MBA in management sciences from George Washington University.

MOTOROLA LABS

MATE

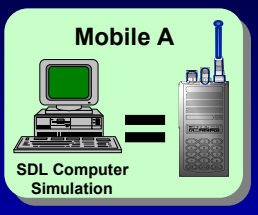
An Integrated System Test Environment

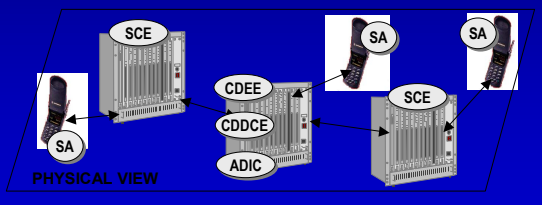
Dr. Nancy Eickelmann
Motorola Labs
1303 E. Algonquin Rd.
Schaumburg, IL 60196
USA
(847)538-0745
(847)576-3280
Nancy.Eickelmann@motorola.com

MOTOROLA LABS

Agenda

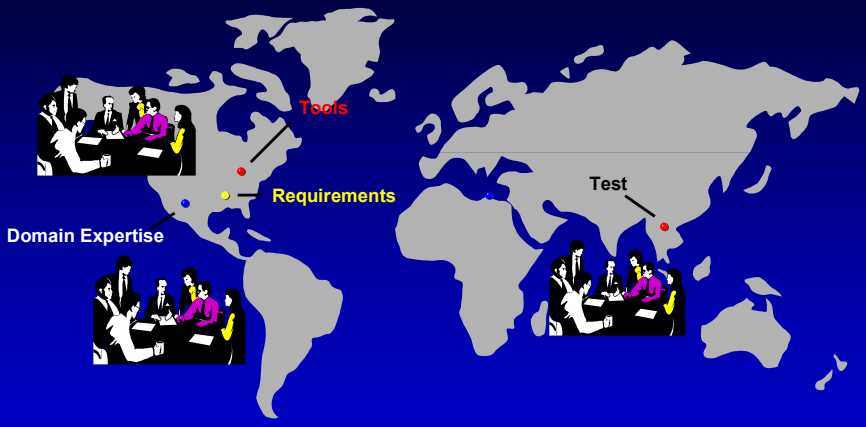
- Why do we need MATE ?
- How do we decompose the problem space ?
- How do we analyze our solution sets ?
- What did we learn ?





MOTOROLA LABS

Distributed Teams



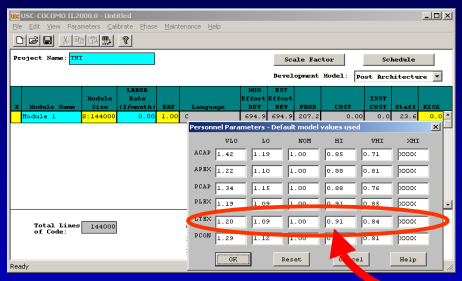
MOTOROLA LABS

Test Management

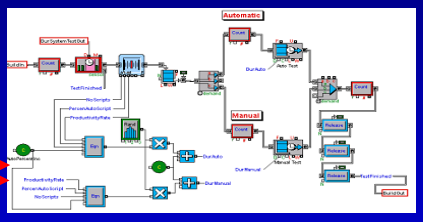
- Resource Allocations
  - Measurement
  - Planning
  - Control
  - Understanding
  - Training and Learning

MOTOROLA LABS

Test... Cost Estimating, Planning and Tracking



SDL Models



$$(NoAutoScripts \times x) + (NoManScripts \times 3 \cdot x) = Duration$$

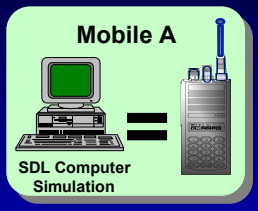
$$\text{Productivity} = x = \frac{Duration}{NoAutoScripts + 3 \cdot NoManScripts}$$

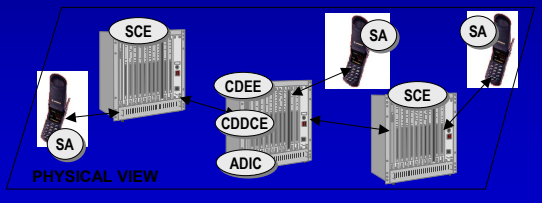
Quality Week May 2001

MOTOROLA LABS

Agenda

- Why do we need MATE ?
- How do we decompose the problem space ?
- How do we analyze our solution sets ?
- What did we learn ?





Dr. Nancy Eickelmann

Quality Week May 2001

MOTOROLA LABS

MATE and CPR

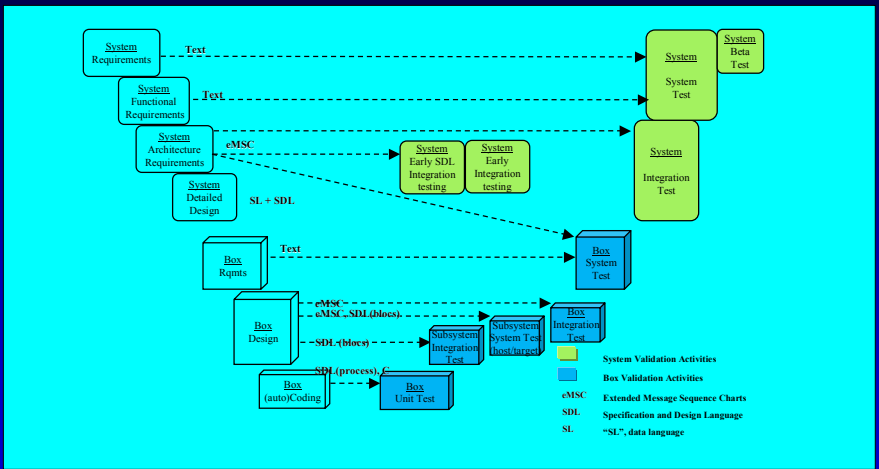
Automation
- Process
- Lifecycle Tools
- Test Technologies

Standardization
- Test Technologies
- Lifecycle Tools
- Process

Integration
- COTS Solutions
- People
- Resources

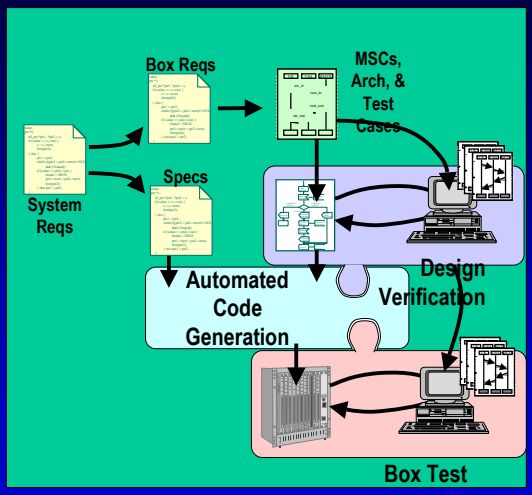
MOTOROLA LABS

Lifecycle View



MOTOROLA LABS

Process View



- Task order
- Work product flow
- Skill sets
- Scheduling
- Task completion criteria
- CM
- RM

MOTOROLA LABS

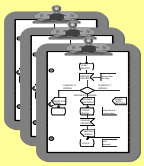
Automation
Test Case Generation

Functional Requirements Specification

MSC with PDU Use

PDU Specs

ptk also performs:
- validation of behaviour
- verification of data use



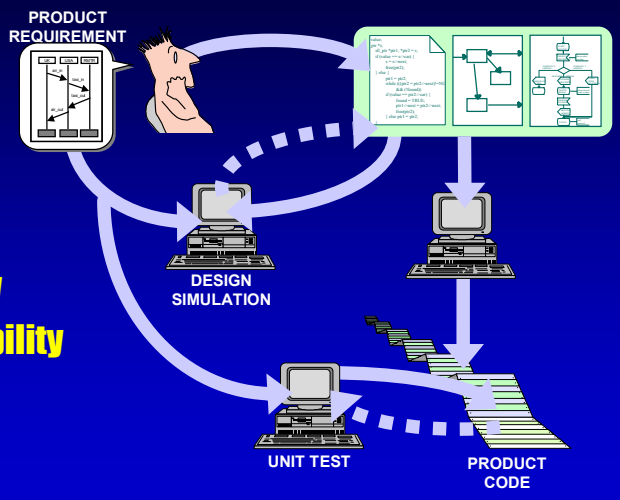
Test Scripts
- SDL
- TTCN

Executable Test Suite

Conformance Test Suite

MOTOROLA LABS

Data View

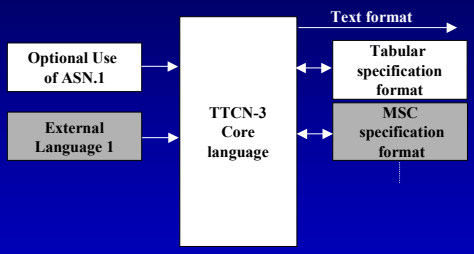


- Traceability
- Reproducability

MOTOROLA LABS

Standardization

- TTCN - 3
- ASN.1
- SDL/SDT- 2000
- MSC - 2000
- UML - 2000



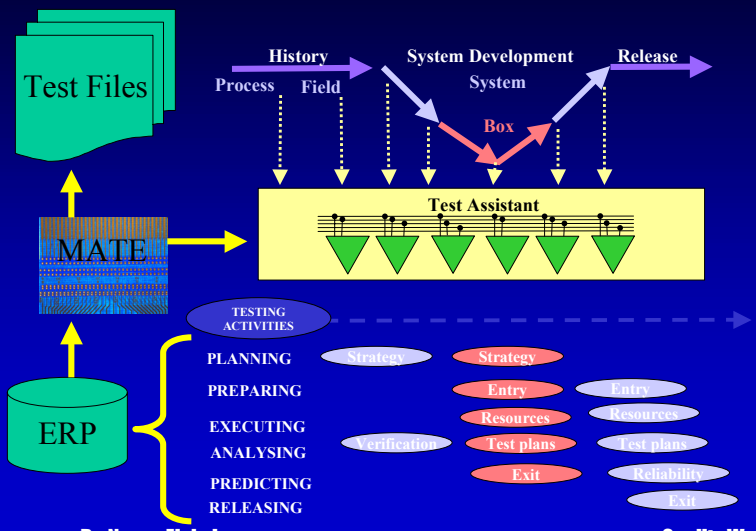
MOTOROLA LABS

Tools and COTS Views

- COTS Integration
  - Analysis and Design tools
  - Test tools
  - CM and PM
  - Resource Management

MOTOROLA LABS

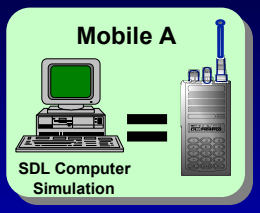
Integration

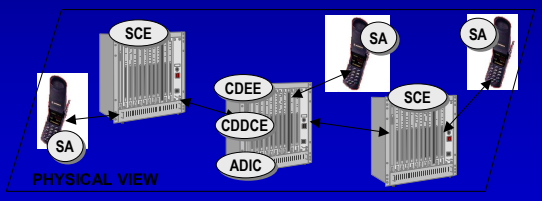


MOTOROLA LABS

Agenda

- Why do we need MATE ?
- How do we decompose the problem space ?
- How do we analyze our solution sets ?
- What did we learn ?





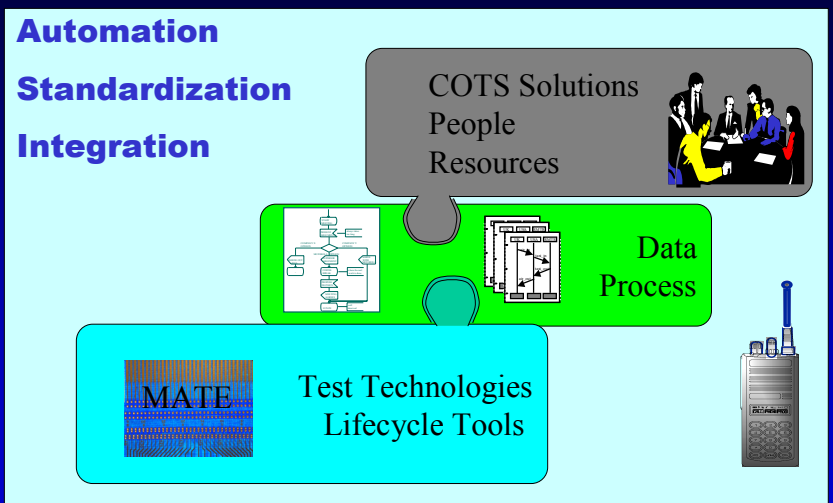
MOTOROLA LABS

Architecture Puzzle

Automation

Standardization

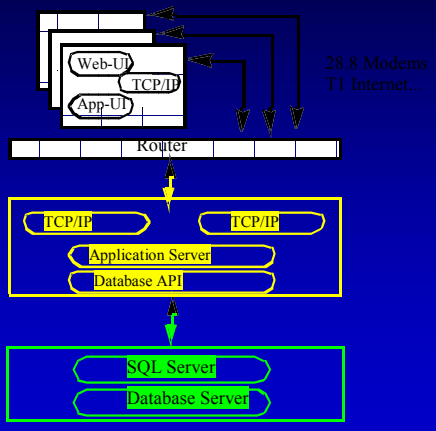
Integration



MOTOROLA LABS

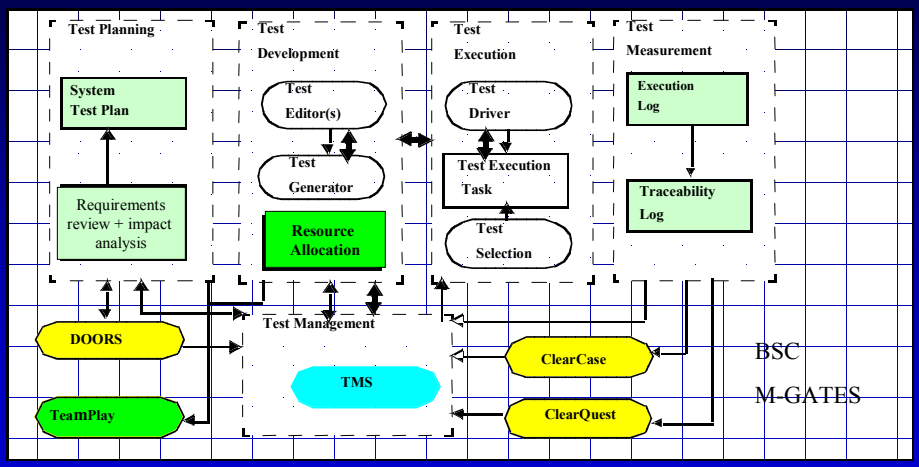
Resource View

- Resource constraints by project, business unit, functional area, country
- Topology requirements
- Security layer requirements



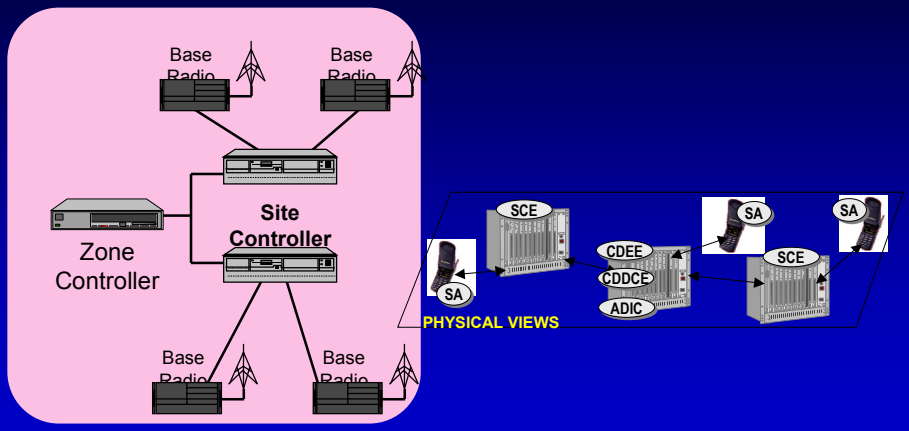
MOTOROLA LABS

Static and Map Views



MOTOROLA LABS

Physical Views



MOTOROLA LABS

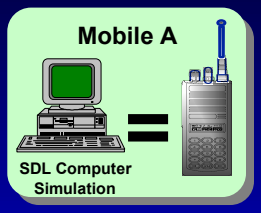
Combined Views

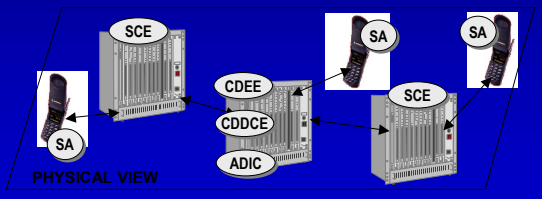
- System Test Functions
- Hardware/Software Structures
- Components/Interfaces
- Object Management
- External Systems

MOTOROLA LABS

Agenda

- Why do we need MATE ?
- How do we decompose the problem space ?
- How do we analyze our solution sets ?
- What did we learn ?





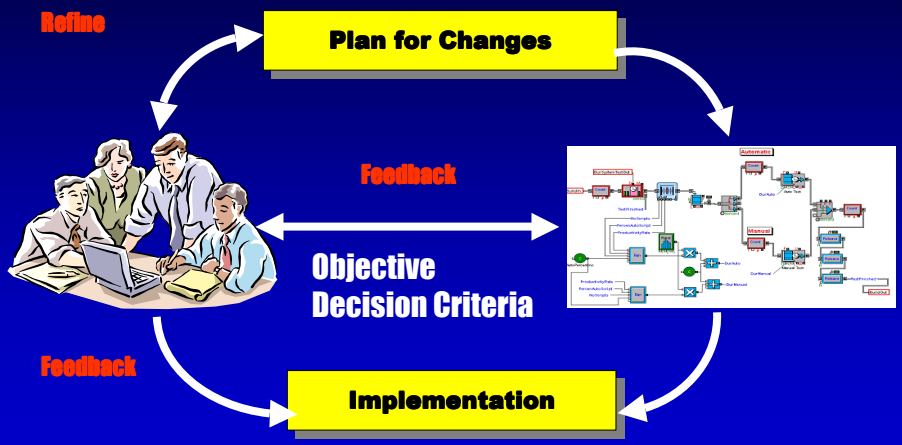
MOTOROLA LABS

Lesson Learned

- Need....
  - better architectural analysis tools
  - understanding of process imposed architecture constraints
  - measures of IT cost/benefit impact at project, business unit and enterprise level

MOTOROLA LABS

Technology Evaluation

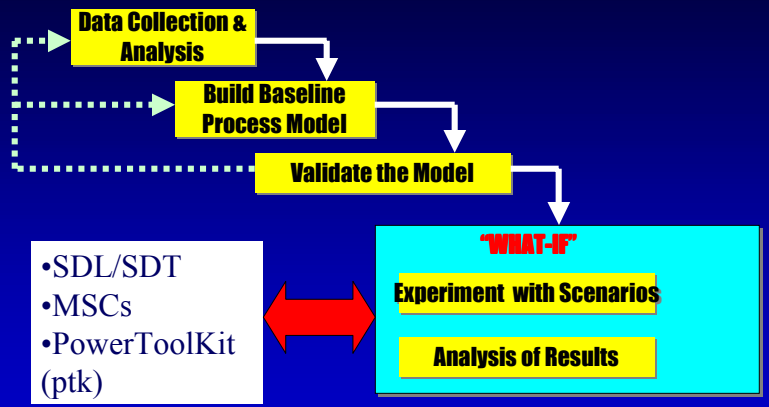


Dr. Nancy Eickelmann

Quality Week May 2001

MOTOROLA LABS

Tool Insertion Impact Analysis



Dr. Nancy Eickelmann

Quality Week May 2001

MOTOROLA LABS

Issues

- Technology
  - » Test Quality
  - » Lifecycle traceability
  - » Test reproducability
  - » COTS interoperability, variability, stability
- Process Maturity
  - » Organization(s) maturity
  - » Multi-site organization maturity variance
  - » Communication and collaboration
- People
  - » Distributed teams of people
  - » Specialization of skill sets required

MOTOROLA LABS

Questions? Answers? Comments?

An Integrated System Test Environment

Nancy S. Eickelmann
Motorola Labs
1303 East Algonquin Road
Schaumburg, IL 60196 USA
+1 847 538 0745
Nancy.Eickelmann@motorola.com

Allan L. Willey
Motorola Labs
1303 East Algonquin Road
Schaumburg, IL 60196 USA
+1 847 576 6343
Allan.Willey@motorola.com

ABSTRACT
The Motorola Automated Test Environment (MATE) provides a means of automating the test process and integrating tools to support required testing capabilities across the lifecycle. The MATE architecture is provided as a foundation to discuss the issues of test tool integration with COTS products that automate the full system test lifecycle. Specific tool integration issues include data, control, process, platforms and user-interface integration issues. The software architecture of MATE can facilitate or impede modifications such as changes to processing algorithms, data representation, or functionality. Architectural analysis is conducted to provide insight into the properties of the Motorola Automated Test Environment.

Keywords
Automated Test Environment (ATE), COTS Tool Integration, System Test.

1 INTRODUCTION

Software and system testing is a critical activity in the development of high quality products. When testing is performed manually it is highly error-prone, time-consuming, and costly [9,11]. Automated test environments overcome the deficiencies of manual testing through automating the test process and integrating testing tools to support a wide range of test capabilities. Industrial use of automated test environments could provide significant benefits by reducing testing costs, improving test accuracy, improving software quality and reliability, and providing for test reproducibility [12]. Despite the critical importance of automated test environments in the development of high quality products, full lifecycle integration of tools is rarely achieved in practice. To understand what prevents full lifecycle integration of tools with respect to data, process, platforms, user-interfaces and control, Motorola is conducting concurrent, focused R&D initiatives, Motorola Automated Test Environment (MATE) and Core Process Redesign (CPR).

The Motorola Automated Test Environment (MATE) initiative provides technology to design, develop, and test high quality software and systems. The MATE initiative addresses a subset of the product development issues focusing on testing, which remains a very costly, and time-consuming phase of the product lifecycle.

The Core Process Redesign (CPR) initiative is identifying the necessary integration factors to insert an automated test environment that seamlessly inter-operates in the context of the overall product lifecycle. The CPR identifies decision criteria (decision gates), process input/output pairs, entry/exit criteria and common resource requirements and constraints.

The overriding objectives of MATE and CPR are to radically improve time-to-market and predictability in schedules, costs, and quality of products developed by:

- development of a common core process that enables consistent practices
- build a common platform that supports interoperability and reuse
- increase resource allocation flexibility to facilitate cost effectiveness

The MATE and CPR initiatives are complementary efforts that focus on standardization, automation, and integration of tools, data, and processes. A primary goal of the MATE initiative is to develop a common and automated test environment. With a common environment the entire corporation would be able to take advantage of optimization improvements made to the environment based on technology

advancements external to Motorola and technology acquisition internal to Motorola. Therefore the MATE architecture must accommodate legacy tools, process changes and future needs met by technology insertion. As such the MATE architectural requirements are a negotiated construct among the identified stakeholders [8].

This paper examines software architectural constraints in relation to software and system test automation environments. The MATE architecture requires a mapping of multiple views of the environment including structure, functionality, process and data perspectives. An overview of the environment is represented by mapping the test functionality to hardware and software structures of the environment. An architectural representation provides a foundation for evaluating the impact of architectural and COTS choices on system test engineers and test managers. MATE integrates COTS tools such as DOORS, ClearQuest and ClearCase; internally developed test management support, TMS; and automation of the system test process.

2 BACKGROUND

Motorola Labs is the research department of Motorola Inc. There are groups around the world conducting research on the leading edge technologies and on advancements to current product offerings. A majority of the research conducted by the Labs is initiated by requests from the product groups within the Communications Enterprise (CE) of Motorola. The CE is comprised of several large business units and accounts for more than half of Motorola's sales. The major businesses in the CE include the cellular handset and cellular infrastructure businesses, the paging device and paging infrastructure businesses, and public and private radio products with their associated infrastructures. All of these products, whether handheld or infrastructure, are computer-based. For the last two decades these products have evolved from a hardware component base to a computer platform with functional layers comprised of software specifically written to address the product requirements. The R&D organizations in Motorola have evolved accordingly so that at this time far more than half of the engineers developing new products worldwide in Motorola are software engineers.

The size of software components in our products vary according to their function. A simple one-way handheld pager will have no more than a few thousand lines of code. A cellular infrastructure product has over twenty million lines of code in its various components. A small core of the infrastructure products are concerned with "call processing," on the order of a few hundred thousand lines of code, and this core will be the most highly reliable part of the system. A similar core component of cellular handset and two-way radios will be key to their functionality, and will consist of tens of thousands of lines of highly optimized code. Numerous additional functions are a part of each product, for example billing tracking and customer authentication components of a cellular system. Some of these, such as authentication, are real-time and interact with the core call processing functions. Others, such as system maintenance functions, are less time-critical but often tend to require large functional blocks.

The rate of growth of the software component of all of these products reflects general telecommunication industry experience. Further, we expect this evolution to continue as future product directions seem to show an unabated demand for new products. For example, while the popularity of simple one-way paging devices is declining, two-way short-message-service paging offerings are a growing market. These two-way devices are an order of magnitude more complex, and contain that much more embedded code.

Numerous other examples can be cited of increased product complexity and size, but underlying these significant changes is the fact that testing has remained a very costly, time-consuming phase of the product development life cycle [2,9,11,12]. As the market pressures increase to deliver more, better, and faster products, testing is being viewed increasingly as a bottleneck to success.

To bypass the testing bottleneck, Motorola has launched focused R&D initiatives to provide technology to design, develop, and test high quality software and systems. The MATE initiative addresses a subset of the issues. A concurrent corporate initiative of Core Process

Redesign (CPR) addresses another view of the issues. The CPR initiative is instrumental in identifying the necessary integration factors to insert an automated test environment that seamlessly interoperates with the rest of the product lifecycle. The overriding objective of CPR is to radically improve time-to-market and predictability in schedules, costs, and quality of product development by:

- Development of a common process across the divisions.
- Identify points of synchronicity across divisions.
- Improve efficiency of processes.
- Identify best practice tools to create a common and global R&D environment
- Build a foundation to propel common platform design and reuse
- Increase resource allocation flexibility across the corporation.

The establishment of the Motorola automated test environment contributes to all of the above goals as well. The testing activities have tremendous effect on product time-to-market, cost and quality [3,4]. The above goals can also be applied to internal product deliveries from tool development groups. Establishment of common tools and processes eliminates the redundancy of the internal tool groups and allows for reallocation of personnel to focus on product delivery and not internal support. It also allows for innovative ideas to be implemented in a common platform and taken advantage of by a multitude of users instead of restricting technological benefits to local groups. The MATE initiative will bring Motorola closer to achieving its goals. MATE focuses on standardization, automation and integration of tools, data and processes. To facilitate our discussion and provide common definitions for terms we introduce the automated test environment reference architecture used in our analysis.

3 AUTOMATED TEST ENVIRONMENT REFERENCE ARCHITECTURE

A reference architecture, the STEP model [5] provides a basis for the representation and formalization of the MATE architecture. The reference architecture segments the required functionality for the environment and captures the relationships among the functionalities in a diagrammatic format.

The domain is partitioned into six canonical functions: test execution, test development, test failure analysis, test measurement, test management, and test planning. Each of these functions is defined to provide consistency in the use of terms.

- **Test Execution** includes the execution of the instrumented source code and recording of execution traces. Test artifacts recorded include test output results, test execution traces, and test status.
- **Test Development** includes the specification and implementation of a test configuration. This results in a test suite, the input related test artifacts, and documentation. Specific artifacts developed include test oracles, test cases and scripts, and test adequacy criteria.
- **Test Failure Analysis** includes behavior verification and documentation and analysis of test execution pass/fail statistics. Specific artifacts include pass/fail state and test failure reports.
- **Test Measurement** includes test coverage measurement and analysis. Source code is typically instrumented to collect execution traces. Resulting test artifacts include test coverage measures and test failure measures.
- **Test Management** includes support for test artifact persistence, artifact relations persistence, and test execution state preservation. Test process automation requires a repository for test artifacts. A passive repository such as a file serves the basic need of storage. However, an active repository is needed to support relations among test artifacts and provide for their persistence.
- **Test Planning** includes the development of a master test plan, the features of the system to be tested, and detailed test plans. Included in this function are risk assessment issues, organizational training needs, required and available resources, comprehensive test strategy, resource and staffing requirements, roles and responsibility allocations, and overall schedule."

The STEP model, shown in Figure 1, stratifies test functionalities from the apex of the pyramid to its base in a corresponding progression of the test process lifecycle as described in [9]. The test process evolution is aligned with the arrow to the right of the pyramid and segments test functionality according to the test lifecycle focus. Each segment represented in the pyramid includes the functionalities of previous periods as you descend from the apex to the base.

The top section of the pyramid represents the function of test execution. Test execution is clearly required by any test process. The test process focus of debugging includes only test execution.

The second segment of the pyramid, from the top, is divided into two scalene triangles.

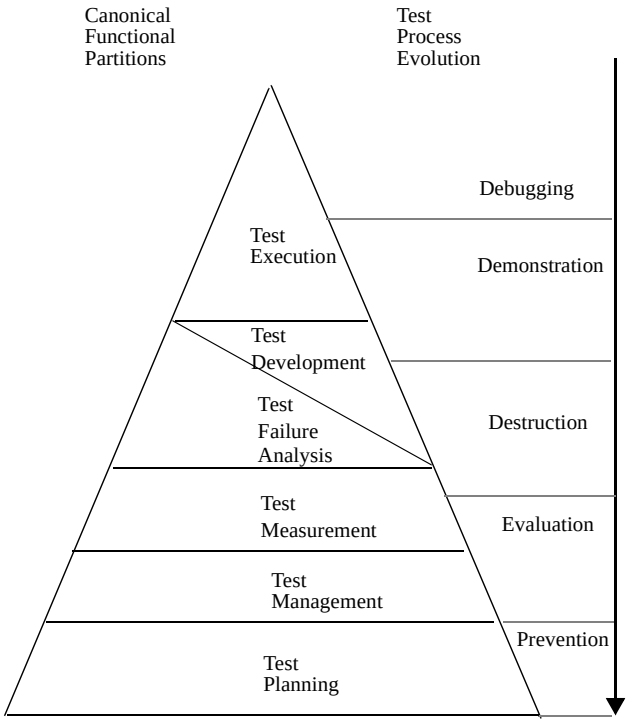


Figure 1. STEP model reference architecture [5].

The smaller scalene triangle represents test development. The larger scalene triangle represents test failure analysis. The relative positions and sizes have semantic significance

Test development played a more significant role to the overall test process when focused on demonstration and destruction due to the manual intensive nature of test development. Test development methods have not significantly changed, although they have improved in reliability and reproducibility with automation. Thus, their role in test process diminishes in significance as you automate the test development process.

Test failure analysis is less important when performed manually, as interactive checking by humans adds little benefit for test behavior verification. The methods that can be applied to test failure analysis have increased in their level of sophistication, making test failure analysis more significant to the overall test process. One of the most significant advances is specification-based test oracles [5]. This is a key difference in test process focus.

Test measurement is represented by the third segment in the pyramid. Test measurement is required to support an evaluation focus for test, which represents a full lifecycle approach. A significant change in the test process focus is that testing is applied in parallel to development, not merely at the end of development. Test measurement also enables evaluating and improving the test process.

Approaching the base of the pyramid, the fourth segment represents test management, which is essential to the test process due to the sheer volume of information that is created and must be stored, retrieved, and reused. Test management is critical for test process reproducibility.

The base, or foundation, of the pyramid is test planning. Test planning is the essential component of prevention focused test efforts. Test planning introduces the test process before requirements, so that rather than being an afterthought, testing is pre planned and occurs concurrently with development. The STEP model provides the core functionality and process focus to describe the MATE architecture. The architectural description requires integration of multiple views of the test environment including structure, functionality, process and data.

In the next section, the MATE architecture is described and a detailed discussion of test automation and tool integration issues is undertaken. Software architectural analysis is used to determine if a specific structural decomposition and functional allocation to system structures supports or impedes achieving desired properties for MATE. Changes to an ATE such as enhancements to system functionality, improvements to performance (space and time), and reuse of components, data representation and changes to processing algorithms are all sensitive to system architectural constraints [8].

4 MATE ARCHITECTURAL VIEWS

There a several architectural views in the literature including: Map View (mapping of functions and components), Static View (structure diagram), Resource View (mapping of software onto hardware), Dynamic View (operational diagrams).

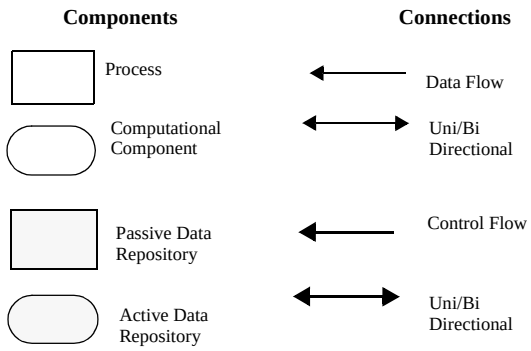


Figure 2. SAAM graphical architectural notation [10].

We provide a diagrammatic representation of the static, map, and resource architectural views for MATE. The static and map views are combined by using the SAAM notation. The resource view uses a generic representation.

4.1 MATE Architectural Static View and Map View

The Software Architectural Analysis Method (SAAM) provides a concise notation that includes a static view and a map view [10]. The static view represents the structure as a decomposition of the system components and their interconnections. The map view groups the components according to their high level functionality. The canonical functions for MATE were defined in the reference architecture.

The SAAM graphical notation is shown in Figure 2. In this notation there are four types of components: a process (unit with an independent thread of control); a computational component (a procedure or module); a passive repository (a file); and an active repository (database). There are two types of connectors: control flow and data flow, either of which may be unidirectional or bidirectional.

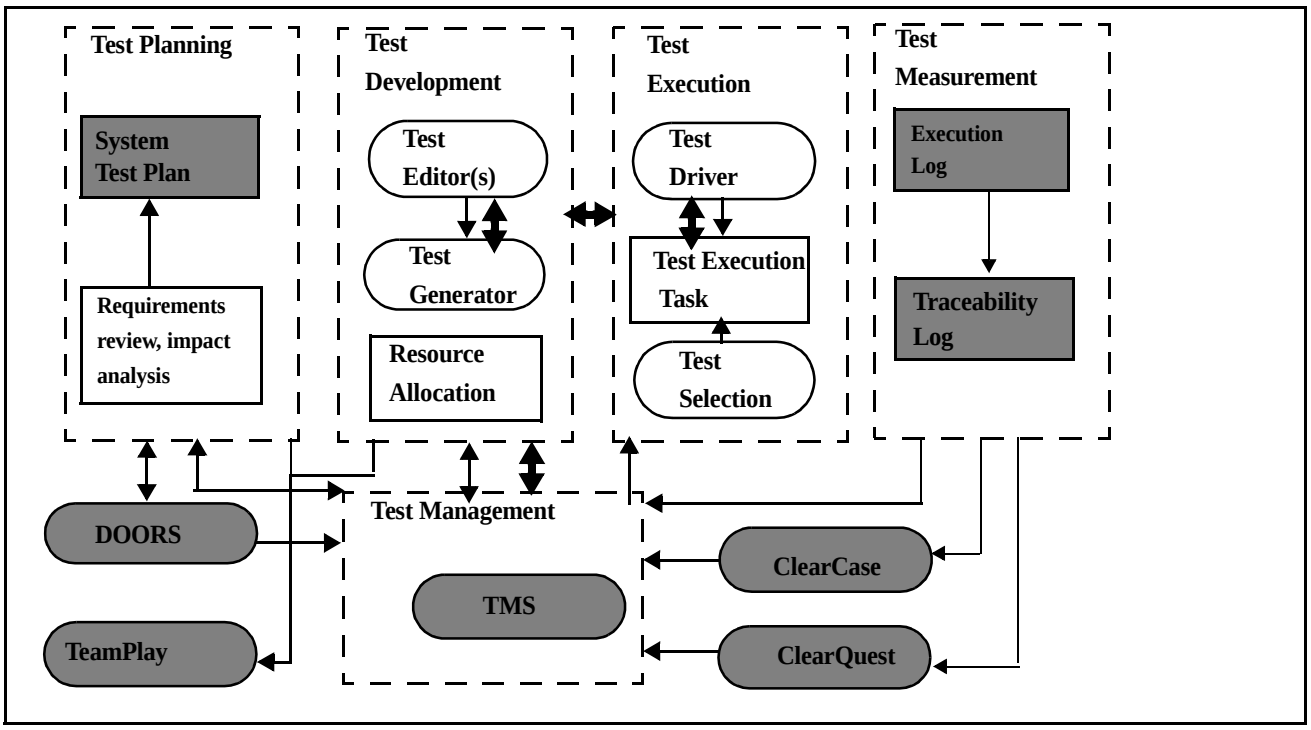


Figure 3. MATE Architecture Static and Map Views

The allocation of domain functionality to the software completes the graphical representation of the ATE, see Figure 3. The allocation provides the mapping of a system's intended functionality to the concrete interpretation in the implementation. The diagram segments the canonical functions for MATE using a broken line. The COTS active repositories are not included in the domain functionality as they interact through the TMS management system. The five functionalities inclusive in MATE, test execution, test development, test measurement and test management are discussed below.

Test Execution includes the execution of the system in the target environment testbed, and recording of execution traces. Test artifacts recorded include test output results, test execution traces, and test status. The test execution environment must provide a bridge to interface with legacy solutions that must be migrated in an evolutionary versus revolutionary approach. Testing in a distributed context with heterogeneous platforms was considered desirable for flexibility and reusability. Another factor of significance was language independence and open interfaces for interoperability.

Test Development includes the specification and implementation of a test configuration. This results in a test suite, the input related test artifacts, and documentation. Specific artifacts developed include test oracles, test cases and scripts, and test adequacy criteria. Test development solutions should support multi-platform and multi-site access to data while providing adequate response time and usability in the user interface. Automated test case generation solutions focused on SDL and MSC model specifications that support auto test case generation. Significant process and data integration issues arose that were addressed by interfacing ClearCase test data with TMS and ClearQuest CM and defect data with TMS. Resource allocations for system test are documented in TeamPlay and provided on-line to the test developers.

Test Measurement includes documentation and analysis of test execution pass/fail statistics. Specific artifacts include pass/fail state, test failure reports and test traceability reports. MATE's support for test measurement must be addressed across all the functionality and interface with legacy data integration issues for test and development groups. A distributed collection and repository tool provides for heterogeneity with transparency of access issues through web-enabled tools.

Test Management includes support for test artifact persistence, artifact relations persistence, and test execution state preservation. Test process automation requires a repository for test artifacts. A passive repository such as a file serves the basic need of storage. However, an active repository is needed to support relations among test artifacts and provide for their persistence. Test management should provide a state-of-the-art solution with web-based access and interface. TMS was available on UNIX platforms and was being migrated to an NT platform as well. Since it was Java based, TMS would easily adopt to new platforms as Motorola test groups use them. Prototypes of TMS had been operational on SUN Solaris, HPUX, Windows NT, Windows 98, MacOS and Linux. TMS has been designed to take advantage of the multi-site features of Oracle and a software configuration management tool by using both as underlying technology to the tool. The TMS API will also allow for a quick replacement of either technology as improved technologies become available. A significant finding through the MATE effort was that terminology was not used uniformly throughout the organization. A living glossary was implemented continues to track and resolve discrepancies in word usage and understanding.

Test Planning includes the development of a master test plan, the features of the system to be tested, and detailed test plans. Included in this function are risk assessment issues, organizational training needs, required and available resources, comprehensive test strategy, resource and staffing requirements, roles and responsibility allocations, and overall schedule. Test planning requires inputs from the DOORS repository and TMS test management function. Test planning also requires project management tool support which is provided by TeamPlay. Test planning is not well supported by most commercially available tools as it requires integration with data repositories, process issues, and tool interfaces.

4.2 Resource View

A common representation for architecture is the resource view. This view of the MATE architecture is diagrammed in generic terms such as provided in Figure 4.

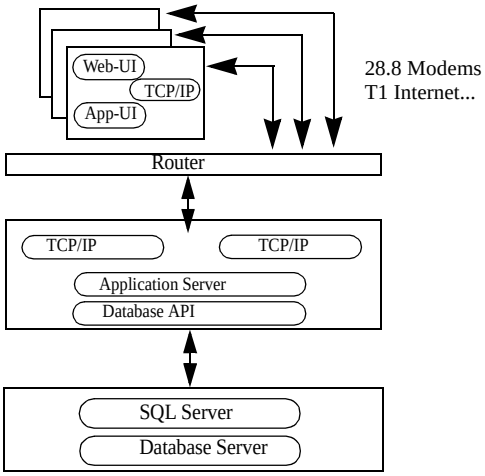


Figure 4. MATE Architecture Resource View.

The resource view focuses on essential communications links such as routers, modems, internet and intranet connections and the architectural topology, client/server. Analysis of this view lends itself to performance analysis, queueing models and throughput simulations, see Table 1. This view is also used to evaluate system availability constructs using a composite quantification of the system hardware failure/time interval and the corresponding software failure/time interval. Mean time to repair is estimated for hardware and corresponding software providing an incidence frequency with an estimate in hours to restore the system. The declaration of system hardware and software computational units also provides for an affordability analysis. The initial investment costs, updates, and maintenance costs can be obtained from the vendors of choice. Given the payback period, and the minimum time required for return on investment, a Net Present Value (NPV) comparison can provide an objective valuation of hardware and software configurations.

Table 1: Performance Analysis

Scenarios	Comments
Network latency from client to server and from server to client in ms.	Latency time provided by network specification
Server latency consists of dequeueing time (Cdq=ms) and task computation time (Cfnc=ms)	Latency time provided in server specification
Distribution time of periodic updates minimum	8@10sec;8@20sec;8@40sec;8@80sec
Distribution time of periodic updates maximum	48@10sec;24@20sec;12@40sec;8@80sec

5 CORE PROCESS DESCRIPTION

The CPR identifies decision criteria (decision gates), process input/output pairs, entry/exit criteria and common resource requirements and constraints. The CPR initiative provides a high level common process with fixed decision gates. However, it also provides for insertion of customized process segments for insertion into organizations with legacy tools and technologies that impose unique process constraints. The software development process and software and system test groups in Motorola follow various lifecycle models that comply with internal standards. An external standard that provides a high level process description for test is the IEEE Std. 1012-1998, Standard for Software Verification and Validation. This current standard is an update of the IEEE Std. 1012-1986. The updated standard is comprehensive in its scope and details full lifecycle activities and cross-references according to compatibility with other process standards. The substantial benefit of a well-defined process with objective decision criteria is realized when it is instantiated in an automated test environment.

6 TECHNOLOGY TRANSITION PLAN

The MATE transition effort is analyzed in the context of the 3 vector space of the SEI Technology Transition Conceptual Framework [6], see Figure 5. The 3 vector space represents the relationships among an increasing magnitude of technological change; the required effort to adapt or learn new skills, procedures, structures, strategies, or culture; and time required to adapt

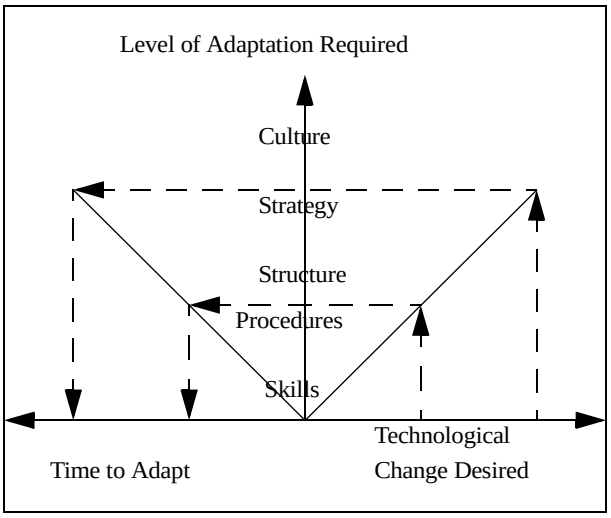


Figure 5. Dimensions of technology transition adapted from [CMU/SEI-93-TR-31]

and institutionalize the changes. This structure supports evaluating the "size" of a change based on the level of adaptation required. We will reference this figure in our subsequent discussion of MATE. The MATE effort as described using the SEI's technology life cycle conceptual framework depicts an effort towards a common test environment that is comprised of technologies in various phases of maturity. The majority of MATE technologies will require new skills and procedures as a foundation for their introduction. These changes are seen as requiring typically a time horizon measured in months.

The time required to achieve a technology transition is partially dependent on an additional classification of the technology life cycle. The technology life cycle includes 3 distinct phases, R&D, new product development, and adoption and implementation. Tools and technologies of MATE span the full technology life cycle and therefore may be described as short term to long term change efforts. The insertion of MATE represents new technologies for the system test organizations, however the technologies include both recent R&D developments of Motorola Labs and mature commercial tools to be integrated into the common environment.

We outline the steps of each of the 3 technology life cycle phases as described in Figure 6. The focus of phase 1, R&D of the technology life cycle, is primarily the technology itself. Inclusive in the R&D phase of the life cycle;

- concept formulation
- development and extension
- enhancement and exploration (internal)
- enhancement and exploration (external)
- early popularization.

The aspects relevant to MATE in phase 2, the new product development phase includes;

- generating new product ideas
- screening those ideas
- testing product concepts

Adoption and implementation, phase 3 of the technology life cycle includes multiple steps relevant to the MATE transition effort;

- needs assessment
- selection of candidate products
- evaluation of candidate products
- introduction of product to user groups, management, stakeholders
- gathering feedback from target groups
- implementation planning and execution

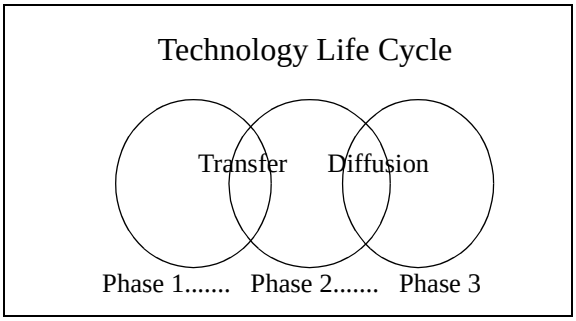


Figure 6. Technology life cycle phases adapted from [CMU/SEI-93-TR-31].

We will use the steps outlined for the 3 phases of the technology life cycle in describing the MATE effort and will reference the tools and technology maturity with regard to this structure as appropriate.

Test execution was comprised of primarily mature technologies in phase 3 of the technology lifecycle, yet aspects of the total solution for full automation and integration might represent less mature technologies. Test case generation

technologies were seen as being in the transfer phase, or phase 2, of the technology lifecycle and experiencing the early stages of transfer. The technology lifecycle maturity of the test management technologies are typically in the third phase of the lifecycle. Test measurement technologies were primarily in phase 3 of the technology lifecycle and represented the diffusion of measurement technologies.

The MATE effort as described using the SEI's technology lifecycle conceptual framework and technology lifecycle maturity evaluation depicts an effort towards a common test environment that is comprised of technologies in various phases of maturity. The majority of MATE technologies are in phase 3 of the technology lifecycle and are well-supported for successful transition.

7 SUMMARY

The Motorola Automated Test Environment initiative will enable Motorola to optimize the test process through standardization of tools; automation of manual processes; and integration of data, processes, and interfaces across a common test environment. Specific benefits of the MATE initiative relate to test management, automated test script generation and execution, UI platform independence, and improved quantification of release criteria.

- **Test management** - Provides a central controller for the test environment, test artifacts and test data.
- **Automated test script generation and maintenance** - Provides the automatic generation of test scripts from a formal requirement representation. Maintenance of the test scripts as the requirements change is facilitated.
- **Automated test execution** - Alleviates staffing shortages at critical process bottlenecks.
- **Web front end** - Provides for access to the test tools and the data from anywhere on any platform. The web was seen as the optimal solution.
- **Reliability prediction** - Provides objective criteria to determine when a product can ship based on predicting remaining defects in a product after a test cycle.

Providing for continuous improvements in core processes often requires the insertion of tools and technologies to automate and standardize industrial practice. Understanding the strengths of tools and applying them appropriately in the context of the organizational maturity is essential.

REFERENCES

1. Brown, Alan W., Earl, Anthony N. and McDermid, John A., Software Engineering Environments: Automated Support for Software Engineering, McGraw-Hill, 1992.
2. Eickelmann, Nancy S., "Emerging Test Technologies: Shifting Tester Requirements," In the Proceedings of the 17th International Conference on Testing Computer Software, June 12-16, 2000.
3. Eickelmann, Nancy S., "Measuring and Evaluating the Software Test Process." European Software Measurement Conference, FESMA `98, Antwerp, Belgium, May 6-8, 1998.
4. Eickelmann, Nancy S. and Richardson, Debra J., "Leveraging the Cost of Software Testing with Measurable Process Improvement," In the Proceedings of the Computing in Engineering Conference, ETCE-ASME `97, Houston, Texas, January 28-30, 1997.
5. Eickelmann, Nancy S. and Richardson, Debra J. "An Evaluation of Software Test Environment Architectures" in proceeding of the Eighteenth International Conference of Software Engineering, 1996.
6. Fowler, Priscilla and Levine, Linda "A Conceptual Framework for Software Technology Transition", Software Engineering Institute Technical Report CMU/SEI-93-TR-031, December 1993.
7. D. Garlan, G. Kaiser, and D. Notkin. "Using tool abstraction to compose systems." IEEE Computer, vol. 25, June 1992.
8. D. Garlan and M. Shaw. "An introduction to software architecture.". Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific Publishing Co., 1993.
9. D. Gelperin and B. Hetzel. "The growth of software testing." Communications of the ACM, 31(6):687-695, June 1988.
10. R. Kazman, L. Bass, G. Abowd, and M. Webb. "SAAM: A method for analyzing the properties of software architectures." In Proceedings of the Sixteenth International Conference on Software Engineering, 81-90, Sorrento, Italy, May 21, 1994.
11. G. J. Myers. The art of software testing. New York, John Wiley and Sons, 1978.
12. E. Miller. Mechanizing software testing. TOCG Meeting, Westlake Village, California, April 15, 1986.
13. Guideline for Lifecycle Validation, Verification, and Testing of Computer Software. National Bureau of Standards Report NBS FIPS 101. Washington, D.C., 1983.

QW2001 Paper 3T1

Mrs. Manjula Madan
(Philips Software Centre, Bangalore)

Defect Reduction Using Orthogonal Defect Classification Methodology

Key Points

- Automated Test Environments
- COTS tool integration
- System test technologies

Presentation Abstract

At the highest level, the eight ODC attributes of a defect i.e. Activity, Trigger, Target, Defect Type, Defect Qualifier, Source, Impact, and Age, truly capture orthogonal (non-redundant) pieces of information. They are designed to be at the right level of granularity (not too large to be useless and too fine to be exhausting) and arguably sufficient to answer most questions of practical interest on the software.

At the level of the individual attribute, the orthogonality (say, Defect Type) relates to the fact that the Defect Type distribution describes the state of a software product much the same way the three (orthogonal) Cartesian coordinates describe the location of an object in a three dimensional space. Notice that the evolution of a software product through a schedule is similar to the motion of an object through space and time.

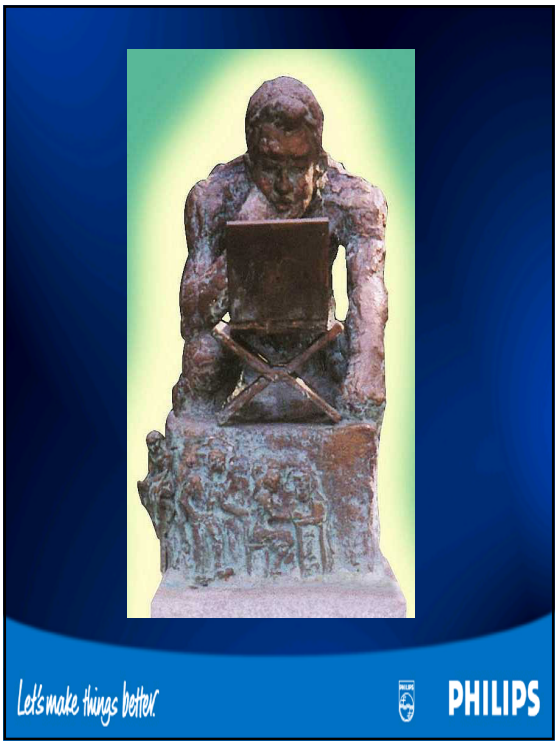
Orthogonal Defect Classification (ODC) provides a good framework for cause-effect analysis. Defect Trigger is also a good idea for providing insight over verification process. In order to implement this defect classification framework, one still has to come up with: * * * *
* Effective attributes to be measures
* Process for analysing attributes
* Action plan based on the analysis result for process improvement

Action plan is independent of ODC. However, action plan is required for process improvement. It would be good if benchmark for applying ODC is available. This will help in applying and analysing ODC results. This can be achieved only by accumulating data over a period of time.

About the Author

Manjula Madan, is working at Philips Software Centre, Bangalore, in the capacity of a Software Quality Engineer from the past 7 months. She has a total of 7 years experience in the IT Industry in which 3 years is in the areas of Quality Control and Quality Assurance. Previous to Philips she was working as a Software Quality Analyst at IBM Global Services India Limited.

Achievements & Awards:
- She is a Certified Quality Analyst (CQA)
- Bronze Medallist in the Programming Exams conducted by National Computer Education, United Kingdom
- Part of the Core Team involved in CMM Level 5 Assessment at IBM Global Services India, Bangalore
- Part of the Core Team involved in CMM Level 5 Assessment at Philips Software Centre, Bangalore
- Part of the team which got ISO 9001 Certification at HCL Perot Systems, Bangalore
- The Customer of the project in which Manjula is currently working has given her a rating as "Excellent SQE"



Our Experiences in Defect Reduction using Orthogonal Defect Classification Methodology

Manjula Madan
Philips Software Centre, Bangalore, India
manjula.madan@philips.com

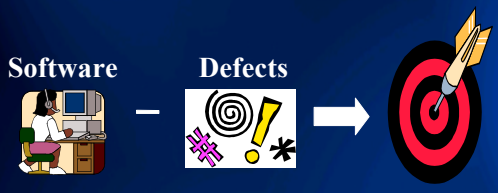
Let's make things better.
PHILIPS

CONTENTS

Introduction

- " Most software professionals spend much of their working lives reacting to defects. They know that each individual defect can be fixed but that its near twin will happen again, and again, and again ….."

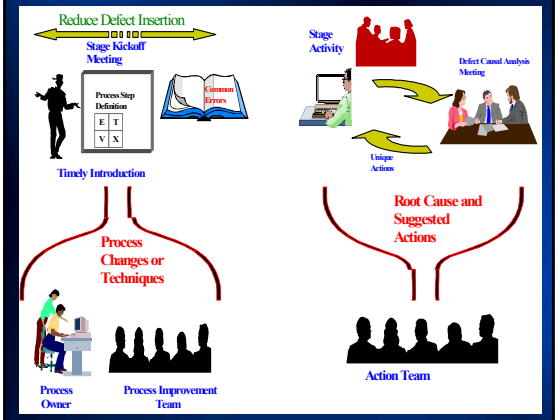
- Watts S Humphrey



Introduction (Cont'd)

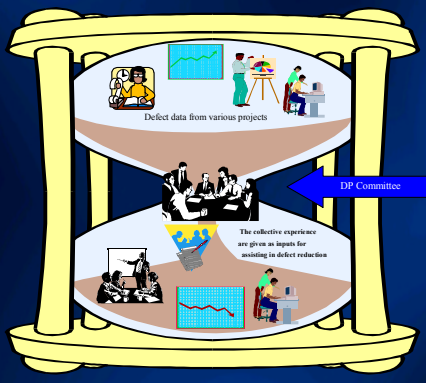
- Traditionally, defects represent the undesirable aspects of software quality
- Defect Prevention (DP) forms the essence of Total Quality Management
- DP - key process area in level 5 of the Capability Maturity Model (CMM)
  - Modeled on techniques used in Japan for decades and is in agreement with Deming's principles
  - Based on three simple steps:
    - Analyze existing defects or errors to trace the root causes
    - Suggest preventive actions to eliminate the defect root causes
    - Implement the preventive actions

Self-correcting Closed Loop System



Way of Working of DP at PSC

- Defect Prevention Committee is "the Organization level team to coordinate defect prevention activities and to provide necessary impetus to the software process improvement"



DP Committee Charter

- Activities performed by the committee :
  - Facilitate DP Training's
  - Ensure activities w.r.t. DP Process are carried out within the projects and LOB's
  - Facilitate documentation of the DP data and tracking
  - Presentation of LoB trends and Provide Consultation on the resulting defect trends
  - Facilitate Learning's across organization of the DP activities carried out Within projects and Within LOB's
  - Facilitate Systemic Corrections and updations / revisions of the DP Process or any other organization's standard software processes resulting from the defect prevention actions
  - Quantify the (in terms of effort and hence cost) benefits obtained as a result of the DP activities
  - Facilitate periodic review of the defect prevention activities by the senior management
  - Facilitate the availability of the necessary tools required to support defect prevention activities

Orthogonal Defect Classification

A Concept for In-Process Measurements

- Invented by Ram Chillarege at IBM Research
- A measurement concept for software development
- ODC brings a quantitative method useful for product management, productivity analysis, quality control and cost management
- ODC makes it possible to push the understanding and use of defects well beyond quality
- Defects is classified on the basis of 8 angles, which are:
  - ***Activity***
  - ***Trigger***
  - ***Defect Target***
  - ***Defect Type***
  - ***Impact***
  - ***Source***
  - ***Defect Qualifier***
  - ***Age***

Improvement cycle

Act - Verify/Monitor Implementation, Report to SEPG, Provide inputs to the Software Process Improvement Plan for continuous improvement based on improvement areas identified at organization level

Plan - Prepare a plan to facilitate implementation of Defect prevention activities throughout PSC

Check - Review of Defect Prevention activities and Results, Track Implementation / progress/trends & Causal Analysis/Root Cause Analysis/Feedback

Do - Provide consultation, execute DP activities , Use ODC methodology for defect classification, Identify the areas of Improvement based on defects and common causes, Facilitate sharing of best practices of DP within and outside PSC

Deploying ODC at PSC

- Ample and Effective way of classification of Defects
- 3 angles used
  - Defect Type
  - Defect Qualifier and
  - Activity
- Collective experiences of the past projects executed at PSC using the above 3 angles were considered
- I am presenting 2 Case Studies showing Defect Reduction

Explanation

Defect Type

- Assignment/Initialization
- Checking
- Algorithm/Method
- Function/Method
- Timing/Serialization
- Interface/O-O Messages
- Relationship

Defect Qualifier

- Missing
- Extraneous
- Incorrect

Activity

- Detailed Design Review
- Code Review

Intro to Project "A"

- Project "A" is a multi-site project with 4 site locations
- Aimed at developing a DVD Player with Recorder
- Product to have the Player functionality inherited from the Philips DVD Player and the Recording functionality inherited from the Philips VCR
- 5 sub systems in the Project
- PSC responsible for the entire
  - Front controller
  - Parts of RCS,
  - Assists customer with
    - User Interface
    - Audio/Video Switching
    - P50 software
- PSC responsible for bug fixing in the entire RCS and FRC software
- The team was actively involved in the Requirements Analysis and the Top Level Design at the overall Project Level

Phases of the Project

- Component Design
- Implementation
- PR solving (current phase)

- During Implementation phase defect classification on the lines of ODC thought about
- This was done to assist causal analysis
- Project Team involved in classification of defects
- Only code review defects considered for classification

Phase 1 Analysis

Defect Signature

Defect Type	% Defects
Assignment/Initialization	61
Algorithm/Method	23
Checking	14

Defect Qualifier	% Defects
Incorrect	54
Missing	34

Corrective and Preventive Actions (1)

- Assignment/Initialisation:
  - Causal Analysis
    - Most of the defects were found to be due to oversight in re-initialising
  - Preventive Action
    - Code review and Design review checklist to be updated to check for re-initialisation
    - Also the usage of flags to be avoided during debugging
    - The Design Guidelines to be updated with respect to this

Corrective and Preventive Actions (2)

- Algorithm/Method :
  - Causal Analysis
    - Most of the defects were due to scattered inputs and re-use of the code
  - Preventive Action
    - For scattered inputs and re-used code, have a Design Kick-off meeting
    - Understand the functionality of the re-used code
    - Have a kick-off meeting with the Architect and Requirements Engineer before the design, especially when the requirements are close to hardware
    - Training to be provided
    - Update the Design Guidelines

Corrective and Preventive Actions (3)

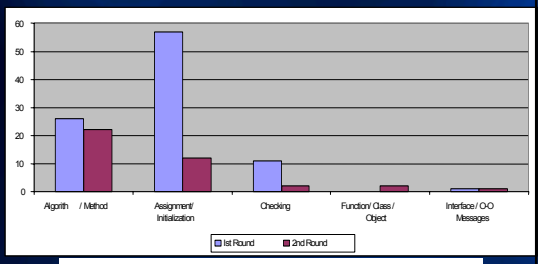
- Checking
  - Causal Analysis
    - Most of the defects were found to be due to oversight
  - Preventive Action
    - Update the design and code review Checklist to check for the incorrect or missing validation of parameters

Corrective and Preventive Actions (4)

- Incorrect
  - Causal Analysis
    - Most of the defects were found to be due to oversight
  - Preventive Action
    - Update codes review checklist and design review checklist to check for omission of qualifier

Phase 2 - Analysis

- At the end of the Implementation Phase, the defect data of the rest of the code reviews was consolidated
- The trend of the defects is shown below

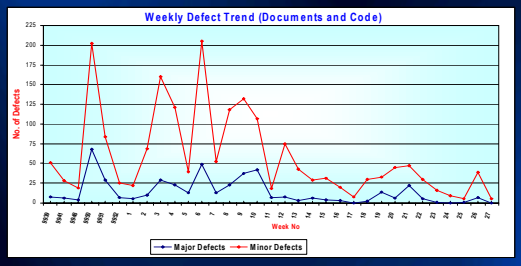


Comparison of Defects across the 2 round of analysis

Findings of the Comparison

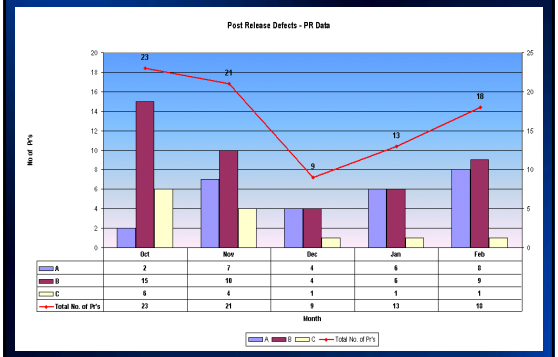
- In the category
  - Missing - Assignment/Initialization, the percentage has come down from 23.16% to 2.56%
- In the category
  - Extraneous - Assignment/Initialization, the percentage has come down from 10.53% to 0%
- In the category
  - Missing - Checking, the percentage has come down from 7.37% to 0%

Trend of defects - Both Phases



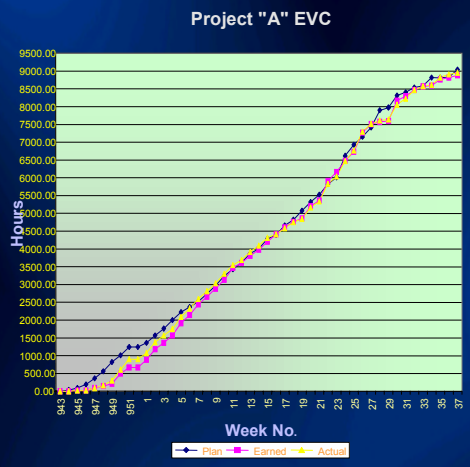
- The defect trend - major and minor defects separately
- Tracked by SQE

Post Release Defect Trend



- Post Release phase defects includes
  - Integration, Verification and Acceptance test defects
  - Tests done at the customer site
  - Total code size 28 KLOC

Project "A" Schedule



Causes Category (1)

- Defects due to **Communication** errors result from a breakdown in communication between groups or among team members. For example, A design concept stated by a higher level designer is misinterpreted by a lower level designer
- Defects due to **Education** errors: These occur due to a team member's failure to understand something causes the error. Educational errors can be further divided into the following:
  - New Function: The programmer does not understand the function and makes an error
  - Old Function: The base code or function is not well understood, and when a new function is added to it, the implementation causes the problem. (i.e. the programmer does not understand the base code or function well enough to know that the addition of new function causes regression problem
  - Other: The programmer needs education in a subject other than the function being developed. (e.g., compiler knowledge)

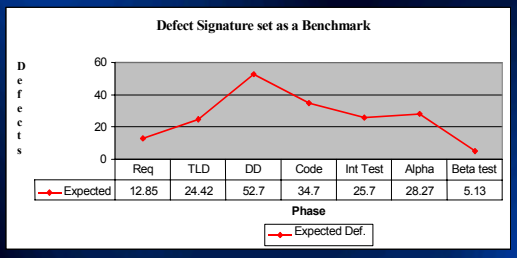
Causes Category (2)

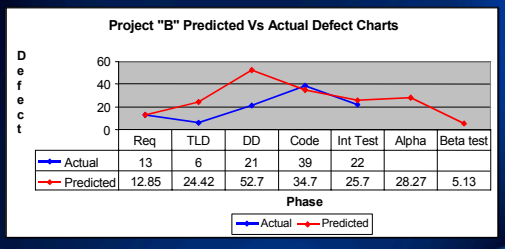
- Defects due to An **Oversight** : These arise when all the possible cases or conditions are not considered or handled. (e.g., an error condition is missed)
- Defects due to **Transcription** error: These occur when the programmer knows every thing in detail about a program, but simply makes a mistake. (e.g., types in the wrong label)

Intro to Project “B”

- Software for the (DVDv3S) DVD Video Player has to be developed
- The aim is at creating the Step DVD Philips product that will incorporate the additional features over the DVDv2B+ software stack such as
  - DTS
  - New Keys for Bit Rate Indicator
  - Timesearch
  - Disc Lock on the Remote Control
  - Late Resume Functionality
  - FTD Dimmer
  - Jog Shuttle on the Front Panel

Phase 1 - Analysis





Further Action

To Measure effectiveness of DP practices at the project level and also organization level by using

- Cost of Non-quality %: (Person-hours for review rework for all life-cycle stages till last completed stage + Person-hours for regression testing + Person-hours for PR solving + Person-hours in rework for all life cycle stages till last completed stage) * 100/ (Project effort in person-hours).
- Number of testing defects against the number of review defects
- Number of Testing defects * 100 / Total number of Pre Release Defects

Any Questions ?

Thank You

Our Experiences in Defect Reduction using Orthogonal Defect Classification Methodology

Manjula Madan, Software Quality Engineer
Philips Software Centre,
Philips Innovation Campus,
No. 1 & 2, Murphy Road, Ulsoor,
Bangalore, India
manjula.madan@philips.com
Phone: +91-80-5579000
Fax: +91-80-5561280

Abstract

In today's software development environment, the demanding compromise over functionality, time to market, and quality, drives all business decisions. The success of a software development effort is dependent on whether the development team can efficiently design, code, test and support the software in a timely fashion. This article describes an objective and time-tested method to meet the needs of all the key people in a software organization: the project manager, quality assurance manager, developers, and testers. The method is based on the application of software defects as a diagnostic probe in an organization and the capture of semantic information from the defect analysis via the "Orthogonal Defect Classification" methodology.

Although some approaches to quality improvements involve exhaustive defect classification schemes or complex mathematical models, the approach that I present relies on basic techniques that can be implemented readily by the typical software organization. This approach has been tried and used successfully across many projects in various Product Divisions in Philips Software Centre, Bangalore.

This paper describes Orthogonal Defect Classification, a means by which defects can be used to provide feedback on the development process. A careful selection of classification codes with orthogonal properties provides signatures in the distribution of the codes. These signatures reflect the progress of the process, detect departures when they occur, and provide the necessary insight to make adjustments. The paper describes these attributes and illustrates their use with results from pilot studies in many projects in Philips Software Centre, Bangalore. It is noted that Orthogonal Defect Classification has the merit of being independent of product, thereby providing a framework for general use.

Also this paper provides the overview, motivation, and benefits of using Orthogonal Defect Classification (ODC), for software process measurement and defect causal analysis. ODC provides a significant step forward in being able to understand the dynamics of software development by using classification of defects, so that they provide measurements.

About the Author

Manjula Madan, is working at Philips Software Centre, Bangalore, in the capacity of a Software Quality Engineer from the past 1-year. She has a total of 7 years experience in the IT Industry in which 3 years is in the areas of Quality Control and Quality Assurance. Previous to Philips she was working as a Software Quality Analyst at IBM Global Services India Limited. She is a Certified Quality Analyst (CQA) and Bronze Medallist in the Programming Exams conducted by National Computer Education, United Kingdom. This is the first time that Manjula is presenting a paper in a Conference. Her interests are in the areas of Quality Control and Quality Assurance.

Our Experiences in Defect Reduction using Orthogonal Defect Classification Methodology

Manjula Madan,
Philips Software Centre, Bangalore, India
manjula.madan@philips.com

" Most software professionals spend much of their working lives reacting to defects. They know that each individual defect can be fixed but that its near twin will happen again, and again, and again...."

Watts S. Humphrey

Introduction

Traditionally, defects represent the undesirable aspects of software quality. Finding and fixing defects accounts for much of the software development and maintenance cost. When one includes the costs of inspections, testing and rework, as, much as half or more of the typical development cost is spent in detecting and removing defects. More so, the process of fixing defects is even more error-prone than original software creation. Thus with a low quality process, the error rate spiral will continue to escalate. Prevention of defects is crucial to the software process. The defect prevention process is not itself a software development process. Rather it is a process to continually improve the development process.

Defect Prevention forms the essence of Total Quality Management. The Software Engineering Institute has identified Defect Prevention as a key process area in level 5 of the Capability Maturity Model (CMM). The Defect Prevention Process (DPP) was modeled on techniques used in Japan for decades and is in agreement with Deming's principles. It is based on three simple steps:

- Analyze existing defects or errors to trace the root causes
- Suggest preventive actions to eliminate the defect root causes
- Implement the preventive actions

Way of Working at PSC

The Defect Prevention (DP) process at Philips Software Centre, Bangalore (PSC) works in the following mode. There is a Defect Prevention Committee, which collects data from the projects and stores it in a central repository for further analysis. At the project level the analysis is done by the quality engineer and this is consolidated by the Quality Leader at the Line of Business (LoBs) level and sent to the Defect Prevention Committee.

Purpose

The activities performed by this committee are

- Facilitate documentation of the DP data and tracking across the teams coordinating defect prevention activities
- Presentation of LoB trends by the LoB representatives to the Committee if consultation or advice is required
- Provide Consultation on the resulting defect trends. The results of the Defect Prevention activities are reviewed to ensure the effectiveness of those activities
- Facilitate Audits for
  - Defects captured,
  - Classification of defects and Causal Analysis
  - Defect Prevention activities carried out within the projects and LoBs
- Facilitate Learning's across organization of the Defect Prevention activities carried out

- Within projects
- Within LOB's
- Facilitate feedback on the status and results of the organization's and project's defect prevention activities on a periodic basis

- Facilitate Systemic Corrections and updations/revisions of the Defect Prevention Process or any other organization's standard software processes resulting from the defect prevention actions
- DP Committee is responsible for all the organization level Defect Prevention activities and the review of systematic elimination of the same
- Normalization of defect data across Projects, Lob's and PD's
- Reports to SEPG
  - Quantify the (in terms of effort and hence cost) benefits obtained as a result of the DP activities.
  - Facilitate periodic review of the defect prevention activities by the senior management
- Facilitate DP Training's
- Facilitate the availability of the necessary tools required to support defect prevention activities

The Defect Prevention Committee is " the Organization level team to coordinate defect prevention activities and to provide necessary impetus to the software process improvement". Further this team is a part of the group responsible for the organization's software process activities.

DP Committee Charter

DP Committee has set up a Charter for itself to work towards the defect prevention activity. The Committee comprises of representatives from each of the LoBs of PSC. The DP committee activities are event driven or need driven or business driven. The purpose of the Defect Prevention Committee is to identify the common cause of defects and prevent them from recurring. It is achieved by analyzing defects that were encountered in the past and taking specific actions to prevent the occurrence of those types of defects in the future. This committee reports to the SEPG (Software Engineering Process Group) on a defined periodicity.

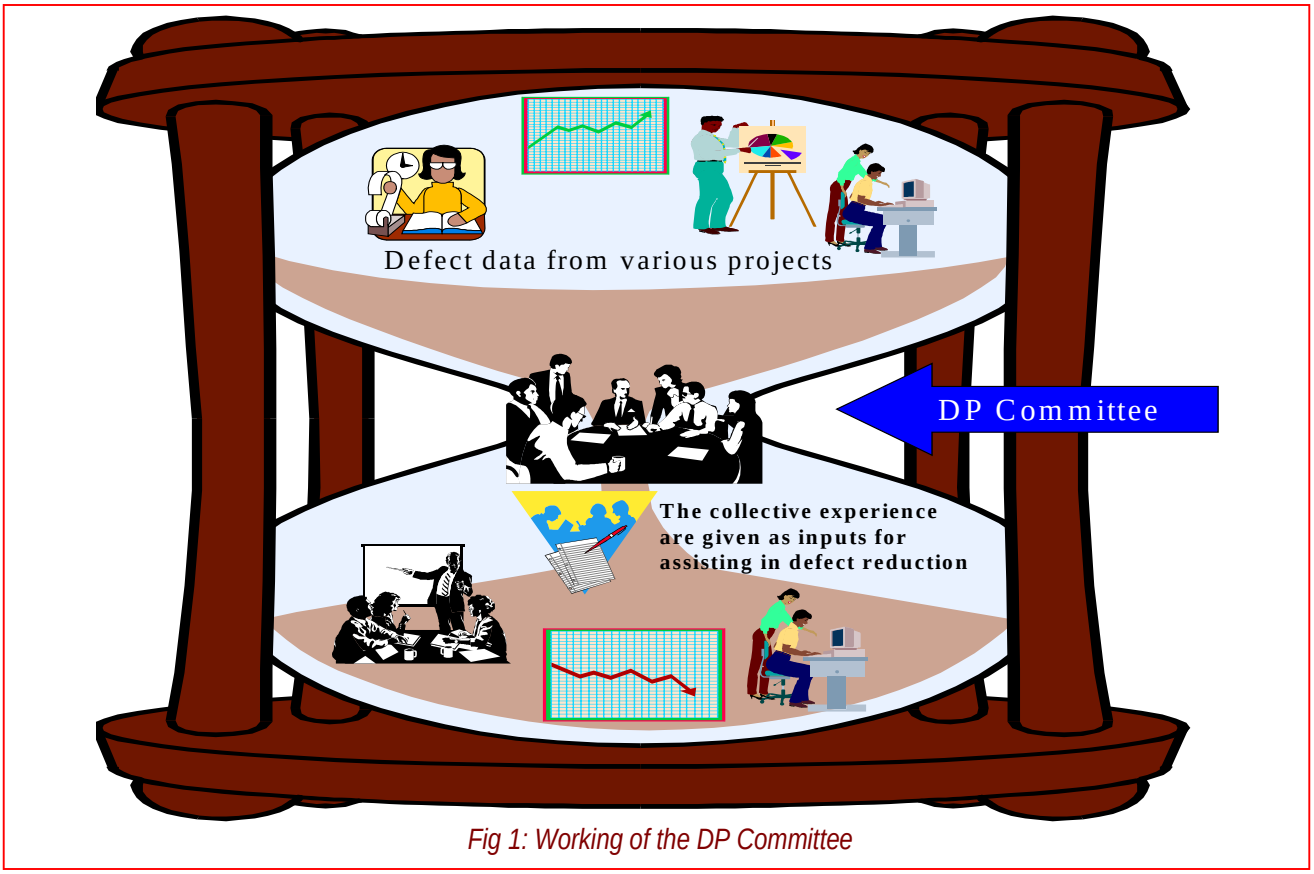


Fig 1: Working of the DP Committee

The improvement cycle involves the following:

- **Plan**
  - Prepare a plan to facilitate implementation of Defect prevention activities throughout PSC

- **Do**
  - Provide consultation and/or training on how to Plan and execute DP activities in projects and then in PD's (Product Division)
  - Use ODC methodology for defect classification.
  - Identify the areas of Improvement based on defects and common causes of defects at organizational level as reported by LOB's and PD's
  - Facilitate and Ensure dissemination of DP knowledge across PSC.
  - Facilitate sharing of best practices of Defect Prevention within and outside PSC

- **Check**
  - Review of Defect Prevention activities and Results.
  - Track Implementation/progress/trends & Causal Analysis/Root Cause Analysis

- **Act**
  - Verify/Monitor Implementation.
  - Report to SEPG based on defined periodicity.
  - Provide inputs to the Software Process Improvement Plan for continuous improvement based on the Defect Prevention activities and improvement areas identified at organization level

The Process

For successful implementation of a defect prevention program in a organization, the following system should be in place: the right process, and some means of measuring the effectiveness of this process. But measurement alone is not sufficient. Analysis, feedback, and suitable action to improve the process on the basis of this feedback should follow measurement. The defect prevention model depicted in the fig. 2 is a self-correcting closed loop system designed for continuous improvement. Defect prevention is achieved not by correcting the product, but by correcting the process that produces this product. For each phase of this life cycle model, entry and exit criteria should be clearly defined and documented.

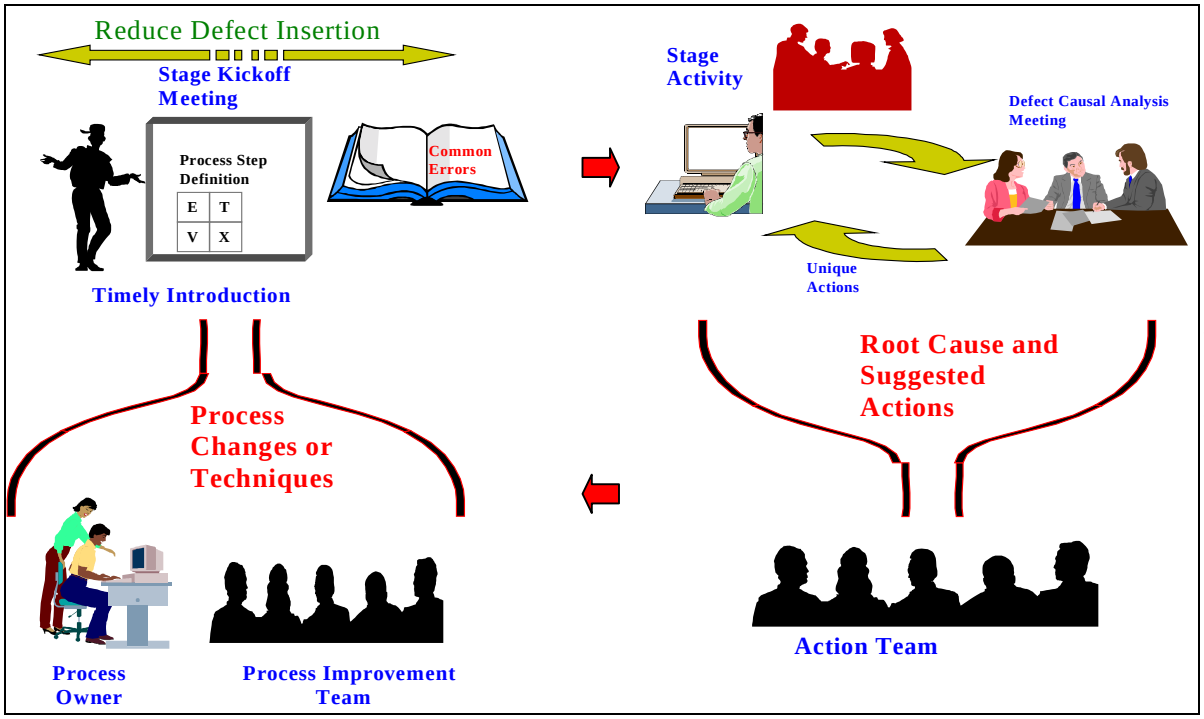


Fig 2: . Self-correcting closed loop system

Root Cause Analysis (RCA) and Statistical Analysis have played useful roles in the analysis of software defects. Effective RCA, while yielding exhaustive details on each defect, takes substantial investment of resources for completion and points to too many actions as a result. Root Cause Analysis looks at each individual defect through a magnifying glass. It is very valuable in terms of understanding Cause and Effect relationships in the context of a single defect. Given the typical workload of a developer or tester, there are not enough resources in an organization to perform causal analysis on all or even just the important defects. In addition, you end up in identifying too many actions and as a result you end up looking at the forest from the tree level. The relative prioritization among the choice of actions becomes a significant part of the process to implement solutions.

Statistical Analysis, on the other hand, provides an easy way to monitor trends, but is not capable of suggesting corrective actions due to the inadequate capture of the semantics behind the defects.

The industry experience with causal analysis has led us to believe it is a very powerful method for making dramatic reductions in defect numbers, thus reducing both overall project's costs and cycle time. Causal analysis can positively engage developers in the quest for continuous improvement. This improvement benefits each individual who learns how to do his or her job better and the team, which wants to deliver quality products cost effectively.

Orthogonal Defect Classification

A Concept for In-Process Measurements[3]

ODC is a measurement concept for software development. It gets its name since it uses the defect stream as a source of information on the product and the development process. The keyword is measurement - ODC brings a quantitative method useful for product management, productivity analysis, quality control and cost management. ODC gives an alternative to provide rapid root cause analysis with prioritization with minimal resources. In most cases analysis exploiting the eight ODC attributes and the other useful information typically captured by an organization, such as Open and Closed dates, Component, Subsystem, Severity, etc. are more than adequate to drive and monitor change.

Invented by Ram Chillarege at IBM Research circa '89, it has grown over the years, at the Center for Software Engineering, and is now also a subject of active research at a few universities. The concept is in practice in a major way at IBM, and several software companies such as Motorola and Bellcore.

ODC brings a scientific approach to measurements in a difficult area that otherwise can easily become adhoc. It also provides an in-process measurement paradigm for extracting key information from defects and enables the metering of cause-effect-relationships. Specifically, the choice of a set of orthogonal classes, mapped over the space of development or verification, can help developers by providing feedback on the progress of their software development methods.

ODC essentially means that we categorize a defect into classes that collectively point to the part of the process which needs attention, much like characterizing a point in a Cartesian system of orthogonal axes by its (x, y, z) coordinates. Although activities are broadly divided into design, code, and test, in the software development process, each organization can have its variations. It is also the case that the process stages in several instances may overlap while different releases may be developed in parallel. Process stages can be carried out by different people and sometimes by different organizations. Therefore, for classification to be widely applicable, the classification scheme must have consistency between the stages. Without consistency it is almost impossible to look at trends across stages. Ideally, the classification should also be quite independent of the specifics of a product or organization. If the classification is both consistent across phases and independent of the product, it tends to be fairly process invariant and can eventually yield relationships and models that are very useful. Thus, a good measurement system, which allows learning from experience and provides a means of communicating experiences between projects has at least three requirements:

- orthogonality,
- consistency across phases, and
- uniformity across products

One of the pitfalls in classifying defects is that it is a human process, and is subject to the usual problems of human error, confusion, and a general distaste if the use of the data is not well understood. However, each of these concerns can be handled if the classification process is simple, with little room for confusion or possibility of mistakes, and if the data can be easily interpreted. If the number of classes is small, there is a greater chance that the human mind can accurately resolve between them. Having a small set to choose from makes classification easier and less error prone. When orthogonal, the choices should also be uniquely identified and easily classified.

ODC makes it possible to push the understanding and use of defects well beyond quality. In the ODC Method each of the defects is classified on the basis of 8 angles, which are:

1. ***Activity***
2. ***Trigger***
3. ***Defect Target***
4. ***Defect Type***
5. ***Impact***
6. ***Source***
7. ***Defect Qualifier***
8. ***Age***

Deploying ODC at Philips Software, Bangalore

At Philips Software Centre, Bangalore (PSC) for classifying defects, ODC was found to be an ample way. Of the 8 angles of ODC, 3 were considered and implemented. They were Defect Type, Defect Qualifier and Activity.

The collective experiences of the past projects executed at PSC using the above 3 angles were considered and the decision from the DP Committee was to continue with the same. The other rationale was also to have fewer data points to start working on and show improvements. This implementation happened at a crucial stage of the continuance of PSC, where in the company was working toward the SEI CMM Level 5 assessment.

In the 2 case studies given below, it can be learned that ODC methodology can be used as a purport means for classification of defects.

Case Study 1:

- **Project "A" Introduction**

Project "A" is a multi-site project with 4 sites locations, aimed at developing a DVD Player with Recorder. It is conceived to have the Player functionality inherited from the Philips DVD Player and the Recording functionality inherited from the Philips VCR. This project was divided into the 5 sub systems. The Bangalore team is responsible for the entire front controller, parts of RCS, and also assists customer with User Interface, Audio/Video Switching and P50 software. The Bangalore team is also responsible for bug fixing in the entire RCS and FRC software. The team was actively involved in the Requirements Analysis and the Top Level Design at the overall Project Level.

- **Phases of the Project "A"**
  - Component Design
  - Implementation
  - PR solving (current phase)

During the initial Implementation phase, the project team at Bangalore started classifying the defects on the basis of defect type and defect qualifier. This classification methodology was agreed upon so that the causal analysis could be done, and preventive actions could be arrived at for identifying assignable causes and addressing them. The specific preventive actions that would be arrived at could be used in the same phase of the project that had to be completed in this project and also future projects wherever applicable.

- **Phase 1 - Analysis**

Orientation on ODC was given to the team so that they could have an understanding of the usefulness of the classification methodology and also be able to implement it.

Defect Type	Defect Qualifier	Activity
Assignment/Initialization *Checking* *Algorithm/Method* *Function/Method* *Timing/Serialization* *Interface/O-O Messages* *Relationship*	*Missing* *Incorrect* *Extraneous*	*Component Design – Detailed Design Review* *Implementation- Code Review*

- The majority of the defects were found to occur due to 3 defect types and 2 Defect Qualifier types as listed below
- In the classification based on **Defect Type**, about 98% of the defects were categorized under

Defect Type	% of Defects
Assignment/Initialization	61
Algorithm/Method	23
Checking	14

- In the classification based on Defect Qualifier, about 88% of the defects were categorized under

Defect Qualifier	% of Defects
Incorrect	54
Missing	34

- **Identification of Causes and Preventive Action**

Defect analysis data was shared with the project team. The team brainstormed on the causes and the Corrective and Preventive actions was arrived at. These actions were put into implementation immediately. The details of it is given below:

- **Assignment/Initialization:**

Cause :

Most of the defects were found to be due to oversight in re-initializing

Preventive Action :

i. For the defects associated with this defect type, it was decided that the code review and design review checklist should be updated to check for re-initialization
ii. Also the usage of flags shall be avoided during debugging
iii. The Design Guidelines shall also be updated with respect to this

22.1	Update Design and Code Review Checklist	
Assigned	Info:	Re-initialization to be checked during the Design/Code review
	Action:	Manjula

22.2	Update Design Guidelines	
Assigned	Info:	Raise a Quality System Change Request on the Design Guidelines to include checking for re-initialization
	Action:	Manjula

22.3	Re-initializing of Flags to be avoided during debugging	
Ongoing	Info:	Re-initializing of Flags which is inserted during debugging to be checked and removed
	Action:	All Developers

- **Algorithm/Method :**

  Cause:

  Most of the defects were due to scattered inputs and re-use of the code.

  Preventive Action :

i. For scattered inputs and re-used code, have a Design Kick-off meeting
ii. Understand the functionality of the re-used code
iii. Have a kick-off meeting with the Architect and Requirements Engineer before the design, especially when the requirements are close to hardware
iv. Training to be provided
v. Update the Design Guidelines

22.4	Kick-off meeting to be held before Design of components with scattered inputs or design using re-used code	
Ongoing	Info:	Have a kick-off meeting with the Architect and Requirements Engineer before the design, especially when the requirements are close to hardware
	Action:	PL, TL

22.5	Functionality of the re-used code to be understood	
Ongoing	Info:	While estimating this activity has to be included and the Estimation Checklist to have an item to accommodate the study phase of the Functionality
	Action:	All developers

- **Checking :**

  Cause:

  Most of the defects were due to oversight

  Preventive Action :

  Update the review Checklist

22.6	Update Design and Code Review Checklist	
Assigned	Info:	To check for the incorrect or missing validation of parameters or data in conditional statements
	Action:	Manjula

- **Incorrect:**

  Cause:

  Most of the defects were found to be due to oversight

Preventive Action :

Update code review checklist and design review checklist

22.7	Update code review checklist to check for omission	
Assigned	Info:	In correct qualifier
	Action:	Manjula

- **Missing:**

Cause:

Most of the defects were found to be due to oversight

Preventive Action :

Update code review checklist and design review checklist

22.7	Update code review checklist to check for omission	
Assigned	Info:	Missing qualifier
	Action:	Manjula

- **Phase 2 - Analysis**

At the end of the Implementation Phase, the similar activity was done of classifying the defects and following was observed. This is shown in Fig 3. Improvements were observed and the team attributed this to the corrective and preventive actions put into place.

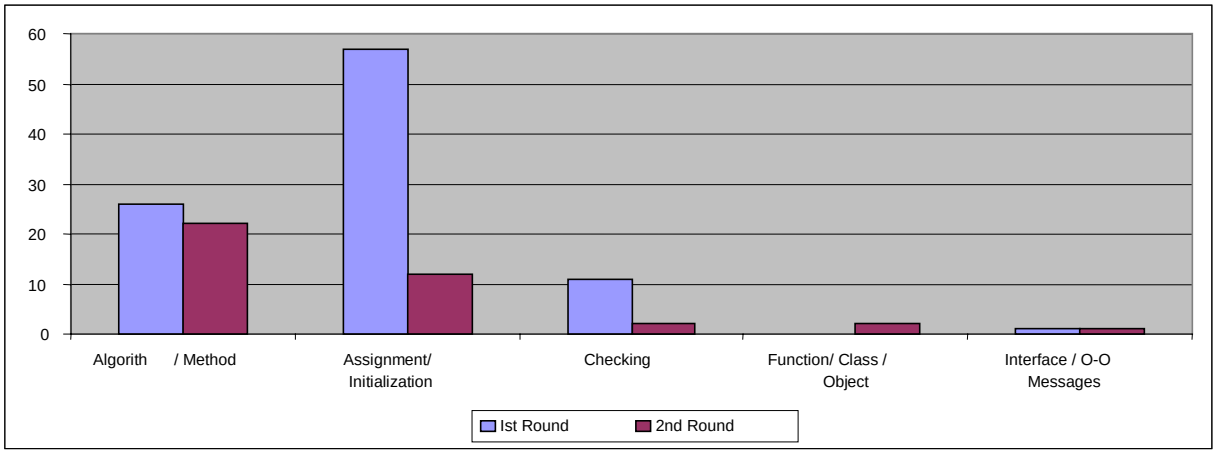


Fig 3: Comparison of Defects across the 2 round of analysis

The findings of the comparison were

- In the category - **Missing - Assignment/Initialization**, the percentage has come down from 23.16% to 2.56%.
- In the category - **Extraneous - Assignment/Initialization**, the percentage has come down from 10.53% to 0%.
- In the category - **Missing - Checking**, the percentage has come down from 7.37% to 0%

The project team has been attributed the above improvements to corrective and preventive actions that were built into the system. Also at the end of the Implementation phase, the trend of the defects which were being tracked by the SQE on a weekly basis saw a decline as shown in the figure 4.

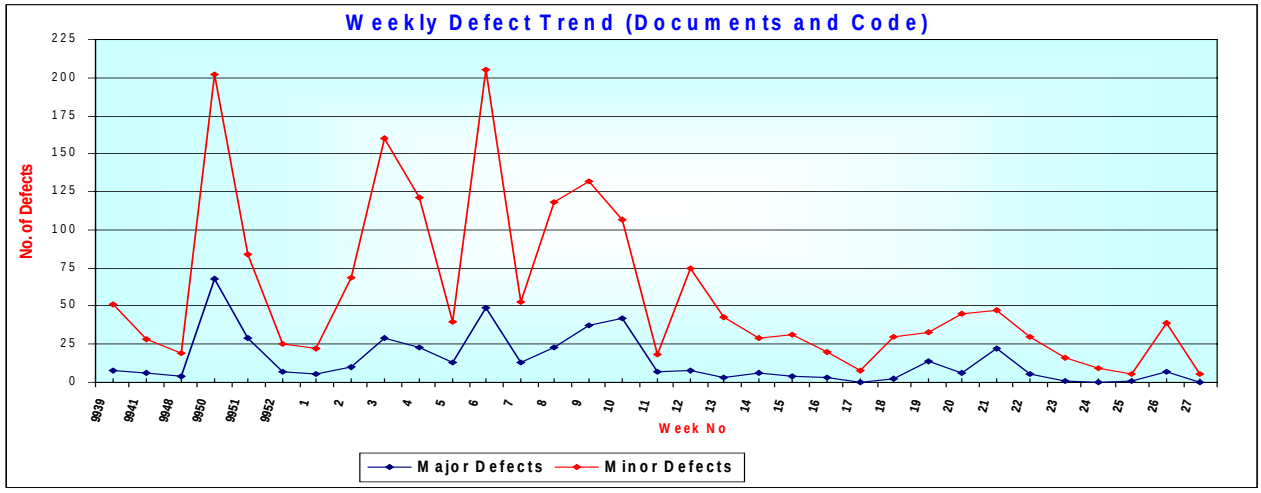


Fig 4: The trend of the defects observed weekly

- **Detailed Causal Analysis and Preventive Action**

Defect analysis data was again shared with the project team at PSC, Bangalore. The Preventive actions that were suggested by the project team "A" had been logged in the project database for future projects. The team felt it necessary to do a complete causal analysis of the defects found in both the phases of the project. This succored to arrive at preventive actions, so that they could be implemented in the projects of the same Lob and also in the other projects in the same PD.

Positive Mien on the Project

The post release phase of the project "A" includes integration, verification and acceptance testing. All these are done at the customer site. For the total code size of the project, which is 28 KLOC and the post release defect as of February 2001 is 87, this is excellent progress. The trend of the post release defects is shown in Fig 5.

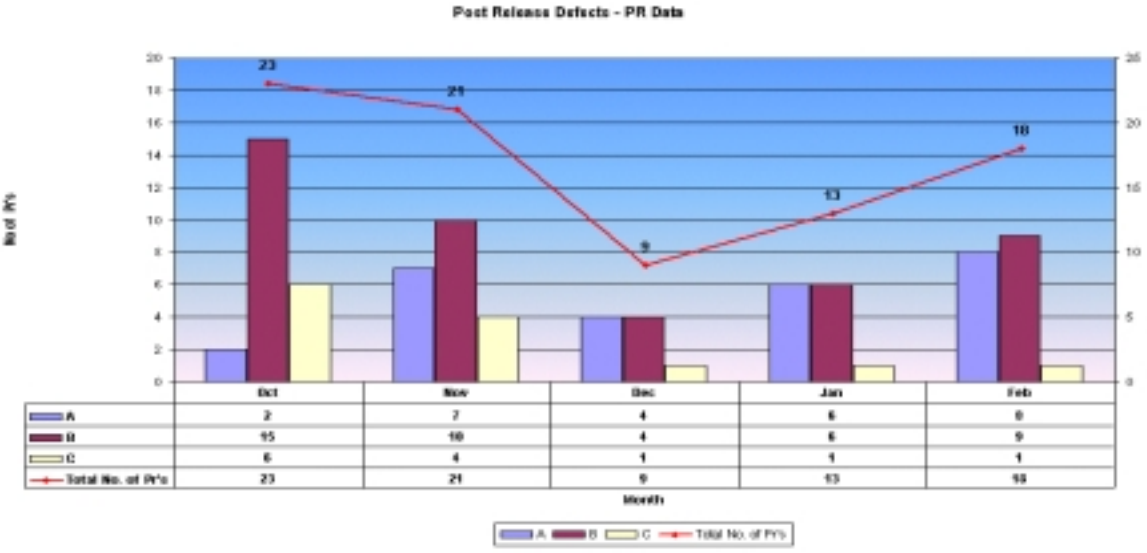


Fig 5: Post Release Defect Trend

Also another positive aspect of the project is that the schedule was adhered to and out of the 4 sites, PSC was the only team to finish right on target. The Earned Value Chart of the project is given below in figure 6 for reference. These factors influenced the Customer in giving good ratings to the project in the quarterly 'Customer Satisfaction' feedback.

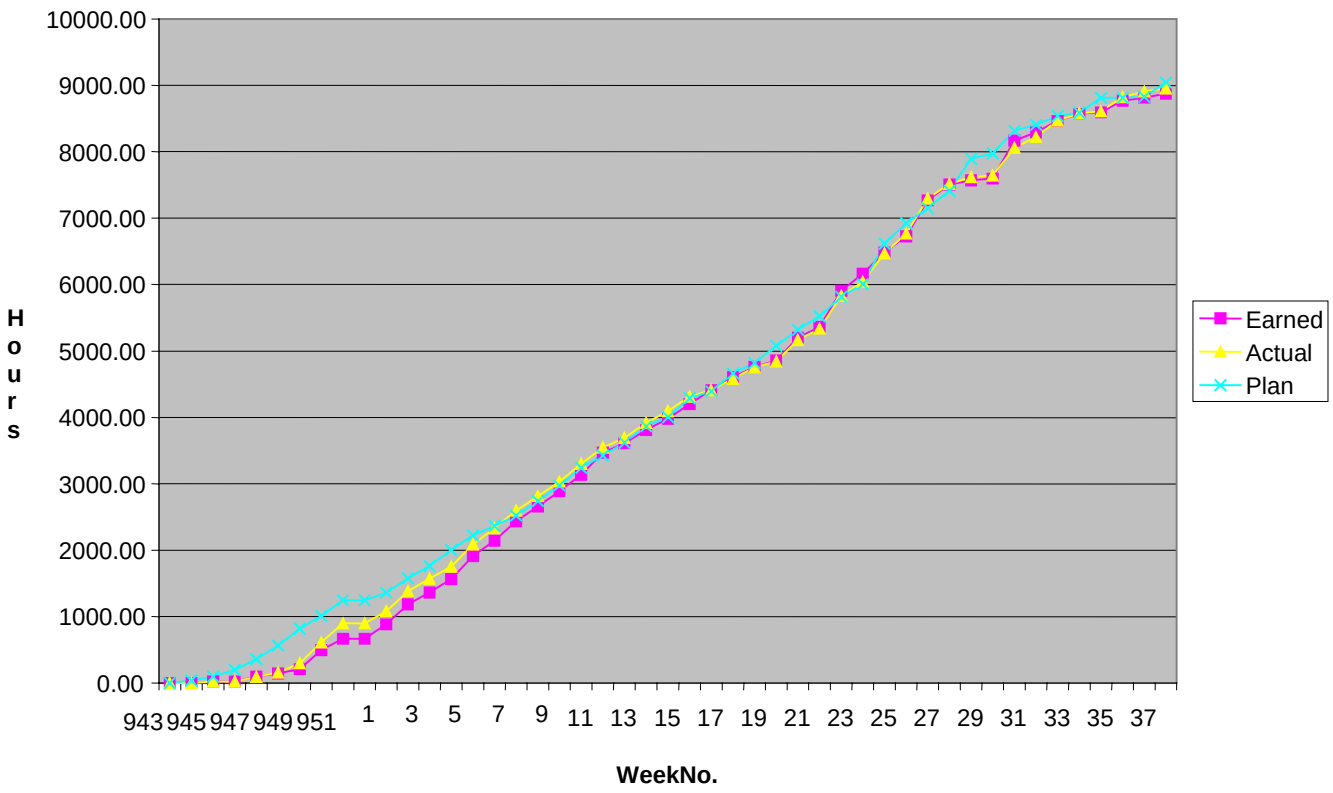


Fig 6: Earned Value Chart of Project "A"

Further Momentum in the Defect Reduction path at PSC

After further progress, an Assessment (CBA IPI) was made and PSC was found to be performing at the Optimizing Level. During this period Defect Prevention Committee decided to go in for the usage of the Cause categories as cited by Watts Humphrey[1].

- **Causes Category**

The causes broadly falling into various categories are discussed below.

- Defects due to **Communication** errors result from a breakdown in communication between groups or among team members. For example, A design concept stated by a higher level designer is misinterpreted by a lower level designer

- Defects due to **Education** errors: These occur due to a team member's failure to understand something causes the error. Educational errors can be further divided into the following:

  - **New Function**: The programmer does not understand the function and makes an error
  - **Old Function**: The base code or function is not well understood, and when a new function is added to it, the implementation causes the problem. (i.e. the programmer does not understand the base code or function well enough to know that the addition of new function causes regression problem
  - **Other**: The programmer needs education in a subject other than the function being developed. (e.g., compiler knowledge)

- Defects due to An **Oversight** : These arise when all the possible cases or conditions are not considered or handled. (e.g., an error condition is missed)

- Defects due to **Transcription** error: These occur when the programmer knows every thing in detail about a program, but simply makes a mistake. (e.g., types in the wrong label)

- **Some of the defect prevention techniques proposed**
  - **Pre-review & Review Meetings**
    Before the beginning of each increment or stage in the project life cycle as a part of the previous stage review a preview is held to discuss the problems anticipated and possible prevention. Review meetings also help reduce defects though the immediate focus of reviews is defect detection & not prevention.

  - **Updation of Checklists based on improvements planned**
    Checklists are tools for defect prevention when these are prepared before the activity based on improvement planned.

  - **Concurrent coding and unit testing**
    Instead of waiting until the entire coding is completed, unit testing can be done in parallel with coding. If each subprogram is tested as soon as it is coded, then the analysis of errors found in these subprograms can be used to prevent their re-occurrence in the remaining portion of the code.

  - **Prototyping**
    Prototyping helps create a scaled down model of a real-life scenario with the intention of identifying most of the major issues or problem areas are identified upfront. Once these issues are identified, suitable measures could be taken to ensure that similar defects are prevented in future. This model can be shown and sometimes used by the customer, to confirm that the requirements have been fully understood, by both the parties, and the finished product is going to be correct, especially in terms of functionality. Even though prototypes can take time to develop, and then therefore sometimes prove to be expensive way of developing the system, but when the time taken to review and debug the finished product is taken into consideration, big savings can be done.

  - **Reuse**
    Reuse enables usage of code proven to work in one application since it can be safely used in other applications without the fear of inherent bugs.

Another typical example in which the defect reduction was observed in the similar fashion is described in the second Case Study. This project is taken from the same Product Division but from a different Line of Business.

Case Study 2:

- **Project "B" Introduction**

The detail of Project "B" is the software for the (DVDv3S) DVD Video Player was to be developed. The aim is at creating the Step DVD Philips product that will incorporate the additional features over the DVDv2B+ software stack such as

DTS
New Keys for Bit Rate Indicator
Timesearch
Disc Lock on the Remote Control
Late Resume Functionality
FTD Dimmer
Jog Shuttle on the Front Panel.

- **Phases of the Project "B"**

  - Complete "V" Model Life Cycle (project currently in progress)

- **Phase 1 - Analysis**

In the case of this project, before the start of the project, there was a similar project using the same stack for which the defects had been classified as per the ODC Methodology customized to PSC way of working. So the defect signature was available to the Project "B" to benchmark against. This is depicted is Figure 7.

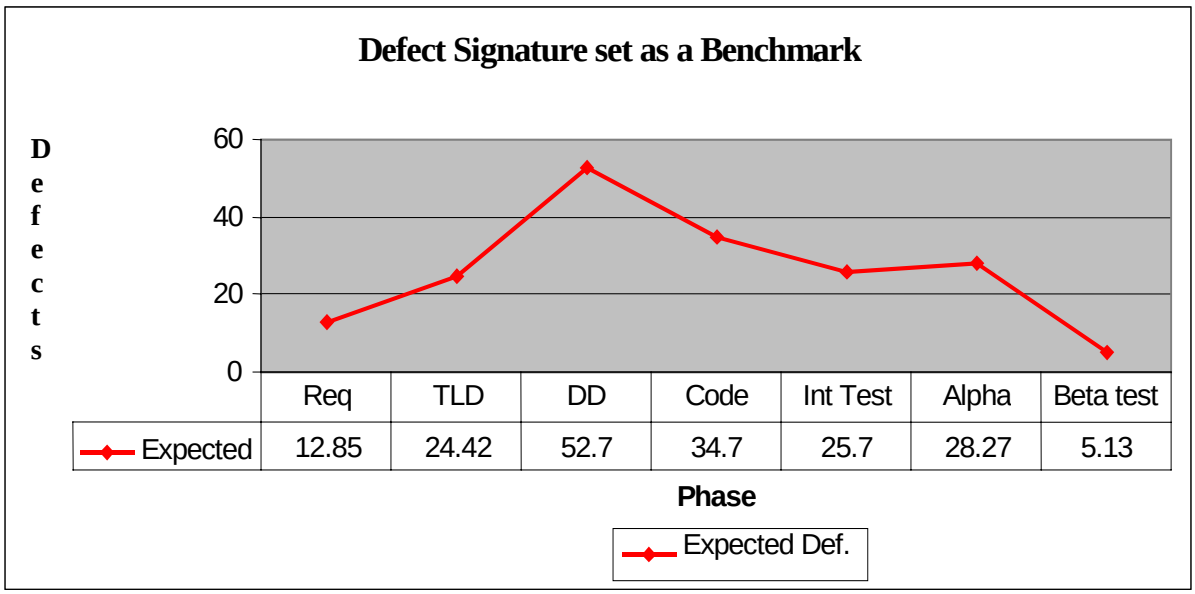


Fig 7: Defect Signature of the earlier project

Causal Analysis was done at the end of the above project life cycle and the measures to be taken to reduce the number of defects were documented.

- **Phase 2 - Analysis**

At the current phase of the Project "B", the figure 8 shows the improvement in the defect reduction maneuver.

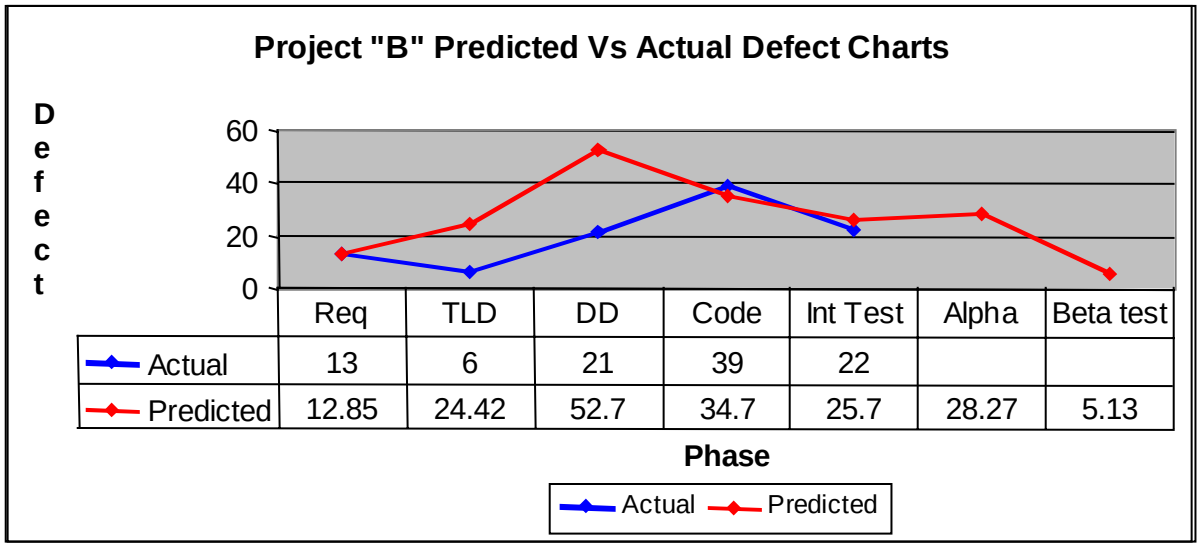


Fig 8: Comparison of the Predicted and Actual

Further Action:

Currently we, at PSC are working at measuring the effectiveness of defect prevention process by using the following measures. This is being piloted in the various projects to measure effectiveness of DP practices at the project level and also organization level.

- **Cost of Non-quality %:** (Person-hours for review rework for all life-cycle stages till last completed stage + Person-hours for regression testing + Person-hours for PR solving + Person-hours in rework for all life cycle stages till last completed stage) * 100/ (Project effort in person-hours).

- Number of testing defects against the number of review defects

- Number of Testing defects * 100 / Total number of Pre Release Defects

Summary

Orthogonal Defect Classification (ODC) provides a good framework for cause-effect analysis. In order to implement this defect classification framework, one still has to come up with:

- Effective attributes to be measures
- Process for analyzing attributes
- Action plan based on the analysis result for process improvement

Action plan is independent of ODC. However, action plan is required for process improvement. It would be good if benchmark for applying ODC is available. This will help in applying and analysing ODC results. This can be achieved only by accumulating data over a period of time.

The fundamental objective of defect prevention is to make sure that errors, once identified and addressed, do not occur again. One or two people cannot do defect prevention, and it cannot be done sporadically. Everyone must participate by faithfully executing the process – almost as if his or her lives depended on it. As with any other skill, it takes time to learn defect prevention well, but if everyone on the project participates, it can transform an organization. The bottom line of the Organization should be as shown in Figure 9.

Fig 9: Software without Defects is "Bulls Eye"

References

1. ***Managing the Software Process***
By Watts S Humphrey

2. ***Metrics and Models in Software Quality Engineering***
By Stephen H Kan

3. ***Orthogonal Defect Classification***
By Chillarege, I S Bhandari, M J Halliday, J K Chaar

4. ***ORTHOGONAL DEFECT CLASSIFICATION - A BREAKTHROUGH FOR IN-PROCESS MEASUREMENT***
By Ram Chillarege - A Tutorial - Fifth International Symposium on Software Reliability Engineering, Monterey, California, November 6-9, 1994

5. ***Project "A" and "B" database***
Consumer Electronics, Philips Software Centre, Bangalore

6. ***Defect Prevention Charter***
Philips Software Centre, Bangalore

QW2001 Paper 3T2

Dr. Mark R. Blackburn, Mr. Robert Busser, Mr. Aaron Nauman
& Dr. Ramaswamy Chandramouli
(Software Productivity Consortium)

Model-based Approach To Security Test Automation

Key Points

- Test Automation Technology and Experience
- Security Functional Testing
- Test Engineering using Model

Presentation Abstract

This paper describes the objective of the security functional testing initiative and the approach applied. It provides both a process perspective describing the roles of developers and tool automation to reduce significant manual effort from the traditional security testing process. It illustrates the development of a model for test automation using a small set of security specifications that deal with “Granting Object Privilege Capability” in the Common Criteria Security Target Document for an Oracle Database Server. It provides an overview of the test case generation process. It also describes the process used to generate test drivers for an SQL database engine.

To assure that audience that the underlying capabilities of model-based development and test automation can be applied to their applications, the paper and presentation will briefly summarize some of the other applications types in which this model-based approach has been used. Specifically, the approach has been applied to non-critical applications like workstation-based Java applications with GUI user interfaces, database applications, as well as critical applications like telemetry communication for heart monitors, flight navigation, guidance, autopilot logic, display systems, flight management and control laws, airborne traffic and collision avoidance while supporting automated test driver generation from standard languages (e.g., C, C++, Java, Ada, Perl, PL/I, SQL, etc.) as well as proprietary languages, COTS test injection products and test environments.

About the Author

Dr. Blackburn is a Software Productivity Consortium Fellow, President of T-VEC Technologies, Inc. and co-inventor of the T-VEC system. He has twenty years of software systems engineering experience in development, project leadership and applied research in object technology, requirement and design specification, model-based development, formal methods, and formal verification. His more recent technical activities have been focused on transforming various functional, OO, and control-system models from 3rd party tool systems into a representation

that can support requirement defect removal and test automation. He is also involved in functional security testing, developing strategies for integrating knowledge management and e-business, and has also been involved in applied research and technology demonstrations in web-based knowledge engineering, domain engineering, and reverse engineering. He has also spent over ten years in the development of real-time flight critical avionics systems. He earned a BS in Mathematics from Arizona State, MS in Mathematics from Florida Atlantic University, and a Ph.D. in Information Technology from George Mason University.

Mr. Busser is co-founder of T-VEC Technologies, Inc. and co-inventor of the T-VEC system. He has over twenty years of software systems engineering experience in development, and management in the area of advanced software engineering, and expertise in software engineering processes, methods and tools. He is the chief architect of the T-VEC system. He has extensive experience in requirement and design methods, real-time systems, model-based development and test generation tools, model analysis, and verification. He has extensive knowledge about model transformation systems, theorem prover and constraint solving systems. In addition, he has extensive avionics engineering experience and has been involved in several FAA certifications. He has experience applying this knowledge in the development of highly-reliable software systems and the development of state of the art requirements-based software modeling and testing technologies. Mr. Busser has a B.S. in Electrical and Electronics Engineering from Ohio University.

Mr. Nauman has a wide range of systems and applications development experience in both real-time (telecommunications) and information systems domains. He is currently involved in the development of model transformation, and software verification through specification-based automated testing. His experience includes all aspects of product development from requirements analysis through test implementation. Additionally, he has experience in object-oriented technologies, distributed and client/server systems, web-based and components-based software and systems integration. He is a representative on the OMG UML Action Semantics working group. Mr. Nauman graduated Summa Cum Laude from North Carolina State University with a B.S. in Computer Science.

Dr. Ramaswamy Chandramouli is a computer scientist at NIST with over 15 years of experience in both Private Sector and Federal Agencies. His professional interests include Distributed System Security, Access Control Models and Security Specifications. He was one of the authors of “Role Based Access Control Protection Profile” which was the first Common Criteria (V 2.0) Protection Profile to be certified in the U.K. He was also the lead author of the paper titled “Comparison of Role Based Access Control Features in commercial DBMS” which won the Best Professional Paper award at the the 21st National Information Systems Security Conference held at Crystal City, VA, Oct 1998. Dr.Chandramouli holds an MS degree in Operations Research from the University of Texas and a PhD in Information Technology from George Mason University.

Model-based Approach to Security Test Automation

Mark Blackburn, Robert Busser, Aaron Nauman
T-VEC Technologies/SPC

Ramaswamy Chandramouli (Mouli)
National Institute of Standards and Technology

Common Symbols, Terms and Acronyms

Common Symbols


Tool


Manual process


Machine readable artifact


Textual document


Object mapping


SCR table

Glossary of Terms and Acronyms

API	application programming interface
COTS	Commercial off-the-shelf
GUI	graphical user interface
Java	high-level programming language
JDBC	Java Database Connectivity
MCDC	Modified Condition Decision coverage
NIST	National Institute of Standards and Technology
NRL	Naval research laboratory
ODBC	Open Database Connectivity
Perl	high-level programming language
TAF	Test Automation Framework
SCR	Software Cost Reduction
SQL	Structured Query Language
SRS	system/software requirement specification
UML	Unified Modeling Language

Problem

- Software security is a software quality issue that continues to grow in importance as it impacts many aspects of every-day life
- Ubiquitous access to resources through internet-based software increases importance of security
- Shortened development and deployment cycle makes it difficult to conduct adequate security functional testing to verify expected security behavior
- Present practice, developing and performing security functional testing is costly
  - Increased demand for product variations is further increasing cost impacts on security evaluation laboratories

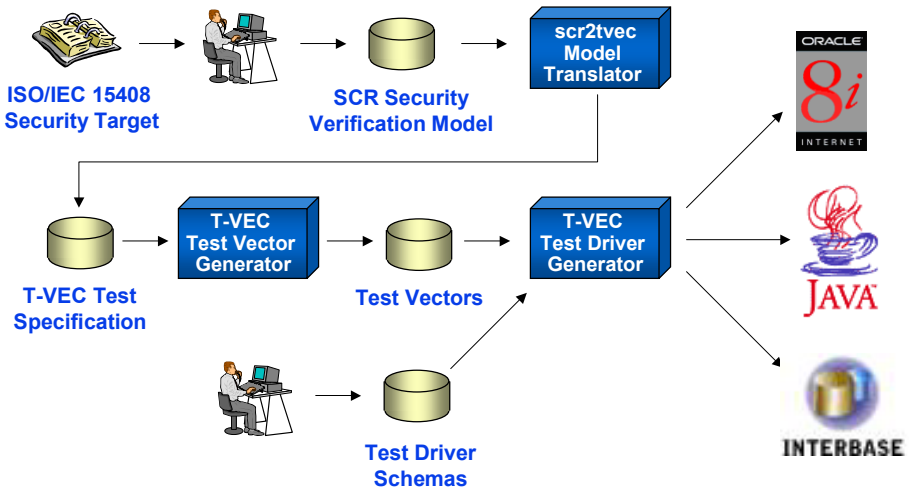
Objective

- National Institute of Standards and Technology (NIST) initiated program to develop methods and tools for automating Security Functional Testing
  - Assess applicability and cost-effectiveness of model-based approach
- Methodology based on expressing security functional requirements as a model
- Toolkit required to automate mechanisms:
  - Check specification for contradictions, requirement defects, feature interaction problem or circular definitions
  - Generate test vectors from security requirements specifications expressed as models
    - Test vector consists of test inputs, expected outputs and an association between test and specification
- Provide assurance that generated tests provide needed coverage, as well as security requirement-to-test traceability

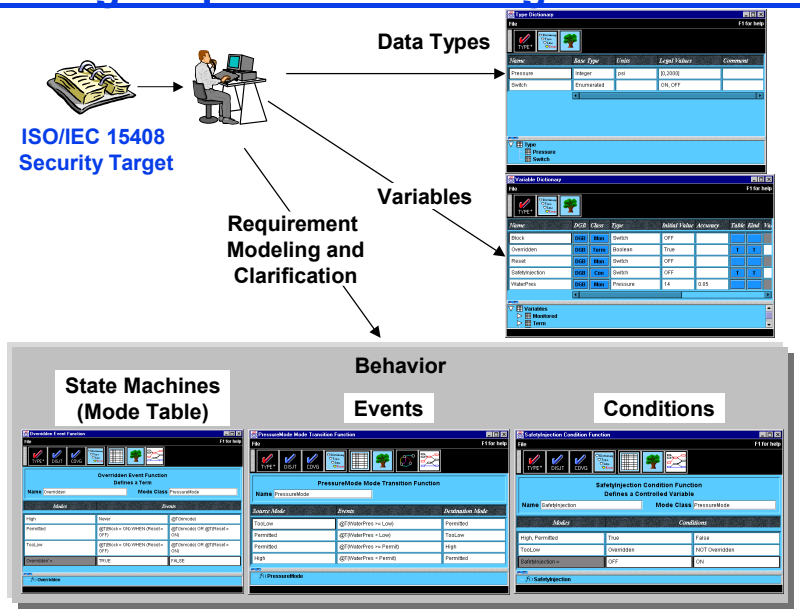
Approach

- Use model-based approach for specification and automated test generation for Security Testing
- Model set of security requirements in Software Cost Reduction (SCR) specifications using Naval Research Labs (NRL) SCRtool
- Translate model into T-VEC linear form (specification language) using model translator
- Generate test vectors from translated model
- Generate test drivers for various target environments and databases
- Execute generated test drivers against target environments and databases

Process Flow



Modeling Requirements Using the SCR Tool



ISO/IEC 15408 Security Target (Oracle Specific)

Grant Object Privilege Specification

Security Functional Requirements

Table 5: Mapping of TSFs to SFRs

	FTA		FDP						FMT										FPT	FTP	FTA		FAU							
	FTA.LIM.1	FTA.LIM.2	FDP.ACC.1	FDP.ACF.1	FDP.IFC.1	FDP.IFF.1	FDP.RIP.2	FDP.ITT.1	FMT.MSA.1	FMT.MSA.2	FMT.MSA.3	FMT.MTD.1	FMT.REV.1	FMT.SAE.1	FMT.SMF.1	FMT.SMR.1	FMT.SMR.2	FMT.SMR.3	FPT.AMT.1	FPT.FDP.1	FPT.TDC.1	FTA.MCS.1	FTA.SSL.1	FAU.GEN.1	FAU.GEN.2	FAU.SAA.1	FAU.SAR.1	FAU.SAR.2	FAU.STG.1	FAU.STG.3	FAU.STG.4
FIA.PRL		Y																												
FIA.UID		Y																Y												
FIA.OSA		Y																												
FIA.CNF				Y																										
FIA.IDE		Y																	Y	Y	Y									
FIA.CSA		Y																	Y	Y		Y								
FIA.CSN		Y																	Y			Y								
FIA.ATT	Y		Y						Y																					
FIM.CNF									Y	Y											Y									
FIM.POL																					Y	Y								
FIM.NSESS																					Y	Y								
FIM.TIME																					Y									
FIM.RSESS																					Y									
FIM.RCALL																					Y									
FDAC.OBID			Y	Y	Y																									
FDAC.OBREF			Y	Y	Y																									
FDAC.SUA			Y	Y																										
FDAC.OBA			Y	Y					Y																					
FDAC.POL				Y	Y	Y			Y	Y									Y											
FDAC.SEP																				Y	Y									
FDAC.UR							Y																							
FAPR.GOP									Y	Y																				
FAPR.MOD									Y		Y																			
FAPR.GRSP									Y	Y	Y																			
FAPR.GRR									Y	Y	Y	Y																		
FAPR.DER																		Y												
FAPR.EDR		Y																Y												
FPRI.SPRIV		Y		Y	Y	Y			Y					Y																
FPRI.XOP		Y																												
FAUD.SOM																							Y						Y	
FAUD.SEV																							Y						Y	
FAUD.ALW																							Y						Y	
FAUD.CNF				Y	Y																								Y	
FAUD.ACC									Y																Y			Y		
FAUD.DEL									Y																				Y	Y
FAUD.INF																								Y	Y					
FAUD.VIEW																											Y	Y		
FAUD.VULN																														Y

Target of Evaluation Specification of Security Functions

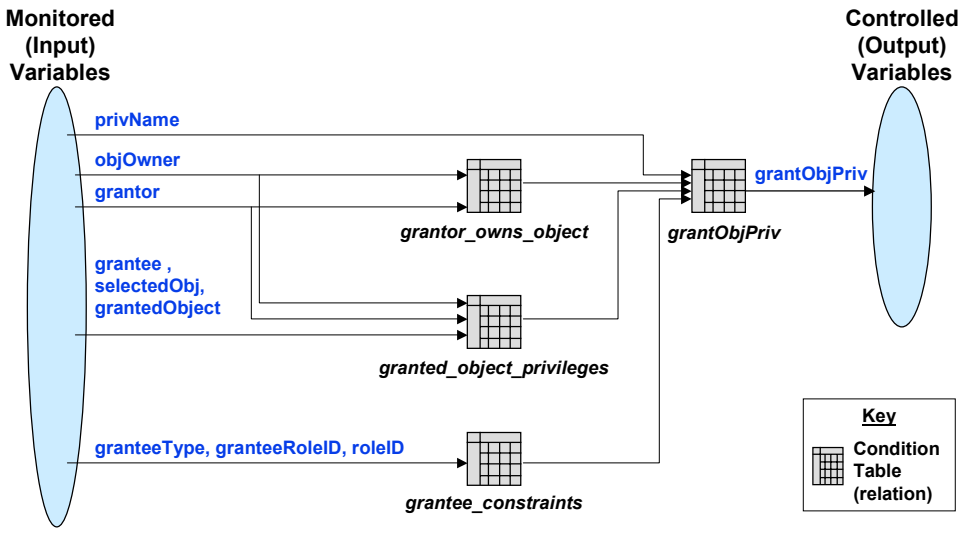
Requirements Modeled

- **Grant Object Privilege** (F.APR.GOP):
  A normal user (the grantor) can grant an object privilege to another user, role, or PUBLIC (the grantee) only if:
  a) the grantor is the owner of the object; or
  b) the grantor has been granted the object privilege with the GRANT OPTION.
- Revoke Object Privilege (F.APR.ROP) – A normal user (the revoker) can revoke an object privilege from another user, role or PUBLIC (the revokee), and any further propagation of the object privilege started by the revokee, only if the revoker is the orginal grantor of the object privilege
- Grant System Privilege (F.APR.GRSP) – A user (the grantor) can grant a system privilege to another user, role or PUBLIC (the grantee), and revoke a system privilege from the grantee, only if:
  a) the grantor (or revoker) is the DBA user; or
  b) the database session of the grantor (or revoker) has the GRANT ANY PRIVILEGE privilege effective; or
  c) the grantor (or revoker) has been granted that system privilege directly with the ADMIN OPTION.
- Grant Role (F.APR.GRP)– A user (the grantor) can grant a role to another user, role or PUBLIC (the grantee), and revoke a role from the grantee, only if:
  a) the grantor is the DBA user; or
  b) the database session of the grantor (or revoker) has the GRANT ANY ROLE privilege effective; or
  c) the grantor (or revoker) has been granted that role with the ADMIN OPTION.

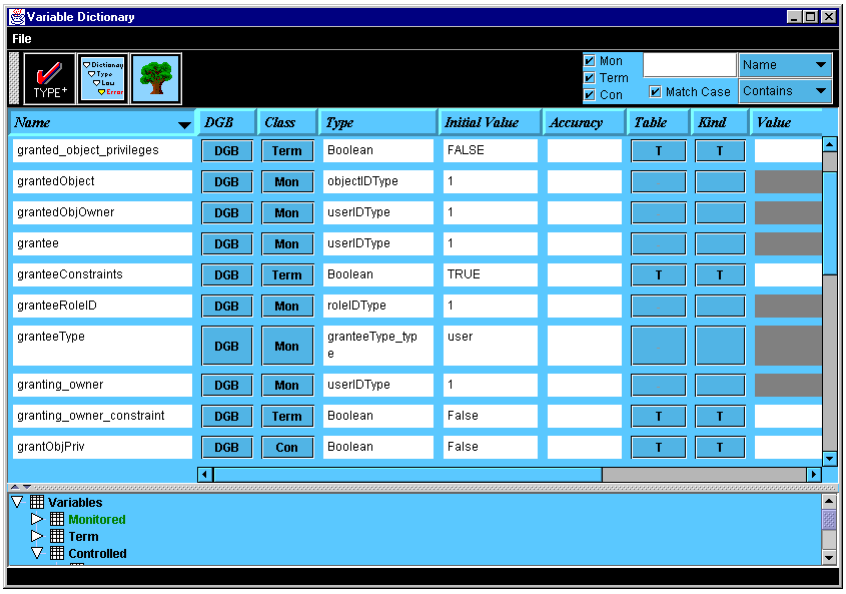
Requirement Analysis

Requirement Statement/Clause	Variables	Relations
A normal user (the grantor) can grant an object privilege to another user, role or PUBLIC (the grantee)	grantor grantee object privilege grantee type	grantee constraints (user, role or PUBLIC)
GOP (a) - a grantor can grant an object privilege to a grantee if the grantor owns the object	grantor grantee object privilege	grantor owns object
GOP (b) – a grantor (that does not own the object) can grant object privileges to the grantee if the object owner previously granted object privilege to the grantor with the GRANT OPTION	grantor grantee object privilege object owner GRANT OPTION granted object	granted object privilege

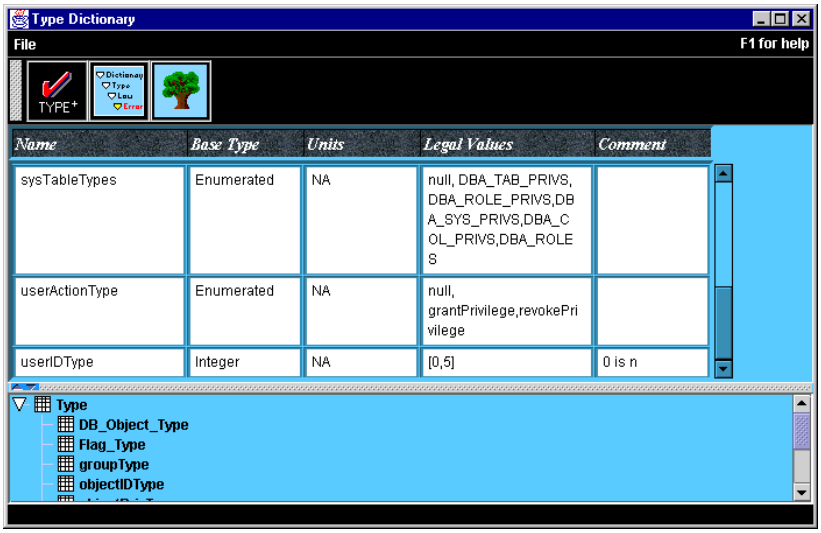
Model Structure



Defining Variables



Defining Data Types



Logic For Grant Object Privilege

Table Name	Condition		
	(grantor_owns_object	NOT(grantor_owns_object)	GOP(a)
	OR (granted_object_privileges AND grantee_constraints))	AND (NOT(granted_object_privileges) AND grantee_constraints)	GOP(b)
	AND (grantor != grantee) AND (granteeType = user OR granteeType = role OR granteeType = PUBLIC) AND (Priv_Name = ALL OR Priv_Name = UPDATE OR Priv_Name = SELECT OR Priv_Name = INSERT OR Priv_Name = DELETE)	AND (grantor != grantee) AND (granteeType = user OR granteeType = role OR granteeType = PUBLIC) AND (Priv_Name = ALL OR Priv_Name = UPDATE OR Priv_Name = SELECT OR Priv_Name = INSERT OR Priv_Name = DELETE)	Test Constraints
grantObjPriv =	TRUE	FALSE	

Term Tables Used for Grant Object Privileges

Table Name	Condition	
	grantor = objOwner	NOT(grantor = objOwner)
grantor_owns_object =	TRUE	FALSE

Table Name	Condition	
	selectedObj = grantedObject AND GRANT_OPTION AND objOwner != grantor AND objOwner != grantee	selectedObj = grantedObject AND NOT(GRANT_OPTION) AND objOwner != grantor AND objOwner != grantee
granted_object_privileges =	TRUE	FALSE

Table Name	Condition	
	(granteeType = user AND granteeRoleID != roleID) OR (granteeType = role AND roleID != NULL AND granteeRoleID = roleID) OR (granteeType = PUBLIC)	 FALSE
grantee_constraints =	TRUE	FALSE

Test Vectors for Grant Object Privilege

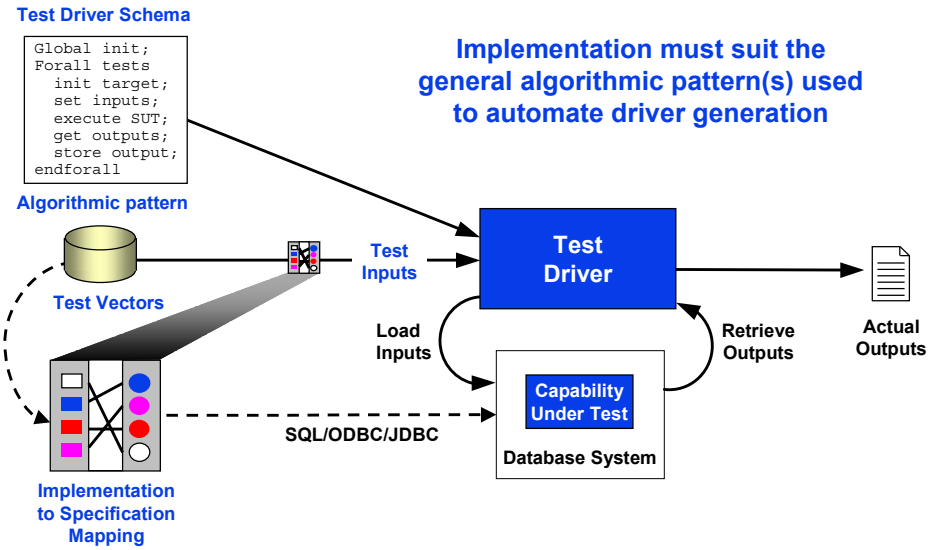
- Translated model results in 20 test specification elements
  - Test specification is logically AND'ed set of conditions for an output
  - Resulted in 40 test vectors (2 / test specification)
  - Based on 6 Term Variables (not shown) and 12 Monitored Variables

Vector #	DCP	grantObjPriv	grantor	grantee	privName	grantee Type	objOwner	selected Obj	granted Object	GRANT_ OPTION	grantee RoleID	roleID
1	1	TRUE	1	2	ALL	user	1	4	4	TRUE	2	2
2	1	TRUE	4	3	ALL	user	4	1	1	FALSE	0	0
3	2	TRUE	1	2	UPDATE	user	1	4	4	TRUE	2	2
4	2	TRUE	4	3	UPDATE	user	4	1	1	FALSE	0	0
5	3	TRUE	1	2	SELECT	user	1	4	4	TRUE	2	2
6	3	TRUE	4	3	SELECT	user	4	1	1	FALSE	0	0
7	4	TRUE	1	2	INSERT	user	1	4	4	TRUE	2	2
8	4	TRUE	4	3	INSERT	user	4	1	1	FALSE	0	0
9	5	TRUE	1	2	DELETE	user	1	4	4	TRUE	2	2
10	5	TRUE	4	3	DELETE	user	4	1	1	FALSE	0	0
▪ ▪ ▪												
37	19	FALSE	1	2	DELETE	user	3	1	1	FALSE	0	1
38	19	FALSE	4	3	DELETE	user	2	4	4	FALSE	2	1
39	20	FALSE	1	2	DELETE	role	3	1	1	FALSE	1	1
40	20	FALSE	4	3	DELETE	role	2	4	4	FALSE	2	2

Test Driver Generation

- Test drivers typically perform the following functions:
  - Initialize the system under test
  - Set system outputs to value other than expected
  - Inject the test inputs
  - Execute the test
  - Retrieve and store test outputs
- General algorithm is encoded into test driver schema
- Mappings used to associate modeled objects with implementation objects or component interfaces
- Test driver generator combines test vectors, schema, and mappings to build test driver

Elements of a Test Driver

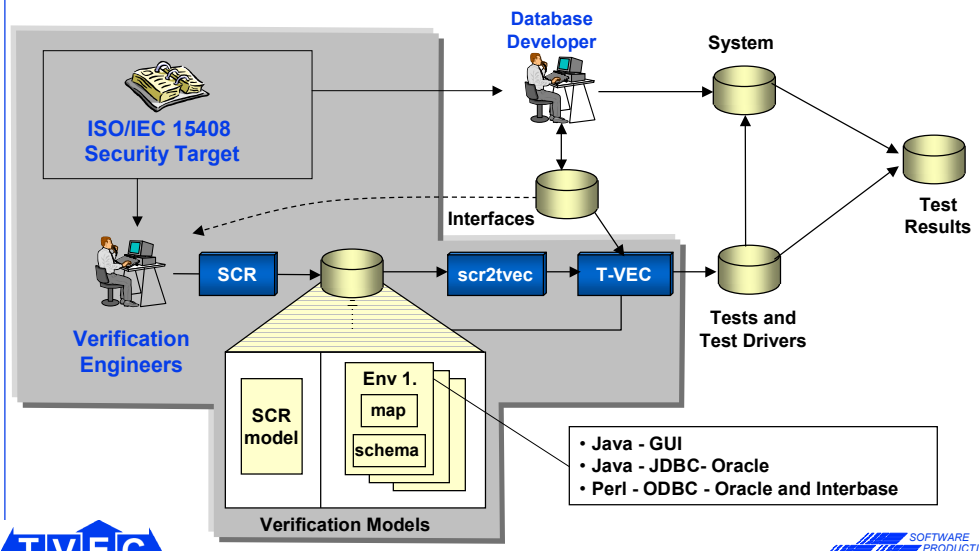


Test Drivers

- Verification model composed of SCR model and one or more environment mappings
  - Verification modeling is best performed in terms of system/component interfaces and specifications
  - Interfaces include SQL language, as well as application program interfaces (APIs)
  - An environment mapping contains the object mappings and schema
- Phase I - generated test drivers for Java application (not discussed here)
- Phase II - generated test drivers using Phase I model for targets:
  - Interbase 6.0 driven with Perl test driver using ODBC interface
  - Oracle 8.0.5 driven with Java test driver with JDBC interface

Developing Verification Models

- Verification model is refinement of requirements defined in terms of system interfaces



Schema Defines Testing Pattern

- Global initialization
  - Logon as system, creates table space, users, assigns roles, etc.
- For each vector
  - Set inputs
    - Setup database (initialize tables, roles, granted privileges)
  - Call method for performing operation (e.g., grantObjPriv, revokeObjPriv, grantSysPriv, grantRole)
    - Perform operation
    - Test operation
    - Access system tables to check for the valid privileges
  - Clean-up
    - Remove privilege, roles, etc.

Execution Trace of Test Execution

- Prior to Line 1, global initialization, creates uses, and table space
- Line 2, user u1 logs on, drops roles, create tables (tab1 – tab4) with initial values of 1 through 4.
- Line 26, 27 u1 creates role1 and role2 (however not used in this test)
- Line 29 user u1 grants ALL on tab4 to user u2 with GRANT OPTIONS
- Lines 30, 31, user u2 logs on and updates tab4 values to a value of 0
- Line 32-35, the test operation performs a logon for SYSDBA and performs a SELECT from System Tables (RDB$USER_PRIVILEGES) to determine if values in tab4 have been modified - outputs privilege settings

```
1.  Test Vector #1
2.  Logon User -> u1
3.     Executed SQL -> DROP ROLE role1
4.     Executed SQL -> DROP ROLE role2
5.     Executed SQL -> CREATE TABLE tab1 (ID INTEGER)
6.     Executed SQL -> INSERT INTO tab1 (ID) VALUES (1)
7.     Executed SQL -> INSERT INTO tab1 (ID) VALUES (2)
8.     Executed SQL -> INSERT INTO tab1 (ID) VALUES (3)
9.     Executed SQL -> INSERT INTO tab1 (ID) VALUES (4)
10.    Executed SQL -> CREATE TABLE tab2 (ID INTEGER)
11.    Executed SQL -> INSERT INTO tab2 (ID) VALUES (1)
12.    Executed SQL -> INSERT INTO tab2 (ID) VALUES (2)
13.    Executed SQL -> INSERT INTO tab2 (ID) VALUES (3)
14.    Executed SQL -> INSERT INTO tab2 (ID) VALUES (4)
15.    Executed SQL -> CREATE TABLE tab3 (ID INTEGER)
16.    Executed SQL -> INSERT INTO tab3 (ID) VALUES (1)
17.    Executed SQL -> INSERT INTO tab3 (ID) VALUES (2)
18.    Executed SQL -> INSERT INTO tab3 (ID) VALUES (3)
19.    Executed SQL -> INSERT INTO tab3 (ID) VALUES (4)
20.    Executed SQL -> CREATE TABLE tab4 (ID INTEGER)
21.    Executed SQL -> INSERT INTO tab4 (ID) VALUES (1)
22.    Executed SQL -> INSERT INTO tab4 (ID) VALUES (2)
23.    Executed SQL -> INSERT INTO tab4 (ID) VALUES (3)
24.    Executed SQL -> INSERT INTO tab4 (ID) VALUES (4)
25. Attempting to create roles.
26.    Executed SQL -> CREATE ROLE role1
27.    Executed SQL -> CREATE ROLE role2
28. Logon User -> u1
29.    Executed SQL -> GRANT ALL ON tab4 TO u2 WITH GRANT OPTION
30. Logon User -> u2
31.    Executed SQL -> UPDATE tab4 SET ID = 0
32.  Logon User -> SYSDBA
33.    Executed SQL -> SELECT * FROM RDB$USER_PRIVILEGES
34.    INSERT->1 REPLACE->1 SELECT->1 DELETE->1 UPDATE->1
35.  Grant ALL -> Pass
```

Other Applications

- Industry applications: flight navigation, guidance, autopilot logic, display systems, flight management and control laws, airborne traffic and collision avoidance
- Demonstrated to work with critical applications like telemetry communication, and mode switching logic for cardiac rhythm management devices
- Automated test driver generation supports standard languages (e.g., C, C++, Java, Ada, Perl, PL/I, SQL),as well as proprietary languages, COTS test injection products, and test environments.
- Non-critical applications like workstation-based Java applications

Summary and Next Step

- Better quality requirements for design and implementation help eliminate rework in those phases as well as during test
- Verification modeling can reduce the time normally spent in verification test planning by up to 50%
- Test generation from a verification model can eliminate up to 90% of the manual test creation and debugging effort
- Both the number of test cases and the phasing of their execution can be optimized, eliminating test redundancy
- A known level of requirements coverage can be planned, and measured during test execution

LM Aero has implemented this process, and results are compelling

SAFFORD 05 TEST
Chart No/11

08/16/2000

Slide from Ed Safford's STC `2000 presentation, used with permission

Summary

- NIST and sponsors cite cost and effort for security functional testing as a large and growing problem
- Assessment of model-based approach performed against security requirements for Oracle Security Target
- Security specifications modeled and tools used to automatically generate test vectors and test drivers
  - Test drivers developed for three environments: Java application, Perl and ODBC for Interbase, and Java and JDBC for Oracle
- Tools continue to be evolved for other modeling approaches: functional, OO-UML, control systems and hybrids
- Additional security models being developed for: audit generation, security management, identification & authentication, and session management

Model-based Approach to Security Test Automation

Mark Blackburn, Robert Busser, Aaron Nauman, T-VEC
Ramaswamy Chandramouli, National Institute of Standards and Technology

Security functional testing is a costly activity typically performed by security evaluation laboratories. These laboratories have struggled to keep pace with increasing demand to test numerous product variations. This paper summarizes the results of applying a model-based approach to automate functional security testing. The approach involves developing models of security requirements as the basis for automatic test vector and test driver generation. In the application, security properties were modeled and the resulting tests were executed against Oracle and Interbase database engines through a fully automated process. The findings indicate the approach, proven successful in a variety of other application domains, provides a cost-effective solution to functional security testing.

1 Introduction

Software security is a software quality issue that continues to grow in importance as software systems are used to manage continually increasing amounts of critical corporate and personal information. The use of the Internet to manage and exchange this data on a daily basis has heightened the need for software architectures, especially internet-based architectures, which are secure. At the same time, the shortened development and deployment cycles for software make it difficult to conduct adequate security functional testing to verify whether software systems exhibit the expected security behavior.

Presently, developing and executing security functional tests is time-consuming and costly. Security evaluation laboratories are struggling to meet demands to test many product variations produced in short release cycles. The situation calls for improving the economics of security functional testing. As a result, the National Institute of Standards and Technology (NIST) initiated a program to develop methods and tools for automating security functional testing [Cha99]. **Security Functional Testing** verifies whether the behavior of a product or system conforms to the security features claimed by the manufacturer (i.e., the product does what it is supposed to do).

NIST and its sponsors initiated a multi-phase investigation to assess the use of a model-based approach to automate security functional testing. Several model-based approaches were accessed as part of the investigation. The approach described in this paper succeeded where others failed to provide end-to-end support including model development, model analysis, automated test generation, automated test execution in multiple environments, and results analysis. The assessment of this approach has demonstrated the feasibility of modeling security requirements to automate testing for various products and target platforms. NIST believes this should improve the economics of security functional testing for security evaluation laboratories, as well as commercial organizations that perform security testing.

1.1 Organization of Paper

Section 2 details NIST's vision for a methodology and toolkit to support automated security functional testing. Section 3 provides an overview of a methodology and toolkit that have been

effective in satisfying NIST's objectives and that form the basis of this report. Section 4 uses an example to illustrate the development of Security Verification Models to support test automation. Section 5 summarizes the activity of model analysis and test vector generation. Section 6 briefly discusses aspects of test driver generation and test execution.

2 NIST Requirements For Automated Security Functional Testing

NIST wishes to develop a methodology and a supporting toolkit to automate the process of Security Functional Testing. This automation will help security evaluation laboratories meet the demand for product testing. The automation approach is based on expressing a product's security functional requirements in a model and using the supporting toolkit to automatically generate tests needed to verify security properties. A model of system security properties is referred to as a **Security Verification Model**. The supporting toolkit processes these models to:

- Check the specification for contradictions, requirement defects, feature interaction problems, and circular definitions. This analysis ensures that the underlying security functional requirements are consistent and reasonable as a basis for testing.
- Generate test cases from the security requirements specifications expressed in the models. These test cases must be effective in demonstrating an implementation satisfies the security requirements. Ideally, the test cases should include test inputs, expected behavior or outputs, and an association between each test and the specification from which it was derived. Test cases of this form are referred to as test vectors to distinguish them from generated tests cases that include only test inputs.
- Check for requirement-to-test traceability and report whether each requirement has an associated test.

As a single fault in security functionality can annul the entire system's security behavior, it is critical that the model representation of the security requirements be complete. The techniques for developing tests to verify the security properties must also provide 100 percent test coverage of the security properties. As system security behavior is often a product of both trusted and untrusted system component, complete testing minimizes the risk of using untrusted components in a system. This risk minimization is an additional objective of the NIST effort.

3 Methodology and Toolkit for Automating Security Functional Testing

The basis for the methodology and toolkit described in this paper is a model-based test automation approach used successfully in various application domains since1996. The approach is referred to as the Test Automation Framework (TAF). The TAF integrates various modeling tools, like the SCRtool for modeling system and software requirements with the test automation tool T-VEC.[*] In this work, that TAF approach was tailored to automate security functional testing through Security Verification Models. The result is a set of guidelines for modeling security requirements. The assessment was based on modeling security requirements in order to automate testing in three distinct environments, as shown in Figure 1. The specific activities carried out in the assessment include:

[*] The Software Productivity Consortium develops TAF translators and methods. The Software Cost Reduction (SCR) method and associated modeling tool, SCRtool, were developed by the Naval Research Laboratory [HJL96]. The T-VEC Test Vector Generation System is commercially available from T-VEC Technologies, Inc.

- Model security requirements in SCR specifications using the SCRtool
- Translate SCR specifications into T-VEC test specification using an existing SCR-to-T-VEC model translator [BBF97; Bla98]
- Generate test vectors from the transformed SCR specification
- Develop test driver schemas for various target test environments
- Generate test drivers for a Java-based application
- Generate Perl test drivers for an SQL database using an ODBC database interface
- Generate Java test drivers for an SQL database using a JDBC database interface

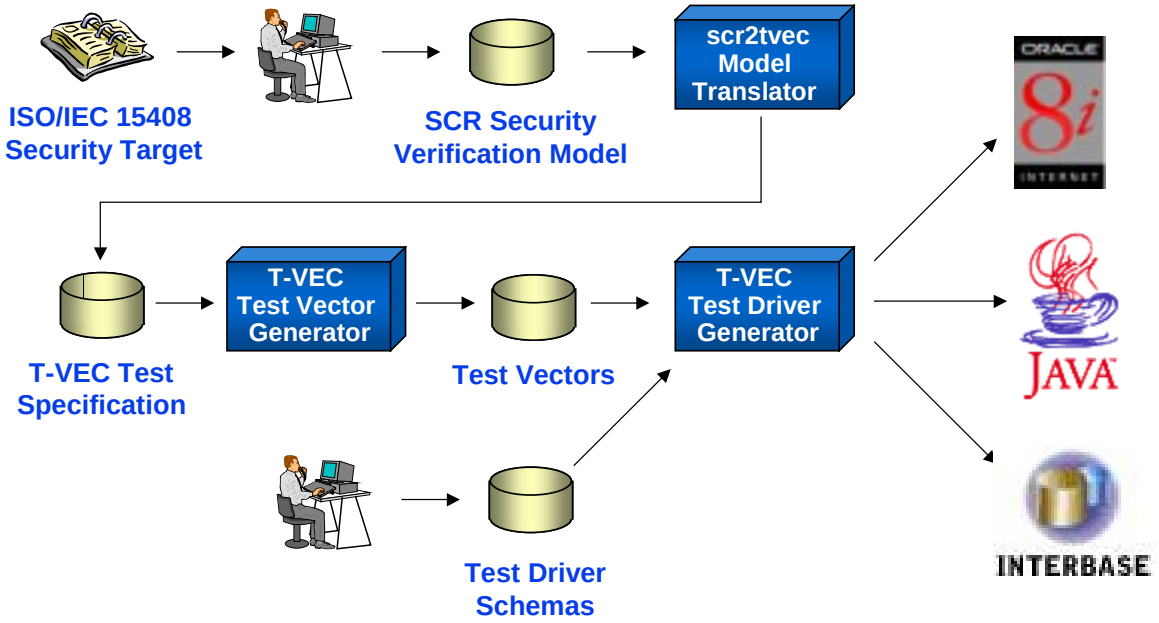


Figure 1. Process Flow Through the Tools

Figure 1 illustrates the process for automated security functional testing used in the assessment. First, security properties from ISO/IEC 15408 Security Target[⊥] specification for Oracle 8 Database Server were modeled in SCR with the SCRtool. An SCR-to-T-VEC translator, developed by the Software Productivity Consortium and T-VEC, was used to translate the SCR model to a T-VEC test specification. T-VEC tools were then used on the T-VEC representation of the security properties to automatically generate test vectors (i.e., test cases with test input values, expected output values and traceability information) and requirement-to-test coverage metrics. The T-VEC test driver generator was used in the assessment to automatically generate test drivers to execute tests against a Java application designed to demonstrate the security properties, an Interbase 6.0 database server and an Oracle 8i database server. These tests were executed and the

[⊥] An ISO/IEC 15408 Security Target is a document that contains a set of Security Functional Requirements, corresponding implementation features and a set of Security Assurance Requirements written in a format that corresponds to an international standard.

results were compared with the expected results from the test vectors to determine each product's compliance to the security properties.[1]

The primary effort in customizing the TAF approach to support security functional testing involved developing heuristics for modeling security properties with SCR and finding techniques for developing test driver schemas to automate execution of SQL statements.

4 Security Verification Model

This section describes the development of a security verification model using the SCRtool through a process of requirement clarification. First, basic SCR modeling concepts are described. This is followed by a description of a security requirement that is then refined into a verification model.

4.1 SCR Modeling Concepts

SCR is a table-based modeling approach, as shown in Figure 2 that models system and software requirements. SCR represents system inputs as **monitored variables**, system outputs as **controlled variables** and intermediate values as **term variables**. Variables are defined as primitive types (e.g., Integers, Float, Boolean, Enumeration) or as user-defined types. Behavior is defined using a tabular approach relating four model elements: modes, conditions, events, and terms. A **mode class** is a state machine, where system states are called system modes and the transitions of the state machine are characterized by guarded events. A **condition** is a **predicate** characterizing a system state. An **event** occurs when any system entity changes value. Terms and controlled variables are functions of input variables, modes, or other terms. Their values are defined in the model through event or condition tables.

4.2 Security Specifications

The security requirements used in the assessment are defined in the Oracle8 Security Target document [Ora00]. This document describes the security functionality (behavior) claimed by Oracle and is submitted along with the product for security evaluation. A subset of the security requirements, referred to as Granting and Revoking Privileges and Roles, was modeled in the assessment. The test vectors derived from the model were used to generate test drivers for two different database servers, Interbase 6.0 and the Oracle 8.0.5.

The following sections describe the process of modeling the *Granting Object Privilege (GOP)* requirement, which is a part of the *Granting and Revoking Privileges and Roles* functionality. The GOP is defined in the Oracle8 Security Target as:

Granting Object Privilege Capability (GOP) - A normal user (the grantor) can grant an object privilege to another user, role or PUBLIC (the grantee) only if:

a) the grantor is the owner of the object; or

b) the grantor has been granted the object privilege with the GRANT OPTION.

A role represents a group of related users. The keyword PUBLIC represents all users.

[1] The process of SCR model translation, test vector generation, test driver generation, and execution against the Interbase database using Perl and ODBC completed in 2 minutes and 54 second running on a 400 MHz Windows NT machine with 256 KB of memory.

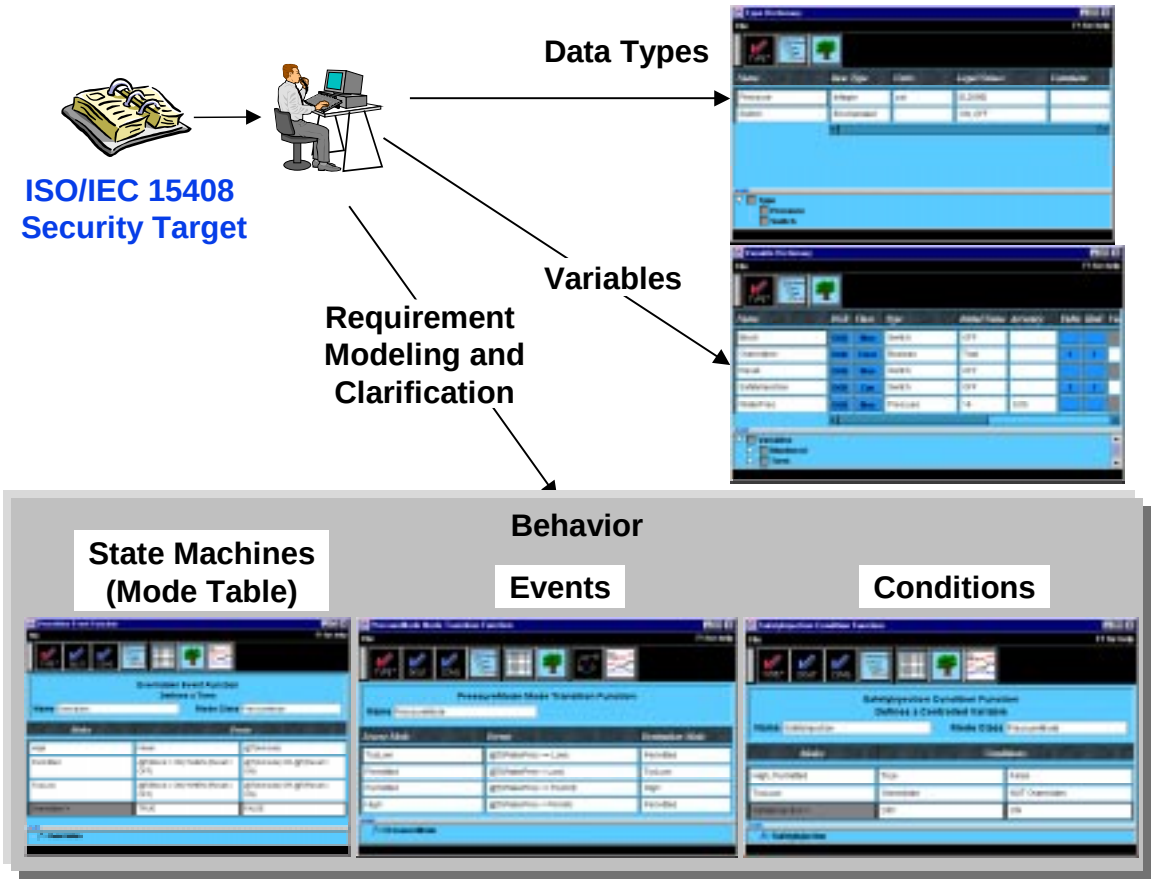


Figure 2. SCR Modeling Constructs

4.3 Requirement Analysis

Developing SCR models requires identifying the system monitored (input) and controlled (output) variables, and defining the relationships between them. This process is typically iterative. It involves defining the variables, the data types associated with the variables, and the tables that define relationships between the variables. A useful guideline for developing SCR models is to work backwards from each output to make the process goal-oriented. The value of each output is defined in terms of the system inputs. Term variables are introduced whenever intermediate values are necessary or useful. The relationships between the inputs and outputs are refined until complete enough to support both manual review and automated analysis. Manual review processes can validate the correctness of the model and completeness with respect to the textual requirements, while automated analysis can identify inconsistencies in the model.

Breaking the GOP requirement into clauses supports identifying variables and relationships. Table 1 contains elaboration and clarification of the GOP requirements to support modeling. In addition, it identifies the variables and relationships associated with each clause.

Table 1. Variables and Relations

Requirement Statement/Clause	Variables	Relations
A normal user (the grantor) can grant an object privilege to another user, role or PUBLIC (the grantee)	grantor	grantee constraints (user, role or PUBLIC)
	grantee	
	object	
	privilege	
	grantee type	
GOP (a) - a grantor can grant an object privilege to a grantee if the grantor owns the object	grantor	grantor owns object
	grantee	
	object	
	privilege	
GOP (b) – a grantor (that does not own the object) can grant object privileges to the grantee if the object owner previously granted object privilege to the grantor with the GRANT OPTION	grantor	granted object privilege
	grantee	
	object	
	privilege	
	object owner	
	GRANT OPTION	
	granted object	

From the analysis above, the monitored (input) variables identified in the system can be refined into the following set:

- privName – type of object privilege that can be granted (ALL, SELECT, INSERT, UPDATE, DELETE, etc)
- grantor – user granting an object privilege
- grantee – user being granted an object privilege
- granteeType – type of grantee for a particular grant operation as defined in the first sentence of the GOP textual requirement; grantee is a user, role, or PUBLIC
- selectedObj – object selected for a particular grant operation
- grantedObject – object for which grant privileges have previously been granted (identified through GRANT OPTION)
- objOwner – owner of the object

Two other variables are related to the concept of a role; a role is a type of grantee as defined in the first sentence of the GOP textual requirement. The related variables include:

- roleID – role being granted an object privilege
- granteeRoleID – role of the grantee (if any) being granted an object privilege

There can be one or more roles defined and known by the database system. The variable roleID is used to refer to a specific role known within the system, and used in various test cases. The granteeRoleID is a specific role assigned to the grantee.

The GOP requirements specify the conditions when privileges are granted for an object. An SCR model of these requirements should ensure that when all model conditions are satisfied, the output indicates the privilege is granted. This output is modeled as the Boolean controlled variable:

- grantedObjPriv– the grant operation executes successfully (TRUE) or fails (FALSE)

4.3.1 Modeling Variables and Data Types

Variables are modeled in the SCRtool through the Variable Dictionary as shown in Figure 3. For example, the grantee is a monitored (input) variable (MON) of type userIDType.

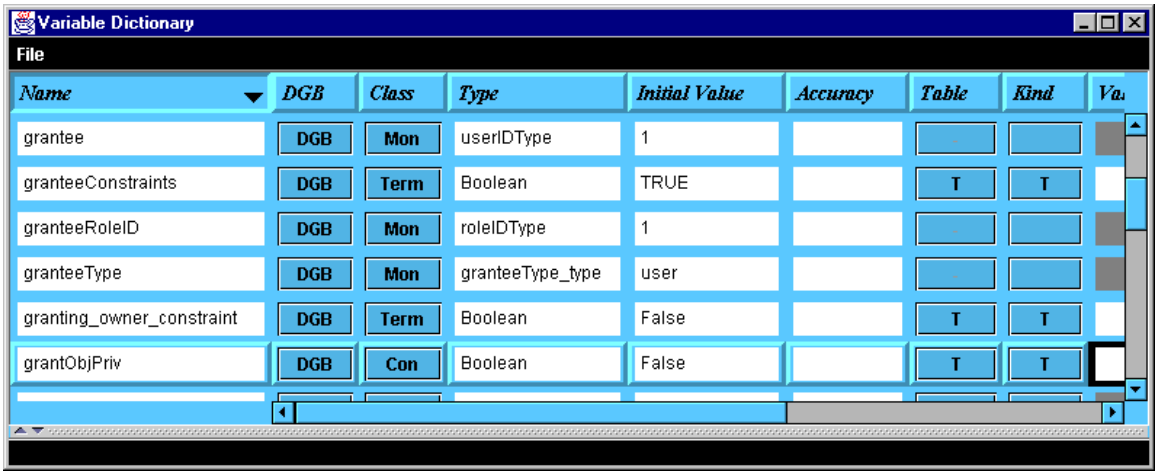


Figure 3. Variables Modeled in SCR

User-defined types are model through the Type Dictionary. Data types can be numeric (Integer and Float), Boolean or Enumerated. Figure 4 shows some of the data types used in the GOP model. The type objectPrivType is an enumerated type whose values define valid privileges associated with an object. The type objectIDType is defined as an Integer with a range of 0 to 5. The SCRtool also has a Constant Dictionary for defining constants.

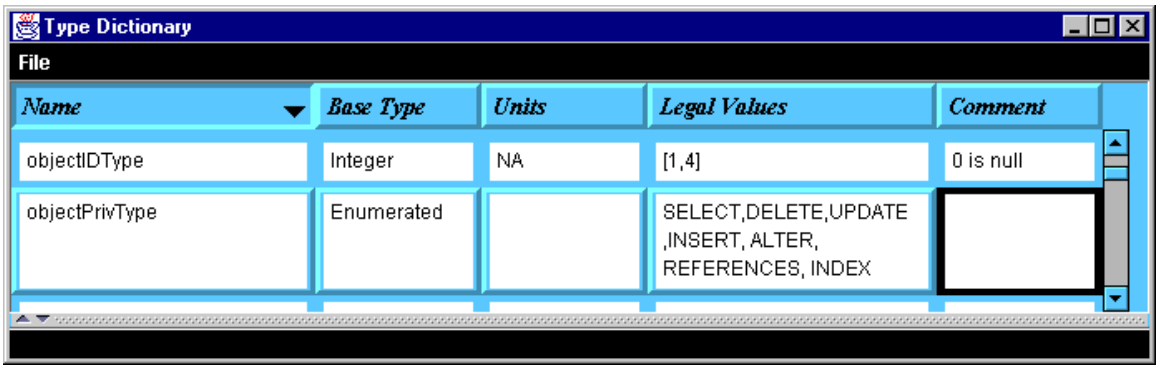


Figure 4. Data Types Modeled in SCR

4.4 Modeling Security Functional Requirements

Once the system's data is defined, its behavior can be modeled. In SCR, this involves defining the values of the controlled (output) variables through condition, event, or mode tables. These tables define the value of a variable in terms of monitored (input) variables, terms (intermediate) variables, and mode (state) machines. Figure 5 provides a representation of the GOP model. The output value, grantObjPriv, is defined by a condition table referencing three other terms. The requirement GOP(a) is directly associated with the term grantor_owns_object and requirement GOP(b) is directly associated with the term granted_object_privileges. The term

grantee_constraints is derived from the first sentence in GOP that defines a grantee as a user, role or PUBLIC.

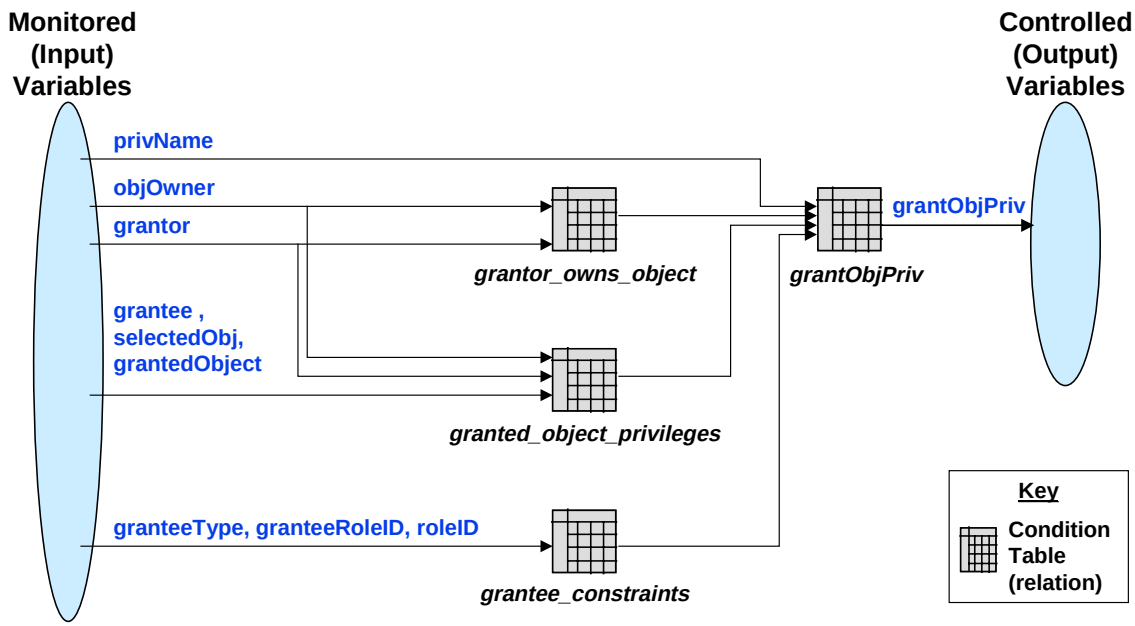


Figure 5. Model Structure for Grant Object Privilege

A value of a **term variable** is defined through a condition or event table as an intermediate value. Terms can be referenced as part of the constraints or value calculations of other terms or controlled variables. They reduce the complexity of the model by simplifying expressions and eliminating redundancies. The following sections describe the terms used in defining the value of grantObjPriv.

4.4.1 Modeling Relation grantor_owns_object

The term grantor_owns_object defines the conditions under which the grantor owns the object for which privileges are being granted. When these conditions are satisfied, the value of grantor_owns_object is TRUE. The condition table for grantor_owns_object is shown in Table 2. It specifies that the term is TRUE (grantor owns the object) when grantor = objOwner, otherwise, the term is FALSE.

Table 2. Table for Relation grantor_owns_object

Table Name	Condition	
	grantor = objOwner	NOT(grantor = objOwner)
grantor_owns_object =	TRUE	FALSE

The conditions within a condition table can include:

- input or term variables
- arithmetic operators (+,-,*,etc.)
- relational operators (=, !=, >, <, etc.)
- logical operators (AND, OR, or NOT)

4.4.2 Modeling Relation grantee_constraints

The first clause of the GOP requirement (See Table 1) specifies that a user, role, or PUBLIC can be granted privileges. These classes of grantees are defined by granteeType. If a role is being granted privileges, the role is identified by roleID. A user can be associated with a particular role, which is represented by the monitored variable granteeRoleID. Table 3 shows the term grantee_constraints that defines the relationships between the granteeType, granteeRoleID, and roleID. There are three cases:

1. If the granteeType is user, then the grantee is a user. To ensure that the grantee is granted privileges as a user and not through the grantee's role, the model specifies that the roleID must not equal the granteeRoleID.
2. If the granteeType is role, then the roleID must be valid, and the granteeRoleID must equal the roleID.
3. If the granteeType is PUBLIC, then the other variables can take on any value (i.e., don't care situation)

Table 3. Table for Relation grantee_constraints

Table Name	Condition	
	(granteeType = user AND granteeRoleID != roleID) OR (granteeType = role AND roleID != NULL AND granteeRoleID = roleID) OR (granteeType = PUBLIC)	FALSE
grantee_constraints =	TRUE	FALSE

The grantee_constraints defines condition on variables that must be TRUE for any grant operation to succeed; therefore, conditions for grantee_constraints are defined when the output is TRUE.

4.4.3 Modeling Relation granted_object_privileges

The GOP(b) requirement states that if a user wishes to grant a privilege to an object and does not own the object, the user must have been granted the privilege with the GRANT OPTION. The term granted_object_privileges shown in Table 4 defines these conditions. The term is TRUE when:

1. the selected object is the object for which the privilege was granted (i.e., the selectedObj is the grantedObject).
2. the privilege was granted with the option to grant others the privilege (GRANT_OPTION is TRUE)
3. the owner of the object is not the grantor
4. the owner of the object is not the grantee

Table 4. Table for Relation granted_object_privilege

Table Name	Condition	
	selectedObj = grantedObject AND GRANT_OPTION AND objOwner != grantor AND objOwner != grantee	selectedObj = grantedObject AND NOT(GRANT_OPTION) AND objOwner != grantor AND objOwner != grantee
granted_object_privileges =	TRUE	FALSE

The FALSE condition for granted_object_privilege requires similar conditions to be TRUE to establish the relationships between the selectedObj, grantedObj, grantor, and grantee, but forces the GRANT_OPTION to be FALSE, because the GRANT_OPTION is the distinguishing condition between these cases.

4.4.4 Modeling Relation grantObjPriv

The definition of grantObjPriv, shown in Table 5, completes the model for the GOP requirement. Its definition includes references to the term tables previously described, as well as additional constraints on monitored variables. The two potential values for grantObjPriv include:

- grantObjPriv = TRUE – test case conditions are such that the privilege should be granted
- grantObjPriv = FALSE - test case conditions are such that the privilege should not be granted

Table 5. Condition Table for Grant Object Privilege (grantObjPriv)

Table Name	Condition		
	(grantor_owns_object	NOT(grantor_owns_object)	**GOP(a)**
	OR	AND	
	(granted_object_privileges AND grantee_constraints))	(NOT(granted_object_privileges) AND grantee_constraints)	**GOP(b)**
	AND	AND	
	(grantor != grantee) AND (granteeType = user OR granteeType = role OR granteeType = PUBLIC) AND (Priv_Name = ALL OR Priv_Name = UPDATE OR Priv_Name = SELECT OR Priv_Name = INSERT OR Priv_Name = DELETE)	(grantor != grantee) AND (granteeType = user OR granteeType = role OR granteeType = PUBLIC) AND (Priv_Name = ALL OR Priv_Name = UPDATE OR Priv_Name = SELECT OR Priv_Name = INSERT OR Priv_Name = DELETE)	**Test Constraints**
grantObjPriv =	TRUE	FALSE	

The conditions are divided into three groups to support explanation. The groups include:

1. GOP(a) – grantor can grant privilege to a grantee because the grantor owns the object
2. GOP(b) – grantor can grant privilege to a grantee because the grantor has been granted object privileges with GRANT OPTION
3. Test Constraints – additional conditions that ensure that the GOP(a) and GOP(b) conditions are fully exercised during test generation. The conditions ensure the following situations are tested:

- grantor is not the grantee
- all possible combinations of the granteeType (user, role, or PUBLIC)
- all possible privileges on operations (ALL, UPDATE, SELECT, etc.)

The differences between the TRUE and FALSE case for grantObjPriv is that the TRUE case establishes the required conditions:

1. the grantor_owns_object relationship that is associated with GOP(a), where the grantor owns the object, or
2. granted_object_privileges and grantee_constraints – that is associated with GOP(b)
3. Test constraints force all combinations to be applied

The FALSE case establishes the conditions under which the grant operation fails:

1. grantor is not the object owner (i.e., NOT(grantor_owns_object))
2. grantor has not been granted object privilege (i.e., NOT(granted_object_privilege))
3. the Test Constraints force complete test coverage of the grant types and privileges

5 Model Analysis and Test Vector Generation

Modeling and test vector generation is typically performed iteratively as the model is developed. The SCRtool provides a number of checks on the model to ensure that individual tables are consistent and complete. The SCR-to-T-VEC model translator and T-VEC tools perform additional checks that identify cross-table inconsistencies and contradictions. These model analysis capabilities support refining the model by identifying and correcting model defects.

The SCR-to-T-VEC model translator transforms each SCR table into a T-VEC subsystem. The T-VEC compiler converts each subsystem into a set of primitive test specifications that are used as the basis of test vector generation [BBF97]. The translated and compiled version of the grantObjPriv requirement includes 20 test specifications. The test vector generator attempts to determine two test vectors for each test specification based on a test selection strategy derived from the concept of **domain testing theory**[2]. Table 6 shows a tabular representation of the 40 test vectors produced for grantObjPriv. The test vectors include 12 monitored variables and 6 term variables (not shown in the table). The test values shown in Table 6 reflect how the test generator systematically selects low-bound and high-bound test points at the domain boundaries. The input

[2] White and Cohen [WC80] proposed **domain testing theory** as a strategy for selecting test points to reveal domain errors. It is based on the premise that if there is no coincidental correctness, then test cases that localize the boundaries of domains with arbitrarily high precision are sufficient to test all the points in the domain. This approach produces test input values that satisfy the conditions of the test specification and that localize the decisions in the specification to maximize defect detection. Once a set of test inputs are selected that satisfy the specification constraints, these inputs are used to derive the value of the output.

values ranges and constraints (e.g., relational operators) of the specification define the domain boundaries. For example, vector # 1, grantor has id = 1, grantee has id = 2, is based on low-bound values of the data type range of userIDType, while vector # 2, grantor has id = 4, grantee has id = 3, is based on the high-bound for the data type range. In addition, the test generator creates a test for each value of privName and granteeType.

Table 6. Test Vectors for grantObjPriv

Vector #	DCP	grantObjPriv	grantor	grantee	privName	grantee Type	objOwner	selected Obj	granted Object	GRANT_ OPTION	grantee RoleID	roleID
1	1	TRUE	1	2	ALL	user	1	4	4	TRUE	2	2
2	1	TRUE	4	3	ALL	user	4	1	1	FALSE	0	0
3	2	TRUE	1	2	UPDATE	user	1	4	4	TRUE	2	2
4	2	TRUE	4	3	UPDATE	user	4	1	1	FALSE	0	0
5	3	TRUE	1	2	SELECT	user	1	4	4	TRUE	2	2
6	3	TRUE	4	3	SELECT	user	4	1	1	FALSE	0	0
7	4	TRUE	1	2	INSERT	user	1	4	4	TRUE	2	2
8	4	TRUE	4	3	INSERT	user	4	1	1	FALSE	0	0
9	5	TRUE	1	2	DELETE	user	1	4	4	TRUE	2	2
10	5	TRUE	4	3	DELETE	user	4	1	1	FALSE	0	0
▪ ▪ ▪												
37	19	FALSE	1	2	DELETE	user	3	1	1	FALSE	0	1
38	19	FALSE	4	3	DELETE	user	2	4	4	FALSE	2	1
39	20	FALSE	1	2	DELETE	role	3	1	1	FALSE	1	1
40	20	FALSE	4	3	DELETE	role	2	4	4	FALSE	2	2

The number of vectors generated and the specific test values depend on the test vector generation mode, test input selection heuristics, and the satisfiability of the test specification conditions. A test specification is considered satisfiable, if a set of input values exist that satisfy all conditions and result in a valid expected output value. Unsatisfiable test specifications typically result from specification errors (e.g., requirement defects).

6 Test Driver Generation and Execution

The last step in the process involves transforming the tests into a test driver that can be executed against a security target, like the Oracle database. NIST stated that the capability to transform models into test drivers for a variety of platforms is an important discriminating capability of this toolset.

The test driver generator combines test driver schemas, user-defined object mappings and test vectors to produce test drivers as illustrated in Figure 6. The test driver schema encodes generic descriptions for test execution based on an algorithmic pattern that is applicable to the specific test environment. The object mappings relate objects in the model to the objects in the implementation or component interfaces. The test driver generator creates test drivers by repeating the execution steps defined in the schema for each test vector. There are typically four primary steps for executing each test case:

- Set the value of the test output to some value other than what is expected
- Set the values of the test inputs
- Cause execution of the test
- Retrieve and save the results of the test execution

Test driver schemas provide a description of how to accomplish each of these steps for a specific testing environment using a small language that can access information about the specification model, data objects, types, ranges, test values, and user customizable information. A schema is also used to describe the form of expected outputs to support test execution and results analysis.

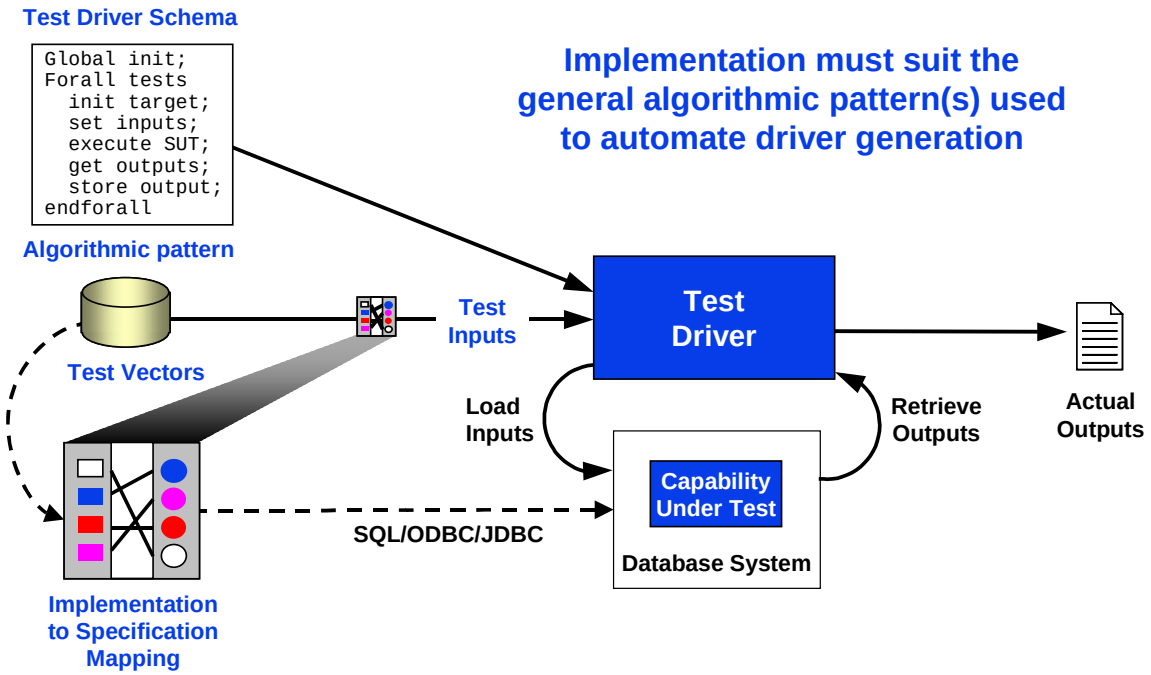


Figure 6. Elements of a Test Driver

Three different test driver schemas and object mapping descriptions were used with the grantObjPriv model to test three different applications. First, a GUI-based Java application was developed to illustrate how test drivers could be injected into an application that has a graphical user interface. Next test drivers were generated for the InterBase 6.0, and Oracle 8 database engine. The Interbase test driver was developed in Perl using ODBC interface to issue SQL commands. The Oracle test driver was developed in both Perl and Java. The Java test drivers used JDBC to communicate to the database.

7 Summary and Future Work

The TAF approach, customized with specific guidelines for modeling security properties and developing test drivers for databases, satisfies NIST's requirements for an automated model-based approach to automated Security Functional Testing. In the assessment of the approach, security requirements for the Oracle8 Security Target were modeled using the SCRtool. These models were then used as the basis of automated test vector and test driver generation with the T-VEC toolset for multiple product applications and test environments. This approach reduces the time and effort associated with security testing, while increasing the level of test coverage. NIST cited the approach's ability to support driver generation for a variety of platforms as a key discriminator. These results demonstrate the feasibility of using model-based test automation to improve the economies of security functional testing. Specifically, the TAF approach is applicable

to security evaluation laboratories and other commercial organizations that need a cost-effective approach for performing security functional testing.

7.1 Other Applications and Results

The core capabilities underlying this approach were developed in the late 1980s and proven through use in support of FAA certifications for flight critical avionics systems [BB96]. Statezni described how the approach supports requirement-based test coverage mandated by the FAA with significant life cycle cost savings [Sta99; Sta2000]. Safford presented results stating the approach reduced cost, effort, and cycle-time by eliminating requirement defects and automating testing [Saf2000]. Safford's presentation summarized the benefits:

- Better quality requirements for design and implementation help eliminate rework in those phases as well as during test
- Verification modeling can reduce the time normally spent in verification test planning by up to 50 percent
- Test generation from a verification model can eliminate up to 90 percent of the manual test creation and debugging effort
- Both the number of test cases and the phasing of their execution can be optimized, eliminating test redundancy
- A known level of requirements coverage can be planned, and measured during test execution

The approach and tools described in this paper have been used for modeling and testing system, software integration, software unit, and some hardware/software integration functionality. It has been applied to critical applications like telemetry communication for heart monitors, flight navigation, guidance, autopilot logic, display systems, flight management and control laws, airborne traffic and collision avoidance. In addition, it has been applied to non-critical applications such as workstation-based Java applications with GUI user interfaces and database applications. The approach supports automated test driver generation in a variety of open languages (e.g., C, C++, Java, Ada, Perl, PL/I, SQL), as well as, proprietary languages, COTS test injection products, and test environments.

7.2 Future Work

The development team continues to evolve the model translation capabilities to support functional, object-oriented, control system and hybrid modeling approaches. In addition, the team is involved in the Object Management Group, UML Action Language Semantics formalization. The team is also involved in the development of modeling guidelines and training material that help integrate commercial modeling approaches with verification tools.

As continued support for NIST, additional models for the Oracle Security Target are being modeled to address the capabilities of: audit generation, security management, identification, authentication, and session management.

8 References

[BB96] Blackburn, M.R., R.D. Busser, T-VEC: A Tool for Developing Critical System. In Proceeding of the Eleventh International Conference on Computer Assurance, Gaithersburg, Maryland, pages 237-249, June, 1996.

[BBF97] Blackburn, M.R., R.D. Busser, J.S. Fontaine, Automatic Generation of Test Vectors for SCR-Style Specifications, In Proceeding of the 12th Annual Conference on Computer Assurance, Gaithersburg, Maryland, pages 54-67, June, 1997.

[Bla98] Blackburn, M. R., Using Models For Test Generation And Analysis, Digital Avionics System Conference, October, 1998.

[Cha99] Chandramouli R., Methodology for Automated Security Testing", NIST Request for Proposal, Nov 1999.

[HJL96] Heitmeyer, C., R. Jeffords, B. Labaw, Automated Consistency Checking of Requirements Specifications. ACM TOSEM, 5(3):231-261, 1996.

[Ora00] Oracle Corporation, Oracle8 Security Target Release 8.0.5, April, 2000.

[Sta99] Statezni, David, Industrial Application of Model-Based Testing, 16th International Conference and Exposition on Testing Computer Software, June 14-18, 1999.

[Sta00] Statezni, David. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, 30 April - 5 May 2000.

[Saf00] Safford, Ed, L. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, 30 April - 5 May 2000.

[WC80] White, L.J., E.I. Cohen, A Domain Strategy for Computer Program Testing. IEEE Transactions on Software Engineering, 6(3):247-257,May, 1980.

QW2001 Paper 4T1

Prof. Warren Harrison
(Portland State University)

A Universal Metrics Repository

Key Points

- Benchmarking
- Decision Support
- Empirical Studies

Presentation Abstract

A neglected aspect of software measurement programs is what will be done with the metrics once they are collected. As a consequence, databases of metrics information tend to be developed as an afterthought, with little, if any concessions to future data needs, or long-term, sustaining metrics collection efforts. A metric repository should facilitate an on-going metrics collection effort, as well as serving as the "corporate memory" of past projects, their histories and experiences. Within this context, four important limitations of contemporary metrics repositories are: obsolescence; ambiguity; augmentation (or lack, thereof); and focus. In order to addresses these issues, we have suggested a transformational view of software development which treats the software development process as a series of artifact transformations. Each transformation has inputs (artifacts) and produces outputs (artifacts). The use of this approach supports a very flexible software engineering metrics repository.

About the Author

Warren Harrison is Professor of Computer Science at Portland State University. His research interests include both software engineering and internet technologies. Professor Harrison's software engineering research includes return on investment for process improvements, software quality assurance, software measurement, and empirical studies of software engineering. He is an active member of the software engineering research community, serving as Editor-in-Chief of the Software Quality Journal and co-EIC with Vic Basili and Lionel Briand of Empirical Software Engineering, as well as being involved with the organizing committees of numerous international conferences and workshops each year. His PhD is from Oregon State University.

A Universal Metrics Repository

Warren Harrison
Portland State University
and the
Oregon Master of Software Engineering

Quality Week 2001
May 29-June 1, 2001

Using Measurement to Gain Information

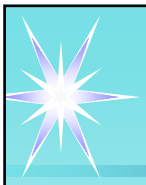
- Decision-making with none, partial or complete information
  - How frequently has an event occurred?
  - What happened when we took a particular action?
  - How many resources did we expend?
- The specific information needed depends on the decision to be made
- Information needs to be collected and stored to support decision-making

Metrics Repositories

- A Repository: *a place, room, or container where something is deposited or stored* (Merriam-Webster's Collegiate Dictionary)
- A software metrics repository is a collection of information pertaining to the development of a software product
- Represents the *objective* corporate memory

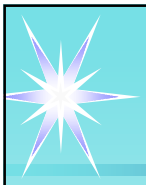
Importance of Metrics Repositories

- Basili, 1980: "*All the data collected on the project should be stored in a computerized data base.*
- Recommendation from the Workshop on Executive Software Issues (Martin, 1989): "*Software organizations should promptly implement programs to: Define, collect, store in databases, analyze, and use process data*".



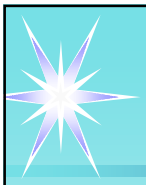
Typical Metrics Repositories

- Most metrics repositories are designed to store a predetermined set of metrics
  - DACS Productivity Dataset
  - Architecture Research Facility Error Repository (ARF)
  - NASA/SEL Dataset



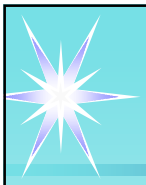
Repository Consolidation

- Electronic repositories allow consolidation of experiences - *if their structure and representation is compatible*
- Consolidation facilitates comparison and makes historical events more significant
- Consolidation is problematic - repositories evolve and become incompatible



Evolving Repositories

- The data collection process is iterative.
- The more we learn, the more we know about what other data we need and how better to collect it.
- As users learn more about the products, processes and metrics, we should expect the information they desire to change.



Contemporary Repositories

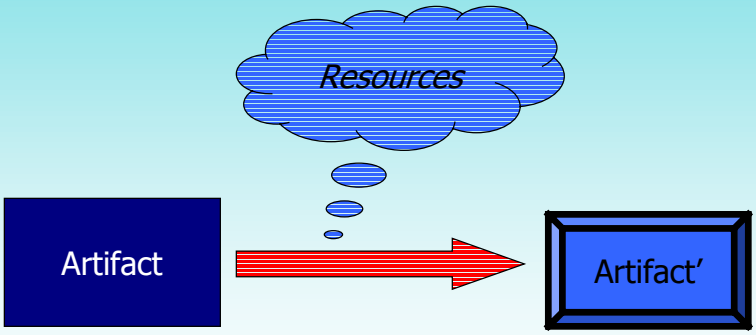
- **Obsolescence**
- **Ambiguity**
- **Augmentation**
- **Focus**

- *leads to an inflexible, product-centric view of software engineering decision-making*

The Transformational Process of Software

- Software is comprised of artifacts - specifications, designs, code, etc.
- Artifacts are transformed into other artifacts
- Artifact transformation consumes resources
- We're interested in recording information about transformations and the artifacts we produce

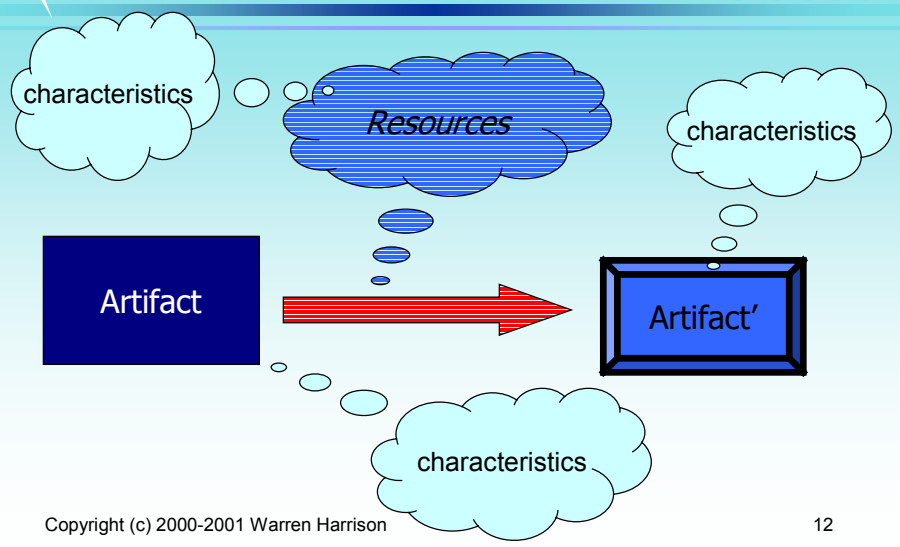
The Transformation Process

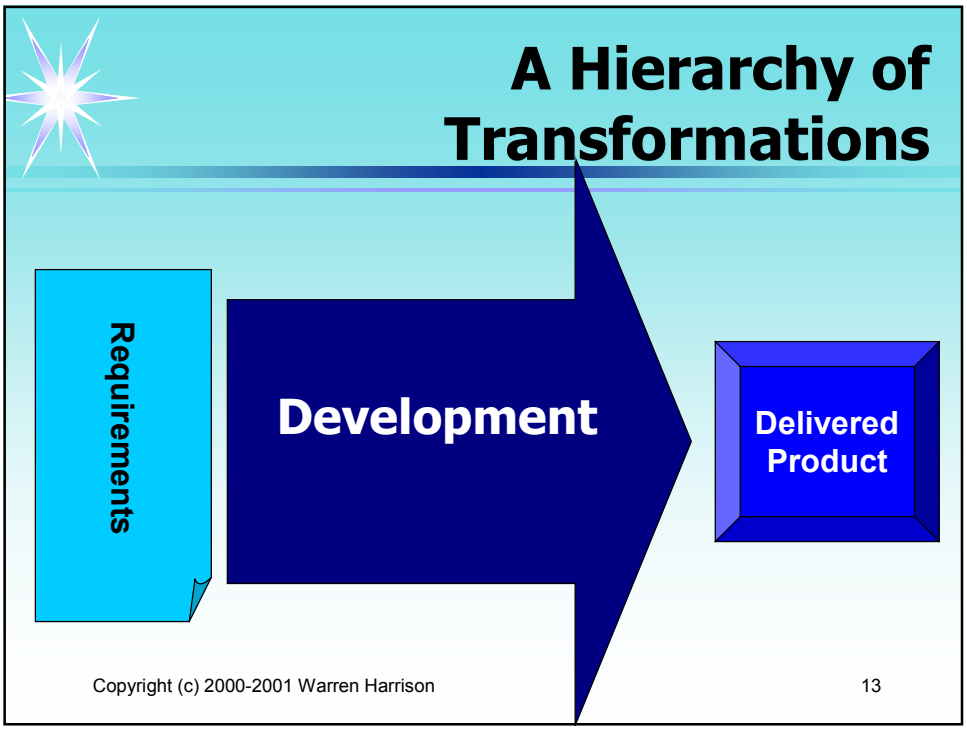


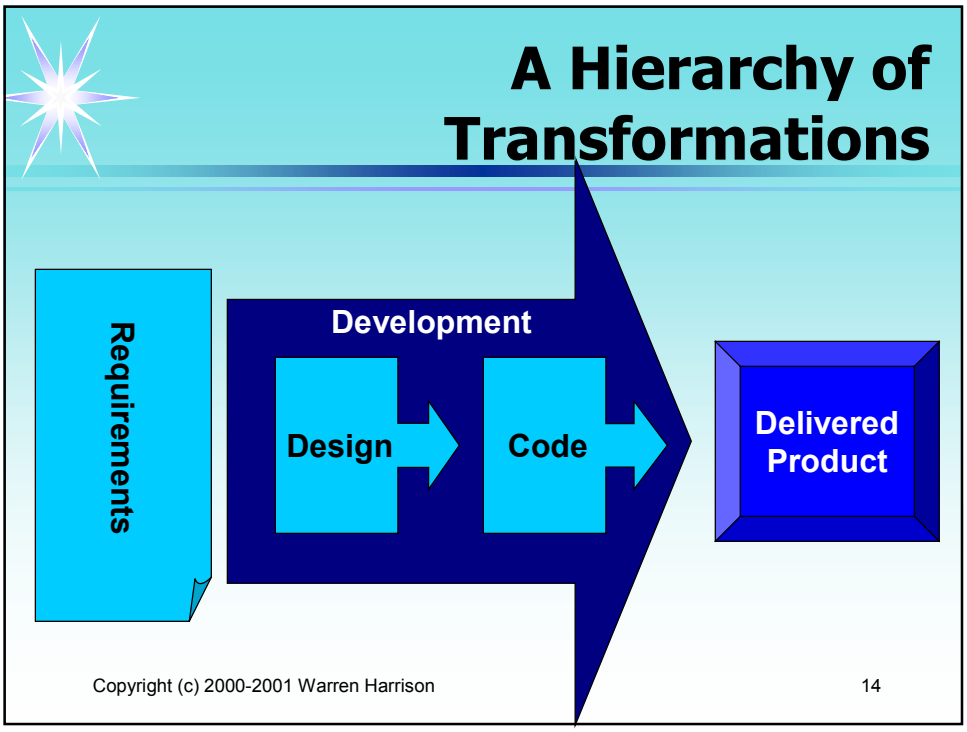
Information Associated With Each Transformation

- Inputs - the input object(s) - capture quantity and characteristics
- Outputs - the output object(s) - capture quantity and characteristics
- Resources - the resources used to perform the transformation

The Transformation Process




A Hierarchy of Transformations
Requirements
Development
Delivered Product
Copyright (c) 2000-2001 Warren Harrison
13


A Hierarchy of Transformations
Requirements
Development
Design
Code
Delivered Product
Copyright (c) 2000-2001 Warren Harrison
14

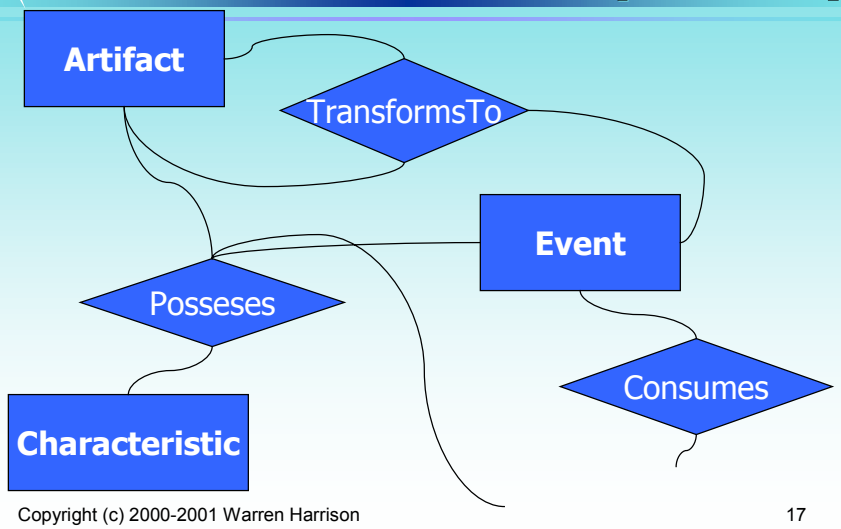
Transformation Granularity

- Some efforts will record information about very coarse transformations
- Some efforts will record information about very fine transformations
- May differ between organizations or even among projects within the same organization

Artifacts/Transformations as Entities & Relationships

- *Entities and Entity Properties* - "things" you want to maintain information about - use properties to characterize such a "thing".
- *Relationships and Relationship Properties* - represent relationships between entities - properties are used to characterize the relationship.

A Transformation Based Repository

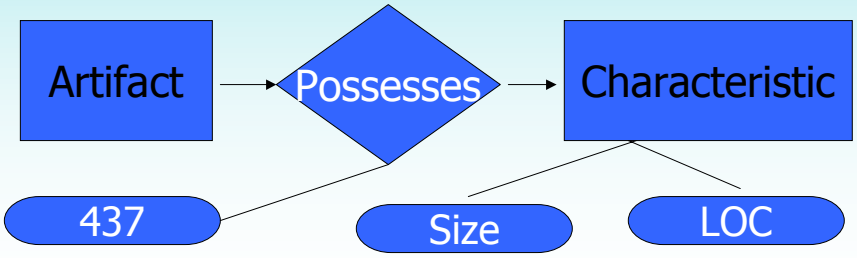


Representing Transformations

- Represent artifacts as entities
- Represent transformations as relationships

Representing Artifact Properties

- **Relate artifacts to properties through explicit relationship.**



Current Status

- *Proof-of-Concept*
- Schema implemented using mySQL (www.mysql.com)
- Populated using Air Force data (DACS) - over 1,000 artifacts
- Implemented mySQL queries via the web (`http://www.cs.pdx.edu/~reposit`)

A UNIVERSAL METRICS REPOSITORY

Warren Harrison
warren@cs.pdx.edu
Department of Computer Science
Portland State University
Portland, OR 97207-0751
http://www.cs.pdx.edu/~warren

ABSTRACT

A metric repository should facilitate an on-going metrics collection effort, as well as serving as the "corporate memory" of past projects, their histories and experiences. Within this context, four important limitations of contemporary metrics repositories are: obsolescence; ambiguity; difficulty of augmentation; and focus. The use of a transformational view of software development supports a very flexible software engineering metrics repository.

Key Words/Phrases: Software Metrics, Software Engineering Repositories, Software Engineering Data Collection

Introduction

When metrics are collected pertaining to a software product or process, the measurements are usually stored for later retrieval. Such a collection of metrics data is known as a *metrics repository*. Unfortunately, a neglected aspect of software measurement programs is what will be done with the metrics once they are collected. As a consequence, databases of metrics information – we hesitate to call some of these efforts *metric repositories* – tend to be developed as an afterthought, with little, if any concessions to future data needs, or long-term, sustaining metrics collection efforts.

For instance, in what was described as a study in "the process and methods used, experience gained, and some lessons learned in establishing a software measurement program" the associated metrics database was simply a collection of ten separate spreadsheet files [Rozum 1993]. We feel that this state of affairs has a significant affect on the viability of measurement programs and software engineering research in general.

This document describes a method of viewing the evolution of a software project that support flexible data collection and describes a prototype repository that addresses these issues.

General Limitations of Contemporary Metrics Repositories

If the only purpose of a metrics repository is as a place to maintain a "tool dump" of a single project or set of completed projects, then the limitations of most repositories are minor. However, if the repository is expected to serve as a home for ongoing collection efforts, then most contemporary metrics repositories are seriously deficient. In our discussions we assume not only that the metric repository is intended to support an ongoing metrics collection effort, but we additionally consider the metrics repository as the "corporate memory" of past projects, their histories and experiences. Within this context, four important limitations of contemporary metrics repositories are:

1. **Obsolescence** – In most examples of contemporary metrics repositories we have surveyed, specific metrics are "built-into" the schema. For instance, consider the measure of "Delivered Source Lines", or "Cyclomatic Complexity". What happens when a specific metric goes out of vogue? Do we continue to collect the obsolete metrics in order to remain compatible with the schema? If new metrics become popular, do we avoid collecting them because "there's no place to put them?" Or do we simply chuck what we have and start over?
2. **Ambiguity** - It may often be the case that users of a particular repository do not know what a specific field within the repository means unless they were involved in the original collection efforts. Florac [1992] points out the importance of *Communicatio*n (will others know precisely what has been measured and what has been included and excluded?) and *Repeatabilit*y (would someone else be able to repeat the measurement and get the same results?). Layout documentation of most repositories is limited (at best) to often out-of-date documentation files. When a user sees the word "Design" in an error report, does that mean the error was injected, found or fixed during the design phase? Similar issues exist for almost every other property of a project we may wish to retain - for instance Goethert, et al [1992] propose a definition for counting staff hours, Park [1992] suggests a method for counting source statements.
3. **Augmentation** – It is expected that as we gain experience with the use of metrics, we'll wish to augment the information set they provide us. A manager who becomes used to obtaining defect information might then desire information on design events or maintenance effort. If the repository does not embrace such augmentation of data, we'll find groups of additional repositories springing up to meet the information needs of various stakeholders. The danger in this is that multiple repositories are terribly difficult to keep consistent and multiple repositories represent significant inefficiencies in effort to both capture and utilize the information. Additionally, the difficulty of mining information from diverse repositories makes it less likely that the entire store of organizational knowledge will be accessed.
4. **Focus** – Most repositories are historical. That is, they reflect occurrences and properties from past projects. In addition, because they focus on what has happened as opposed to what is currently happening, they tend to be product-centric. As such, they are quite good at reflecting information about products, but not so good at reflecting information about processes or events. While product information is important, much decision-making deals with process and event information more than product information.

The current state of repository technology leads to an inflexible, product-centric view of software engineering decision-making. In the following section, we describe a method of viewing the software development process that integrates product and process data in a natural, flexible manner.

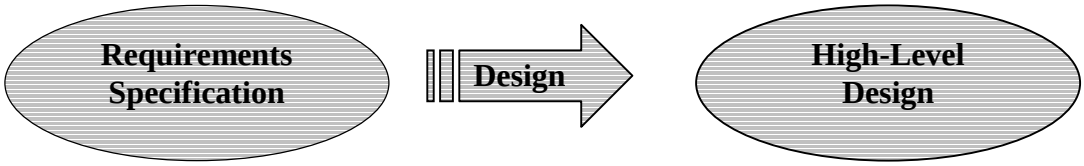
The goal of our work is to define, implement, evaluate and populate a "universal" metrics repository that will address these issues.

A Model of Development to Support Data Collection

In order to address the issue, the repository design must view a software project as a dynamic, growing collection of artifacts as opposed to a static, monolithic bucket of code. Unfortunately, the current view of metrics repositories is exactly that. The general view considers the software development process as simply a collection of intermediate products. For instance, we might characterize a software project as consisting of a requirements specification, a design, and finally a collection of code models.

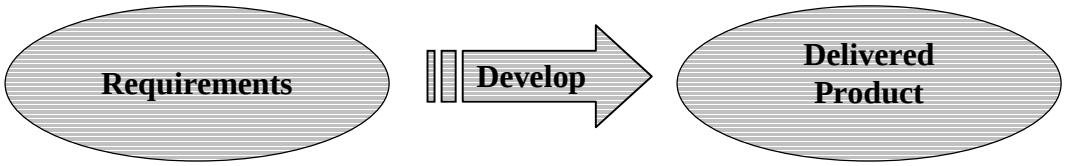
A Transformational View of Software Development

The transformational paradigm of software development views the software development process as a series of transformations of artifacts. An artifact is some identifiable product of an activity that takes place during the software development process. For example, the needs analysis phase of a software development project produces an artifact we sometime refer to as a Requirements Specification. In turn, these artifacts are used as inputs to other transformations to create new artifacts. That is, an activity transforms one or more artifacts into a new kind of artifact or collection of artifacts. Each transformation has inputs (artifacts) and produces outputs (artifacts). For example, the process we usually refer to as “design” transforms a requirements specification into a design document:



The transformations can be described at arbitrary levels of abstraction. For instance, the requirements could actually be logically (and perhaps physically) separated into two pieces – say the user interaction and internal processing segments. This implies that the "Design" transformation actually creates two artifacts: the user interaction and internal processing artifacts.

Of course, the paradigm supports any level of abstraction desired. So for instance, the requirements artifacts could be paragraphs, pages, chapters, documents, or the entire set of requirements depending upon the needs of the user. To illustrate an even more abstract representation, consider the entire software development process that transforms a requirements specification into a delivered product:

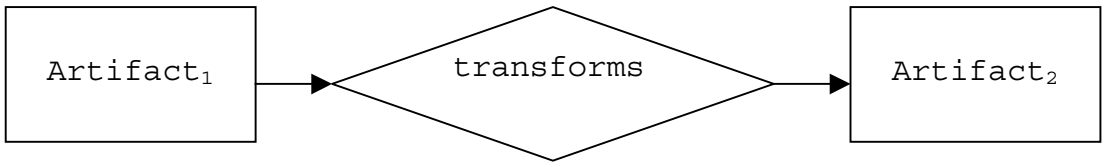


The Transformational Approach to Iterative Software Development

A major weakness of the traditional "intermediate product" view of repositories is its inability to accommodate iterative development. When a product is returned due to a change request, it may be awkward to represent the change in a new version of the product. For instance, should an error be found during testing which results in a code change, either the revised code unit will not be represented in the repository, or the changed code unit will replace the original design product. On the other hand, the transformational view accommodates this situation quite naturally.

Schema Issues

The transformational approach to software development gives rise to the organizational aspect of our proposed repository, which consists of `Artifact` entities connected via `transforms` relationships:



This provides the flexibility to represent an arbitrary flow of artifacts through a software development process, regardless of process model or level of granularity. In order to maintain appropriate information about the transformation, each must be annotated with descriptive items. A transformation is associated with an `event` that denotes the sort of activity or event that took place to effect the transformation, a count of resources consumed during the transformation, and a completion date. This gives rise to the second aspect of our proposed repository, which annotates `event` and `end_date` of each transformation and associates it with resources consumed:

Each artifact also is associated with certain specific information. However, rather than enforcing a standard selection of characteristics (which is what most previous repositories have chosen to do) the third aspect of the proposed repository is that artifacts are linked via relationships to as many or as few characteristic entities as data permits.

For instance, an artifact such as a code module may be data rich or poor. It may be linked to dozens of characteristics, or it may be linked to a single one. Each characteristic includes a `name`, `type` and `description` and the `possesses` relationship is annotated to reflect the quantity of the characteristic.

Thus, an artifact characteristic has a property describing the "domain" of interest, such as *complexity*, *size*, *application area*, etc. In addition, each characteristic has a "working name", such as *Cyclomatic Complexity*, *Software Science Effort*, *Lines of Code*, etc. Because the characteristics that can be associated with an artifact may include measures made over many different time periods, by many different people for many different reasons, the schema includes the `descriptions` property. This meta-data allows each data item to be defined in order to avoid inappropriate combining of data (or discover opportunities to appropriately combine data which have different names) simply because the characteristics have similar names (e.g., "Lines of Code" could mean a variety of different measures which should not be combined).

A Prototype Implementation

We have implemented a proof-of-concept prototype using MySQL. The project homepage can be found at `http://www.cs.pdx.edu/~reposit/`. This site describes the schema and provides limited access to the prototype implementation. The Prototype Repository consists of the following tables or relations (currently we assume the repository consists of metrics for a single project, thus we omit a Project entity, future versions of the repository will support multiple projects):

Entities:

- Artifact (**id**, name)
- Event (**id**, name)
- Resource (**id**, name)
- Characteristic (**id**, name, type, description)

Relationships (all are *n*-to-*n*, and the entire tuple comprises the key):

- transforms (Tid, Artifact.id, Artifact.id, Event.id, EndDate)
- possesses ({Artifact,Resource,Defect,Event}.id,Characteristic.id, qty)
- consumes (transforms.id, Resource.id, quantity)

Each entity instance is assigned a unique unsigned integer identifier (id). This is done by assigning consecutive, unsigned integers to each entity instance as it is added to the database. In addition, each Transforms relationship is also assigned a unique id from this pool of unsigned integers.

The intention is to capture the transformation of one or more artifacts (say a design specification) to another artifact (say a piece of code), and the resources necessary to perform the transformation.

Artifacts represent the specific "trackable" items that make up a project – code units (files, modules, packages, functions – the level of granularity is aribtrary), specification documents, design documents, test cases, etc. *Events* represent types of activities that occur, such as an inspection, a modification, an error correction, etc. *Resources* represent the types of things valued by the project that are consumed in the process of creating the artifacts – effort, calendar time, computer time, etc. *Characteristics* represent types of information about the entities – size, color, weight, etc., as appropriate and available.

The relationships connect various entity instances to other entity instances. The most significant relationship is the *TransformsTo* relationship. *TransformsTo* represents the transformation of one Artifact into another Artifact because of an Event that occurs. We are also interested in the *Resources* that are consumed by a particular transformation, so the *Consumes* relationship associates a particular transformation with a particular resource type as well as the quantity of this type of resource that was consumed during the transformation. Every Artifact, Event, and Resource *Possess* a certain set of Characteristics.

A partial example follows:

```
Artifact(001,A1)
Artifact(002,A2)
Artifact(005,A3)
Artifact(013,A4)
Event(003,Bug Fix)
Resource(004,Programmer Effort in Hours)
Characteristic(007,Cyclomatic Number,Complexity,V(G) per XYZ tool)
Characteristic(008,LOC,Size,Number of non-blank lines in module)
Characteristic(009,Pages,Size,Number of non-blank pages in document)
Characteristic(010,C Code,Type,File consisting of C source code)
Characteristic(011,Requirements,Type, Features the product implements)
Characteristic(015,Bug Report,Type,Record of a bug)
Characteristic(016,Bug Description,Info,Missing Reply Feature)
TransformsTo(012,002,005,003,11-01-99)
TransformsTo(012,013,005,003,11-01-99)
Possesses(001,011,0)
Possesses(002,010,0)
Possesses(005,010,0)
Possesses(002,008,49)
Possesses(005,008,59)
Possesses(001,009,29)
Possesses(013,015,0)
Possesses(013,016,0)
Consumes(012,004,10)
Recorded(002,006)
```

This example represents the following facts (among others).

1. Artifact 1 is a 29 page requirements document
2. Artifact 2 is a 49 line C file

3. Artifact 5 is a 59 line C file that resulted from a modification of Artifact 2 in response to a Bug Fix to correct a “Missing Reply Feature”, which took 10 hours of programmer effort
4. Artifact 13 is a bug report

Summary and Next Steps

We currently have a tentative schema and working prototype for a Universal Metrics Repository. Our next effort will be to expand the repository by populating it with data from several other sources. This exercise will address two interesting questions:

> (a) can a single repository schema in fact represent data from a variety of heterogeneous schemas and
> (b) can the schema successfully integrate data from the various sources to seamlessly base answers to queries across multiple sources.

To this end, we are currently working at populating the repository with data from other sources, including other DACS datasets as well as data from industrial repositories.

References

[Basili 1980] Basili, Victor R., “Data collection, Validation, and Analysis”, *Tutorial on Models and Metrics for Software Management and Engineering*, IEEE Computer Society Press, 1980).

[Baldo 1997] Baldo J., Butcher, D., Nada, N, Poulin, J., Quinones, Y., Trump, D. Scoy, F. and Wu, Z., "Software Reuse Metrics Working Group Summary", *Proceedings of Reuse '97*, 1997.

[Boehm, Brown and Lipow 1976] Boehm, B. W., Brown, J.R., and Lipow, M., “Quantitative Evaluation of Software Quality”, *Proceedings, Second International conference on Software Engineering*, 1976, pp. 592-605.

[DACS 1990] *DACS Productivity Dataset* (http://www.dacs.com/databases/sled/prod.shtml)

[Florac 1992] Florac, W., "Software Quality Measurement: A Framework for Counting Problems and Defects", SEI Technical Report CMU/SEI-92-TR-022 , 1992.

[Goethert, etal 1992] Goethert, W., Elizabeth, K., Bailey, E, and Busby, M., "Software Effort & Schedule Measurement: A Framework for Counting Staff-hours and Reporting Schedule Information", SEI Technical Report CMU/SEI-92-TR-021, 1992.

[IEEE 1992] *IEEE Standard for Software Productivity Metrics*, IEEE Std 1045-1992.

[Martin 1989] Martin, R., Carey, S., Coticchia, M., Fowler, P. and Maher, J., "Proceedings of the Workshop on Executive Software Issues August 2-3 and November 18, 1988", SET Technical Report CMU/SEI-89-TR-006, 1989,

[Park 1992] Park, R., "Software Size Measurement: A Framework for Counting Source Statements", SEI Technical Report CMU/SEI-92-TR-020 ADA258304 , 1992.

[Rozum 1993] Rozum, J. , "The SEI and NAWC: Working Together to Establish a Software Measurement Program", SEI Technical Report: CMU/SEI-93-TR-07, December 1993.

[Van Verth 1992] Van Verth, P., "A Concept Study for a National Software Engineering Database", SEI Technical Report CMU/SEI-92-TR-023, 1993.

About the Author

Warren Harrison is a Professor of Computer Science at Portland State University and Adjunct Associate Professor of Medical Informatics and Outcomes Research at Oregon Health Sciences University. His software engineering research includes models of return on investment for process improvements, software quality assurance, software measurement, formalized decision-making and empirical studies of software engineering. He is currently Editor-in-Chief of the *Software Quality Journal* and co-Editor-in-Chief of the *Empirical Software Engineering Journal*. Warren received his B.S. in Accounting from the University of Nevada – Reno, his M.S. in Computer Science from the University of Missouri – Rolla, and his Ph.D. in Computer Science from Oregon State University.

QW2001 Paper 4T2

Mr. Suresh Nageswaran
(Cognizant Technology Solutions CTS)

Test Effort Estimation Using Use Case Points (UCP)

Key Points

- Test Effort Estimation
- Use Case Points

Presentation Abstract

This paper presents a new approach based on Use Case Points [UCP] as a fundamental project effort-estimation measure. The use of UCP on projects is still in its infancy and it is not a very widely known measure. However, from preliminary applications on our web-based projects, we conjecture that this could in fact be more reliable than FP.

The caveat here is that the V-model must be in use and use case generation must start becoming available right at the requirements gathering phase. The acceptance test plan is then prepared with the use cases from the requirement documents as input.

It is known that a use case to test case mapping is possible. This means that the UCP figure for development can be indirectly used to provide a figure for the number of test cases. Using organizational test execution time metrics it is now possible to arrive at a figure for the total test effort.

About the Author

Suresh Nageswaran holds a Bachelor’s Degree in Computer Engineering from the University of Pune. In addition to this, he also holds the CQA certification. He has over 4 years of experience in Software Testing, Quality Assurance, Design and Development. He has also lectured on software engineering and compiler construction at Pune University.

He currently heads teams of test engineers at Cognizant Technology Solutions (CTS), Pune. He has worked on varied technologies on different platforms and has conducted performance and scalability tests on conventional client/server systems as well as web-based applications.

Test Effort Estimation Using UCP

By

Suresh Nageswaran
Netolutions Group
Cognizant Technology Solutions
Pune, India.
http://www.cognizant.com

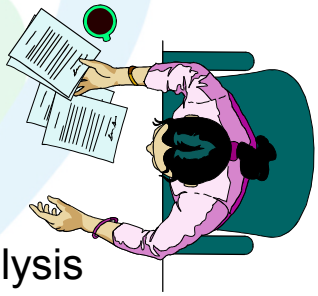
Cognizant Technology Solutions

The Roadmap !

- Introduction.
- Software Test Engineering
- Conventional methods of effort estimation.
- The UCP Approach.
- Example.
- Conclusion.

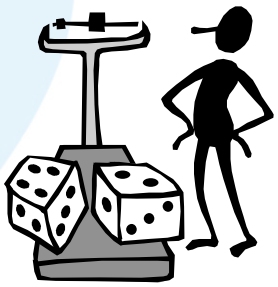
Test Engineering Activities

- Test planning
- Resource Setup time
- Unit Test Rounds
- Defect Tracking
- Metrics Collection and Analysis
- Integration Cycles
- System Release Tests



Why Estimate ?

- An estimate is an appraisal of the value of something.
- The output of an estimation exercise enables us to plan ahead.
- Project profitability is tied to the budget.
- We can schedule and prioritize tasks.



Why Estimates Fail

- Premature estimates - you cannot estimate what you do not understand
- Lack of historical data - estimates are projections of the past to the future
- Lack of estimation process - defined process required
- Failure to manage the estimates
- Failure to update the estimates

Estimation Tasks

- Estimate Size of product
  - LOC
  - FP
  - UCP
- Effort in person-months
- Schedule in calendar months
- Cost in currency

Conventional Approaches

- Ad hoc Method
  - Pre-decided by management or marketing
  - Test until time runs out !
  - Test until money runs out !
- Immature process
- Error margins of over 100% at times
- Not defendable

Conventional Approaches

- Percentage of Development Time
  - Different organizations use different percentages
  - Varies from 10% - 60%
  - Should be based on the risk
- Unscientific
- Schedule overruns from 30% - 60%

Conventional Approaches

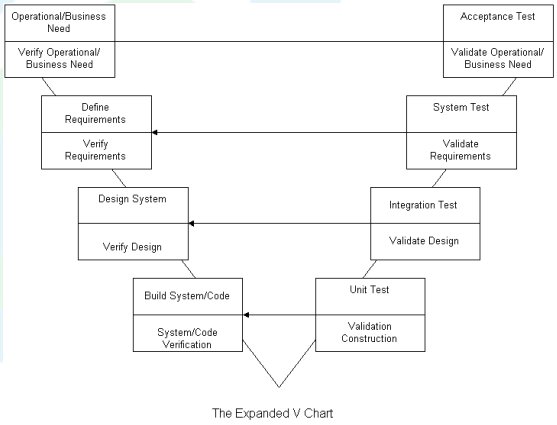
- From Function Point Estimates
  - Capers Jones equation
  - No of test cases = $(FP)^{1.2}$
  - Actual effort calculated with a conversion factor
- Detailed Requirements needed in advance
- Modern OO systems designed with Use Cases in mind

What are Use Cases ?

- Use Cases capture a contract between stakeholders of a system about its behavior.
- Actors initiate interactions with the system to achieve a goal.
- Different scenarios unfold depending on requests made.
- Use Case collects all those scenarios with their exception flows.

UCP Approach Needs

- Usage of the V-Model for development
- Treating Test Engineering as a process, not a stage
- Business Use Cases should have been identified
- As requirements become clearer, estimates are revised.



The Expanded V Chart

UCP Approach

1. Determine number of actors - Unadjusted actor weights
2. Determine number of use cases in the system - Unadjusted Use Case Weights
3. Calculate Unadjusted UCP = UAW+UUCW
4. Compute Technical and Environment Factors
5. Compute adjusted UCP

   AUCP= UUCP*[0.65+(0.01*TEF)]
6. Arrive at final effort

   Effort = AUCP * Conversion Factor

Example

- Product Support Web site for North American software company
- Business requirements documented as Use Cases
- Development time estimated in UCP
- Test Efforts needed to be budgeted for.
- Risks of downtime were significant.
- COM/DCOM technology.

Example

1. Unadjusted Actor Weights Calculation

Actor	*No of Use Cases*	*Factor*	*UAW*
B2C User	15	2	30
Subscribers	13	2	26
Admin User	4	2	8
Total UAW			**64**

- Total UAW = 64

Example [Contd.]

2. Unadjusted Use Case Weights Calculation

Legend: Simple – S, Average – A, Complex – C, Very Complex - VC

Use Case	*Type*	*Factor*	*Reason*
Login	C	15	Server integration
Support Request	VC	20	External Sys Query
User Creation	A	10	
Support Resource Mgt.	S	5	Code Reuse
Fix Notifications	S	5	Trivial
Total		**55**	

Example [Contd.]

3. Unadjusted Use Case Points Calculation

UUCP = UAW + UUCW

UUCP = 64 + 55 = 119

Example [Contd.]

4. Technical Factor Computation

Factor	*Description*	*Assigned Value*	*Weight*	*Extended Value*
T1	Test Tools	5	3	15
T2	Documented inputs	5	5	25
T3	Development Environment	2	1	2
T4	Test Environment	3	1	3
T5	Test-ware reuse	3	2	6
T6	Distributed system	4	4	16
T7	Performance objectives	2	1	2
T8	Security Features	4	2	8
T9	Complex interfacing	5	2	10
			Total	**87**

Example [Contd.]

5. Adjusted UCP Calculation

AUCP = UUCP * [0.65 + (0.01 * TEF)]

AUCP = 119 * [0.65 + (0.01 * 87)]

= 180.88

Example [Contd.]

6. Final Effort

Effort = AUCP * Conversion Factor for COM/DCOM component testing

Effort = 180.88 * 13 = 2351.44

Project Complexity compensation = 15%

Management activity = 10 %

Total Effort = 2351.44 + 352.72 + 235.144

= 2939.3 man-hours

= 367 man-days

Actual Effort = 390 man-days

Conclusion

- Method worked for web project, results for other projects not guaranteed.
- Significant practical advantage if UC approach is in use.
- Further research needed to fine tune the method e.g. TEF table
- Yields accuracy only with historical data over a period of time.

Further Reading

- Capers Jones, “Applied software measurement”, McGraw-Hill,1996.
- Alistair Cockburn, “Writing Effective Use Cases”, 1999.
- Smith John, “Estimation of effort based on Use Cases”, Rational Software
- Dekkers Ton, “Test Point Analysis”, 1999

Thank You!

Contact Information

Suresh Nageswaran

Cognizant Technology Solutions, Pune.

Tel : 91-20-6691960 Ext 2267

E-mail: SureshN@pun.cognizant.com

SureshN@iname.com

Cognizant Technology Solutions

Test Effort Estimation Using Use Case Points

Suresh Nageswaran
Cognizant Technology Solutions,
National Games Road,
Yerwada, Pune – 411006.
Maharashtra, India
(+91-020) 669 19 60
SureshN@pun.cognizant.com
SureshN@iname.com

Abstract

This paper presents a new approach to the estimation of software testing efforts based on Use Case Points [UCP] as a fundamental project estimation measure. From preliminary applications on our web-based projects, we conjecture that this could in fact be more reliable than FP. The caveat here is that the V-model must be in use and use case generation must start becoming available right at the requirements gathering phase. The acceptance test plan is then prepared with the use cases from the requirement documents as input. Further work could provide a more exact relationship between the two.

Introduction

Probably the most crucial difference between the manufacturing industry and the software industry is that the former is able to stick to schedules most of the time. The reason why software development schedules are so unpredictable is not because workers in this industry are lazy or incompetent. To estimate the time make a product from scratch, and in many cases, without prior experience of the technology is no mean feat. However, conventional estimation techniques address only the development effort that goes into it.

It is known that a use case to test case mapping is possible. This means that the UCP figure for development can be indirectly used to provide a figure for the number of test cases. Using organizational test execution time metrics it is now possible to arrive at a figure for the total test effort. This is a viable and systematic approach towards test effort estimation and it makes a leap in providing more realistic figures. This means that the cost of testing can now be factored into projects. The other advantage is that test engineering gets treated as a process and not simply as another lifecycle phase.

Software Test Engineering

Test Engineering covers a large gamut of activities to ensure that the final product achieves some quality goal. These activities must be planned well in advance to ensure that these objectives are met. Plans are based on estimations.

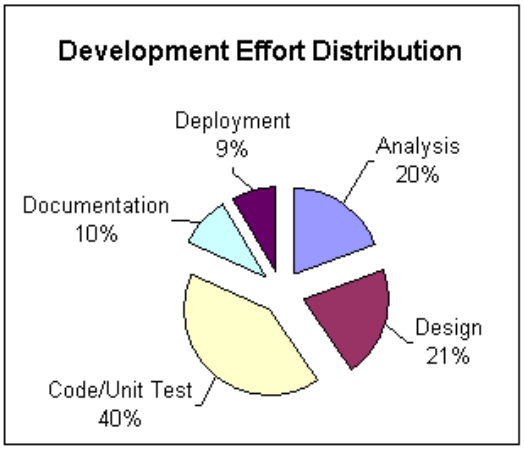
In the early years, the Waterfall model has been applied to software development. This model looks upon test engineering as merely a stage in the entire development lifecycle. When techniques evolved over the years for estimating development time and effort, the concept of estimating test-engineering time was overlooked completely.

Test engineering is seldom planned for in most organizations and as a result, products enter the market insufficiently tested. Negative customer reactions and damage to the corporate image is the natural consequence.

To avoid this, the correct development lifecycle must be chosen and planning should be done early on in the cycle.

Software Project Estimation

According to Rubin [2], the stage-wise effort distribution on software projects is as shown in the pie chart below.



Estimation is basically a four-step approach:

1. Estimate the size of the development product. This is either in LOC [Lines of Code] or FP [Function Points]. The concept of using UCP [Use Case Points] is still in its infancy.

2. Estimate the effort in person-months or person-hours.

3. Estimate the schedule in calendar months.

4. Estimate the cost in currency.

Conventional Approach to Test Effort Estimation

Test engineering managers use many different methods to estimate and schedule their test engineering efforts. Different organizations use different methods depending on the type of projects, the inherent risks in the project, the technologies involved etc.

Most of the time, test effort estimations are clubbed with the development estimates and no separate figures are available.

Here is a description of some conventional methods in use:

1. Ad-hoc method

   The test efforts are not based on any definitive timeframe. The efforts continue until some pre-decided timeline set by managerial or marketing personnel is reached. Alternatively, it is done until the budgeted finances run out.

   This is a practice prevalent in extremely immature organizations and has error margins of over 100% at times.

2. Percentage of development time

   The fundamental premise here is that test engineering efforts are dependent on the development time / effort. First, development effort is estimated using some techniques such as LOC or Function Points. The next step is using some heuristic to peg a value next to it. This varies widely and is usually based on previous experiences.

   This method is not defendable since it is not based on any scientific principles or techniques. Schedule overruns could range from 50 – 75% of estimated time. This method is also by far the most used.

3. From the Function Point estimates

   Capers Jones [1] estimates that the number of test cases can be determined by the function points estimate for the corresponding effort. The formula is

   Number of Test Cases = $(\text{Function Points})^{1.2}$

   The actual effort in person-hours is then calculated with a conversion factor obtained from previous project data.

   The disadvantage of using FP is that they require detailed requirements in advance. Another issue is that modern object-oriented systems are designed with Use Cases in mind and this technique is incompatible with them.

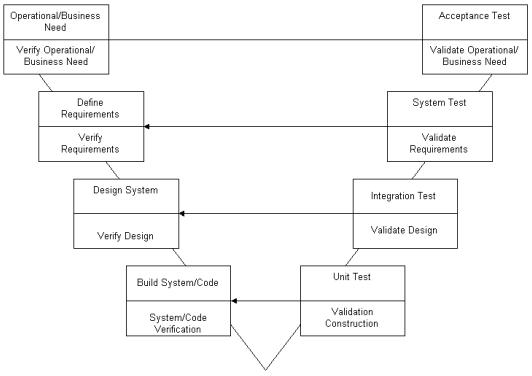
Use Cases

Alistair [5] has this description of a use case:

A use case captures a contract between the stakeholders of a system about its behavior. The use case describes the system's behavior under various conditions as it responds to a request from one of the stakeholders, called the *primary actor*. The primary actor initiates an interaction with the system to accomplish some goal. The system responds, protecting the interests of all the stakeholders. Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and conditions surrounding the requests. The use case collects together those different scenarios.

Mapping Use Cases to Test Cases

Use cases in their most primitive forms are basically representative of what the user wants from a system. The advantages of Use Cases are that they start becoming available early on in the project lifecycle. The appropriate project lifecycle model is the V-Model. The figure below illustrates the same.



The Expanded V Chart

The model clearly has one test engineering activity associated with a corresponding development activity. The topmost rung of the model associates the business requirement identification with the acceptance plan preparation. Each successive step makes sure that the test documentation becomes complete and comprehensive. If the estimation process is fitted in the second rung after the business requirements are available, it is obvious that use cases will serve as the inputs.
The identification of the number of test cases here can be made quite directly. Each scenario and its exception flows for each use case are input for a test case. Subsequently, the estimation calculations can commence.
As the requirements become clearer further downstream, the estimates will also undergo revision.

UCP Approach to Estimation

Estimation using UCP [Use Case Points] is rapidly gaining a faithful following. The approach for estimation using UCP only needs slight modification in order to be useful to estimate test efforts.

1. Determine the number of actors in the system. This will give us the UAW – the unadjusted actor weights.

   Actors are external to the system and interface with it. Examples are end-users, other programs, data stores etc.
   Actors come in three types: simple, average and complex. Actor classification for test effort estimation differs from that of development estimation.

   End users are simple actors. In the context of testing, end-user actions can be captured easily using automated tool scripts. Average actors interact with the system through some protocols etc. or they could be Data stores. They qualify as average since the results of test case runs would need to be verified manually by running SQL statements on the store etc. Complex users are separate systems that interact with the SUT through an API.

   The test cases for these users can only be written at the unit level and involves a significant amount of internal system behavioral knowledge.

Actor Weights

Actor Type	*Description*	*Factor*
Simple	GUI	1
Average	Interactive or protocol-driver interface	2

Complex	API / low-level interactions	3

The sum of these products gives the total unadjusted actor weights. [UAW]

2. Determine the number of use cases in the system. Get UUCW.

   The use cases are assigned weights depending on the number of transactions / scenarios.

Use-case Weights

Use Case Type	*Description*	*Factor*
Simple	<=3	1
Average	4-7	2
Complex	>7	3

The sum of these products gives the total unadjusted actor weights. [UAW]

3. UUCP = UAW + UUCW

   The calculation of the unadjusted UCP is done by adding the unadjusted actor weight and the unadjusted use case weights determined in the previous steps.

4. Compute technical and environmental factors

   The technical and environmental factors for a test project are listed in the table below.

   To calculate one needs to assign weights and multiply them with the assigned values to give the final values. The products are all added up to give the TEF multiplier. The TEF multiplier is then used in the next step.

Technical Complexity Factor

Factor	*Description*	*Assigned Value*
T1	Test Tools	5
T2	Documented inputs	5
T3	Development Environment	2
T4	Test Environment	3
T5	Test-ware reuse	3
T6	Distributed system	4
T7	Performance objectives	2
T8	Security Features	4
T9	Complex interfacing	5

5. Compute adjusted UCP.

   We use the same formula as in the UCP method for development.

   AUCP =UUCP *[0.65+(0.01*TEF)]

6. Arrive at final effort.
   We now have to simply multiply the adjusted UCP with a conversion factor. This conversion factor denotes the man-hours in test effort required for a language/technology combination. The organization will have to determine the conversion factors for various such combinations.

   E.g. Effort = AUCP * 20
   Where 20 man-hours are required to plan, write and execute tests on one UCP when using EJB.

Example

The project under study is a product support web site for a large North American software company. The estimation was done from the business level use cases made available at the time of signing the requirements. The actors at this time were the different types of users identified in those use cases.

1. UAW Calculation

Actor	*No of Use Cases*	*Factor*	*UAW*
B2C User	15	2	30
Subscribers	13	2	26
Admin User	4	2	8
Total UAW			**64**

2. UUCW Calculation

Legend: Simple – S, Average – A, Complex – C, Very Complex - VC

Use Case	*Type*	*Factor*	*Reason*
Login	C	15	Server integration
Support Request	VC	20	External Sys Query
User Creation	A	10	
Support Resource Mgt.	S	5	Code Reuse
Fix Notifications	S	5	Trivial
Total		**55**	

3. Calculation of the UUCP - Unadjusted Use Case Points
UUCP = UAW + UUCW = 64 + 55 = 119

4. Technical factor computation

Factor	*Description*	*Assigned Value*	*Weight*	*Extended Value*
T1	Test Tools	5	3	15
T2	Documented inputs	5	5	25
T3	Development Environment	2	1	2
T4	Test Environment	3	1	3
T5	Test-ware reuse	3	2	6
T6	Distributed system	4	4	16
T7	Performance objectives	2	1	2
T8	Security Features	4	2	8
T9	Complex interfacing	5	2	10
			Total	**87**

5. Adjusted UCP calculation

AUCP =UUCP *[0.65+(0.01*TEF)] = 119 * [0.65+0.01*87] = 180.88

6. Final Effort

Effort = AUCP * Conversion Factor for COM / DCOM testing
Effort = 180.88 * 13 = 2351.44

Project Complexity needs 15% of the estimated effort to be added. 10% is spent in co-ordination and management activity.

Total Effort = 2351.44 + 352.72 + 235.144 = 2939.304 man-hours = 367 man-days

Actual Effort = 390 man-days [Project End]

Discussion And Future Work

There is never a single silver bullet for every problem. In the many approaches to test effort estimation, the UCP approach is one. The author conjectures that this could become a more robust method of estimation over a period of time. The availability of data from past projects will definitely contribute to the accuracy of these estimates. The estimation technique is not claimed to be rigorous, but the approach offers significant practical advantages over ad hoc techniques currently in use. Further research and experimentation will certainly provide more substantial benefits in arriving at an objective method to validate the estimates.

References

1. Capers, Jones 1996. Applied software measurement, McGraw-Hill.
2. Rubin, H. 1995. Worldwide benchmark project report, Rubin Systems Inc.
3. Kathleen Peters, 1999, Software Project Estimation**.**
4. Smith John, The estimation of Effort based on Use Cases, Rational Software.
5. Cockburn Alistair, 1999, Writing Effective Use Cases.
6. Dekkers Ton, 1999, Test Point Analysis.

QW2001 Paper 6T1

Dr. Rainer Stetter
(ITQ Gmbh & Software Factory Gmbh)

Test Strategies for Embedded Systems

Key Points

- Real life example (control system), project running from 1997 to 2000
- Detailed discussion of every project phase
- Recommendation of approaches, measures and tools

Presentation Abstract

Embedded Systems are used more and more in the field of mechanical engineering. This situation leads to an increasing demand for efficient test strategies for Embedded Systems. In my presentation I would like to demonstrate our approach with a real life example. In the example IËll discuss the test strategy for a PC104 based control system for a material testing machine.

About the Author

At the Technical University, Munich I studied mechanical engineering and as I was interested in software engineering I took some classes in computer sciences. While I was doing my PhD in developing a robot simulation system, which I got in 1993, I improved my knowledge in software engineering. From 1993 until 1997 I was working as a Research & Development Manager at Zwick Company, Ulm - Germany.

Since 1997 I have been one of the General Managers of Software Factory GmbH, Munich. In addition, since 1998 I have been working with the Munich based firm itq GmbH as a General Manager. Together, Software Factory GmbH and itq GmbH form the Software Quality Center. With some partners of the Technical University of Munich and VDMA (German Machinery and Plant Manufacturers' Association), we work on approaches to improve the quality especially in the field of embedded systems.

Since 1998 I'm Vice President of the VDMA Software department.

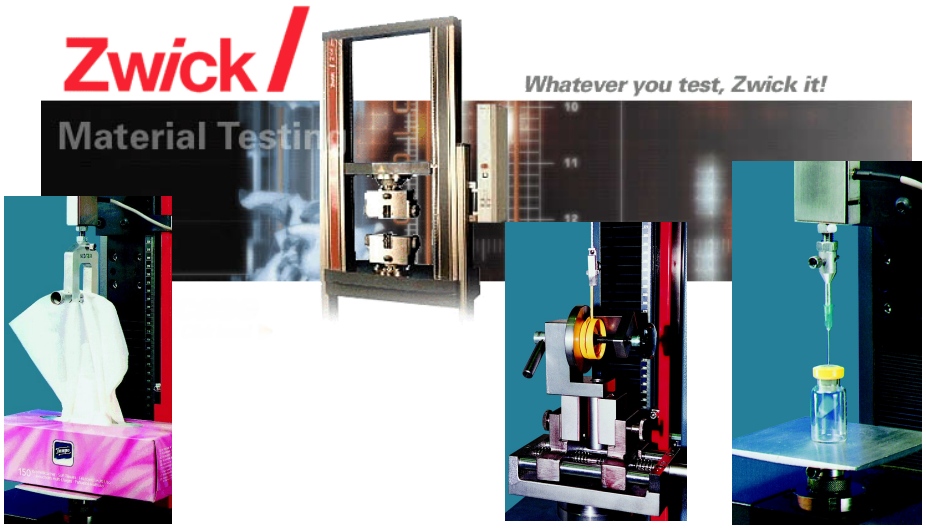
ITQ Information: Technology: Quality

SOFTWARE SF FACTORY

Test Strategies for Embedded Systems

Dr.-Ing. Rainer Stetter
www.itq.de

Example: Material Testing



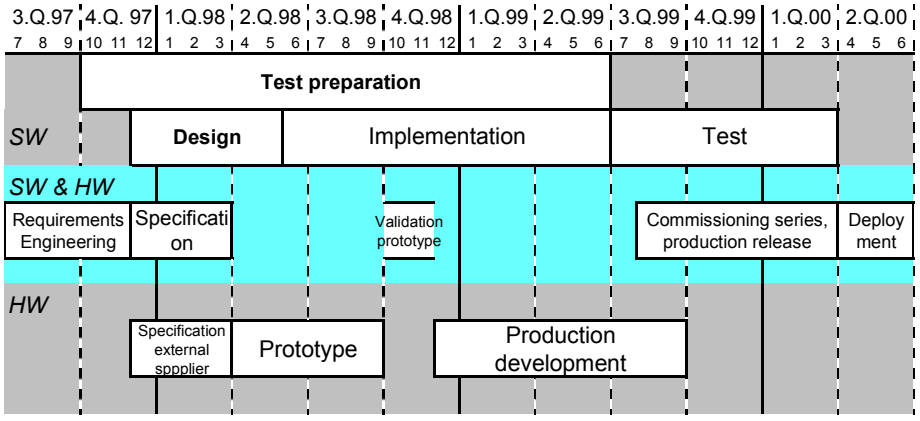
Quality Requirements

- Safety critical
  - Medical devices
  - Life and health
- Quality Requirements according to the FDA
  - stable and detailed process
  - Validation plan

Phases of the Validation Plan

- Split whole process into different phases
  - Requirements Engineering
  - Specification/Design
  - Implementation
  - Test
    - Module
    - Integration
    - Acceptance
    - System
  - Deployment

Time schedule (first version)



Extract of a validation plan

	Validation activity	Rating				Reference	Responsible
		1	2	3	4		
1.	Definition of the user functions of the software		x			..\..\..\Analyse\review\A1-Funktionen der SW.doc	uss-kr
2.	Definition of the required input data		x			..\..\..\Analyse\review\A2-Eingaben der SW.doc	uss-kr
3.	Definition of the required output data			x		..\..\..\Analyse\review\A3-Ausgaben der SW.doc	uss-kr
4.	Definition of limits, defaults, special input which have to be accepted by the software		x			..\..\..\Analyse\review\A4-Bereiche für Ein-Ausgaben.doc	uss-kr
5.	Definition of the required performance		x			..\..\..\Analyse\review\A5-Performance der SW.doc	uss-sj
6.	Definition of the external interfaces and the user interface		x			..\..\..\Analyse\review\A6-Schnittstellen der SW.doc	uss-hib
7.	Definition of error classes		x			..\..\..\Analyse\review\A7-Definition von Fehlerklassen.doc	uss-kr
8.	Definition of the reactions on error classes		x			..\..\..\Analyse\review\A8-Reaktion auf Fehler.doc	uss-kr
9.	Definition of the system environment		x			..\..\..\Analyse\review\A9-Betriebsumgebung.doc	uss-sj
10.	Definition of safety requirements, functions and features	x				..\..\..\Analyse\review\A10-Sicherheit der SW.doc	uss-bi
11.	Software hazard analysis		x			..\..\..\Analyse\review\A11-SW-Gefahrenanalyse.doc	uss-bi
12.	Traceability of system and software requirements		x			..\..\..\Analyse\review\A15-Quercheck-System-SW.doc	uss-sj
13.	Design of the system test plan			x		..\..\..\Analyse\review\A17-Systemtestplan.doc	sf-rs
14.	Design of the acceptance test plan			x		..\..\..\Analyse\review\A18-Abnahmetestplan.doc	sf-rs

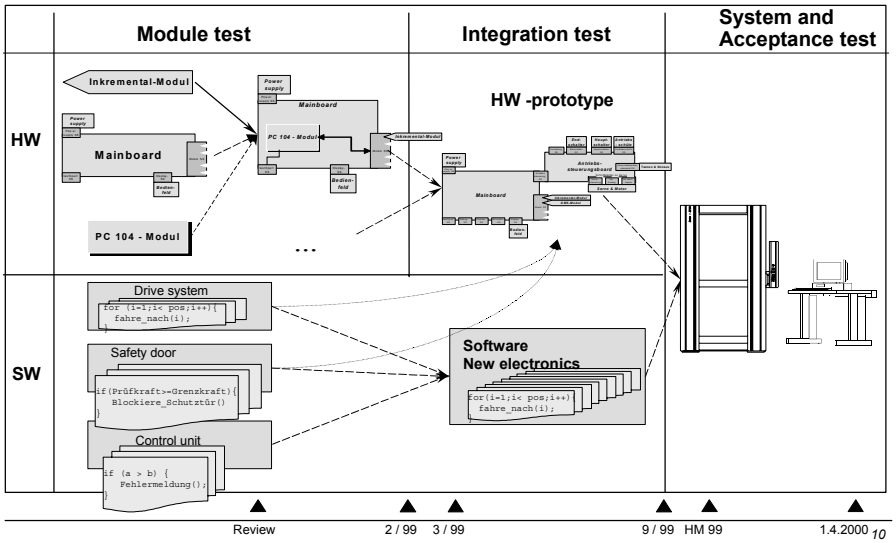
Requirements Engineering

- Descripition of Requirements
  - Prose – Non-formal (50 pages)
    - Formalisation subsequently, but not detailled enough
  - both forms are sensible
    - Prose – Management
    - Formalized – Test
- Hazard Analysis
  - of the system with special focus on software
- Test plans
  - Almost impossible to do in a sensible way
- Trial and error group

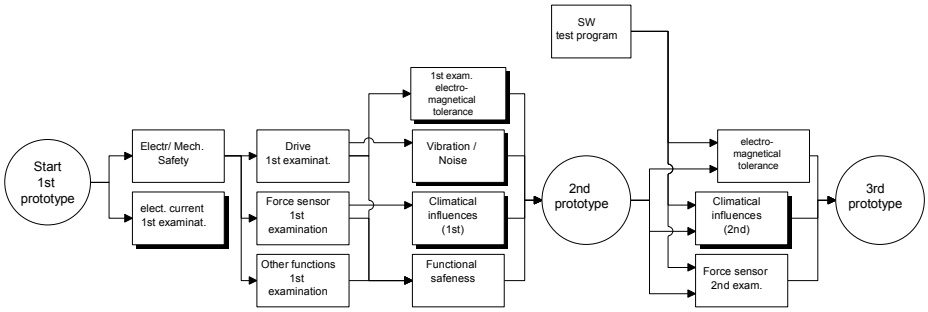
Specification and Design

- Specification
  - One modularized document (500 pages) in prose
    - No additional work for hardware group
  - Written from developers for developers
    - Inconsistent level
    - Not formal enough
    - Hard to develop test procedures
- Test
  - Structure and elements of tests were designed
  - Planning of the integration test
- Software design
  - Guideline for software design
  - UML with weekly design reviews

Structure of the test activities



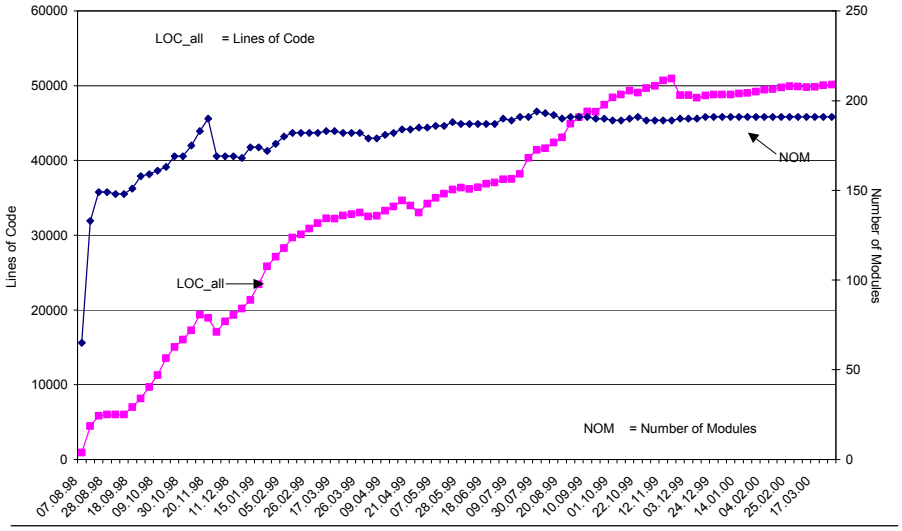
Structure of integration test plan



Implementation

- Organisation
  - Daily check in and build
- Weekly code reviews
  - Social aspect
    - Code is no longer a personal secret
    - Same philosophy / style of coding
  - Technological aspect
    - Only low level errors
      - Reading was too fast -> Tom Gilb
- Test
  - Development of test procedures and approaches

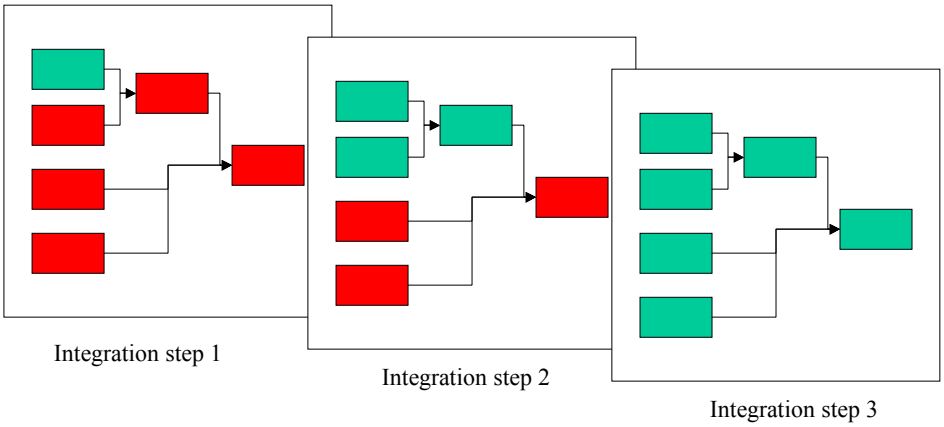
Number of Modules and Lines of Code vs. Time



Integration test

- Theory and practice in planning
  - Delays of software or hardware
    - Very hard time
    - A lot of improvisation
- Design of test plans
  - Have to be modular -> flexibility to adapt to new situation
- Control
  - Regular tracking is very important
  - Visible and understable even for the management
- Code Coverage (starting with module testing)
  - Whole system not testable at once
  - Had to be divided into parts

Tracking of integration test

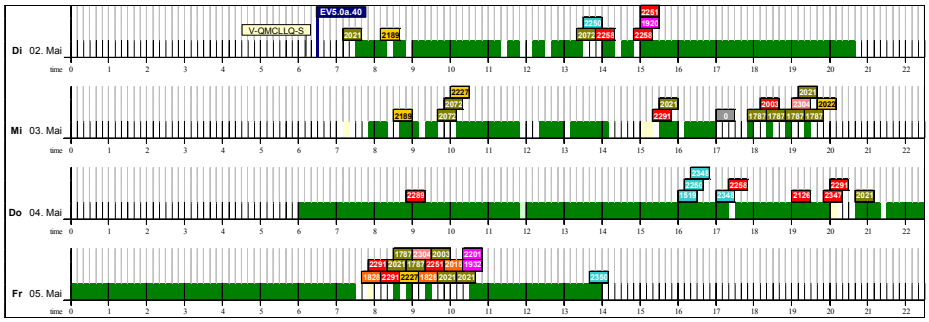


System and acceptance test

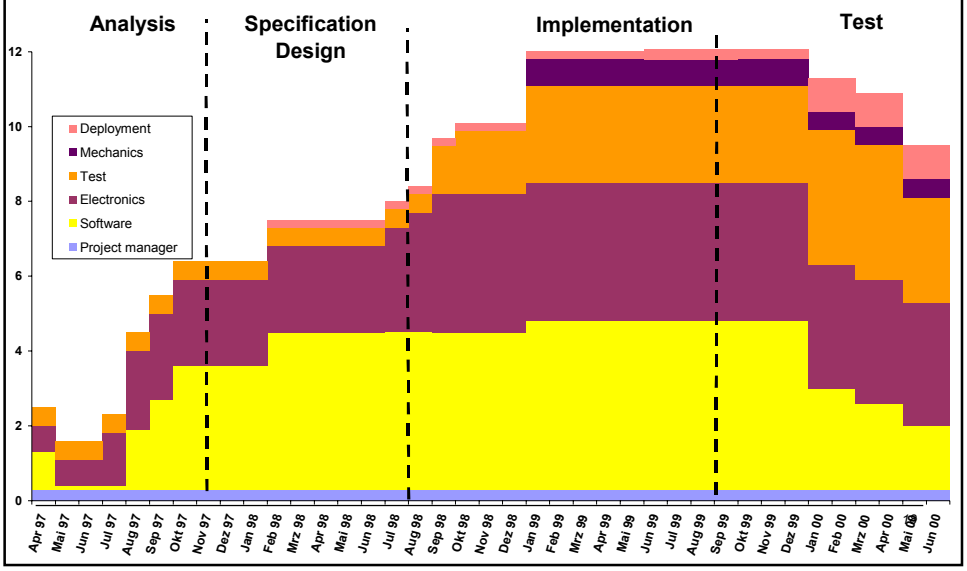
- Test environment
  - Most critical aspect
    - New hardware was not available in required numbers
    - Old machines had to be modified
  - System and acceptance test ran concurrent

- System reliability and defect tracking
  - Very detailled tracking
  - Wide ranging set of tests on technical aspects
    - Humidity
    - Lack of electronic current
    - Different temperatures
    - Safety

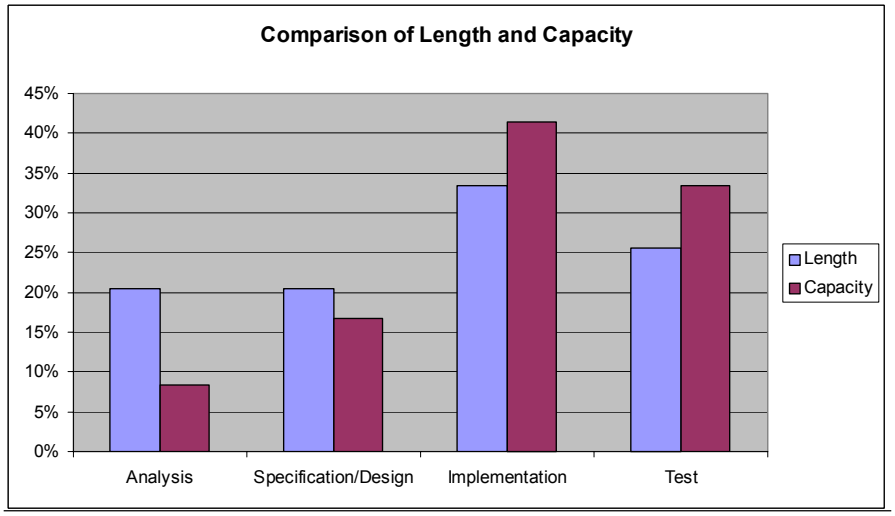
Tracking of the tests



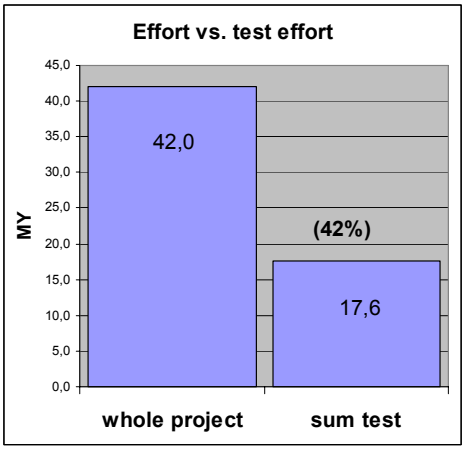
Overview of efforts

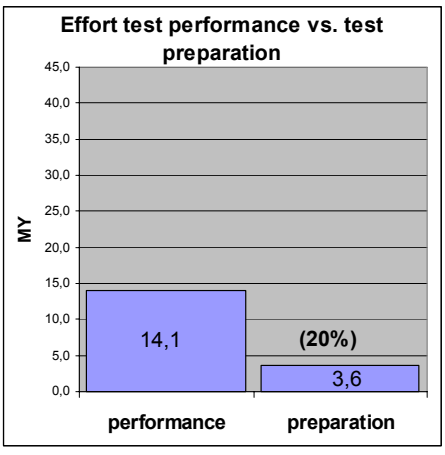


Length of phases vs. manpower in phases



Test effort





Lessons learned (I)

- Validation plan
  - gave structure for the whole project
    - Most useful for the early phases
      - Responsibility for disliked activities
    - Some bureaucratic overhead
  - Reviews very helpful
    - Phases
    - Design and coding
- Design
  - UML is sensible
  - No completion of design without coding

Lessons learned (II)

- Test
  - To start test planning early is very hard, but sensible
    - Modular design of test/test procedure is very important
      - Re-useability of test patterns in critical situations
  - Tracking through all phases -> management
    - Test procedures
    - Defects and reliability
  - Environment
    - You never have enough

Conclusion

- Embedded Systems <-> inter-disciplinary teams
  - Need to encourage communication
    - Modules won‘t understand each other,
    - if people don‘t talk with each other
- To work in a structured and planned way
  - Doesn‘t prevent all problems
  - But makes it easier to handle them
- Think before you work !
  - Life will be easier

"Test Strategies for Embedded Systems"

Dr.-Ing. Rainer Stetter
ITQ – IT & Quality GmbH, Munich
http://www.itq.de

Introduction:

Embedded Systems are used more and more in the field of mechanical engineering. This situation has lead to an increasing demand for efficient test strategies for Embedded Systems. On the basis of a real life example I want to discuss our approach for testing embedded systems.

Example: Control System for Material Testing Machines

Material testing machines are used in a broad area. Any material has to be tested once before it's used in any construction. Therefore there is a high safety and security demand for the control system of a material testing machine. The control system which is to be discussed has to collect all machine data in real time which are used for the determination of the actual control parameters. The real time operating system used is VxWorks. The software runs on a standardized PC module with a 486 processor. The processor has to supervise ten input channels at the same time which are refreshed every two ms. For the design of the software we used UML, based on Rose98 including the code generation and the round trip engineering component.
The development process is to be validated by the FDA because this kind of machine is used for testing medical devices, too. According to the FDA demands we used a validation plan to manage all project activities. The validation plan describes all activities which have to be fulfilled in a project, including all phases from the requirements engineering up to the point of the product release.

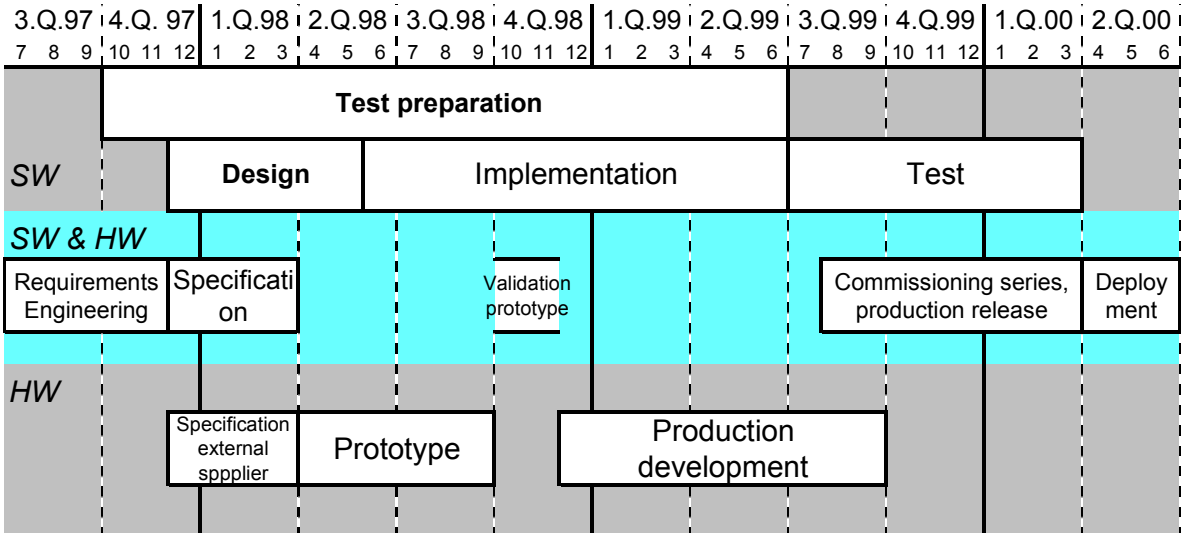


Fig. 1: Overview of the project schedule

Fig. 1 gives an overview of the project. The project was splitt into several phases. The preparation of the test started quite early and ran parallel to all other activities until the start of test phase.

According to the validation plan we had to work on a software hazard analysis in the **requirements engineering phase** to show how the system would react in the case of a severe software fault. This analysis influenced the system design in general. Because we couldn't guarantee an error free software system we had to provide a (mechanical) hardware backup component for any potential safety critical situation. Otherwise we would have had to use a two-processor system which was out of the question because of the additional costs. Another validation activity in this phase was planning the structure of the system and the final acceptance test. Even though you can read about software testing in almost any publication, that consideration of the test process should begin at a very early point in the project, we experienced that it is very hard to this in a effective way.

The description of the requirements was written in prose in a non-formal style. After a review of the requirements specification through the test group we formalized them subsequently. In the test phase we had to go through the experience that the level of formalization used wasn't detailed enough because the requirements were hard to test. To improve the testability of the requirements we will strengthen formalization. But you still need some requirements written in prose because it's hard to convince management or marketing people to read heavily formalized requirements.

After a review of the requirements engineering phase, in which all validation plan activities were discussed in detail, the design phase was started.

In the **specification and design phase** we had to determine in more detail which system function had to be implemented in hardware (electronics) or in software. In this phase it was quite difficult to keep the hardware and software team in touch. Both of these groups were ready to go into details even though their main job at that time was to determine the interfaces.

The specification document was written in prose, too. It contained about 500 pages. To admit concurrent working of several persons we organized the document to be modular. To write a specification document was neither new nor additional work for the hardware group. They were used to work in this way. To convince the software team not to start with more exciting things was quite hard. In the test phase we had to repeat the experience that our work could have been better. Once more we had found out that the specification should be more formal to make it easier to develop good test procedures and test cases, in addition we had to recognize that the level of abstraction was inconsistent.

For the software design we introduced UML. As a matter of fact, the use of the Rose98 was very helpful. The sequence diagrams were especially useful because in this view the dynamic behaviour of the system was quite easy to model. After a period of familiarization, the software engineers worked in small groups interactively with the tool. This was very efficient because the communication between the software engineers was highly

stimulated. In addition the software design was reviewed weekly. The design reviews were continuously attended by a member of the test team.

In this phase the test team worked on the design of all test activities. The structure and the interdependencies of the test activities are shown in fig. 2.

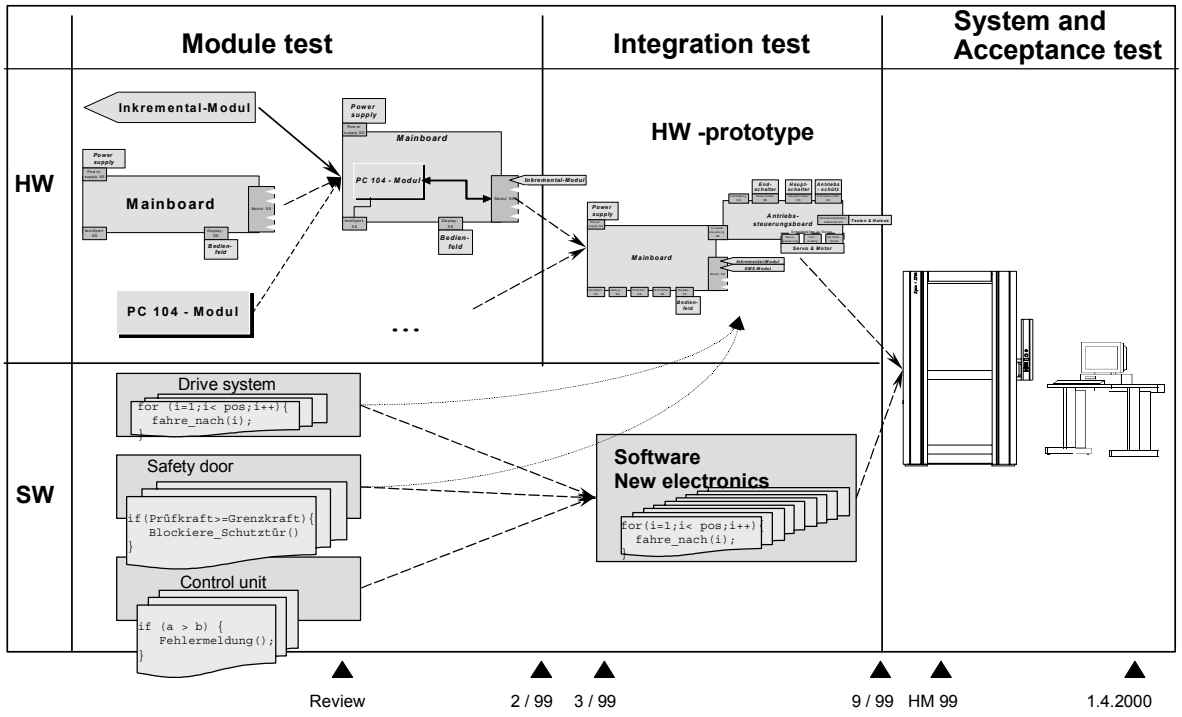


Fig. 2: Structure of the test activities

To describe a module test for hardware modules was quite easy. Due the OO-design of the software it was almost impossible to design a module test for the software. Therefore we decided to run code reviews and to do code coverage tests through the implementation phase to reduce risks and improve software quality. According to the module test plan an initial version of an integration test plan was developed. For the visualization of the interdependencies of the modules and the different test procedures we used a flow chart graphic, see fig. 3.

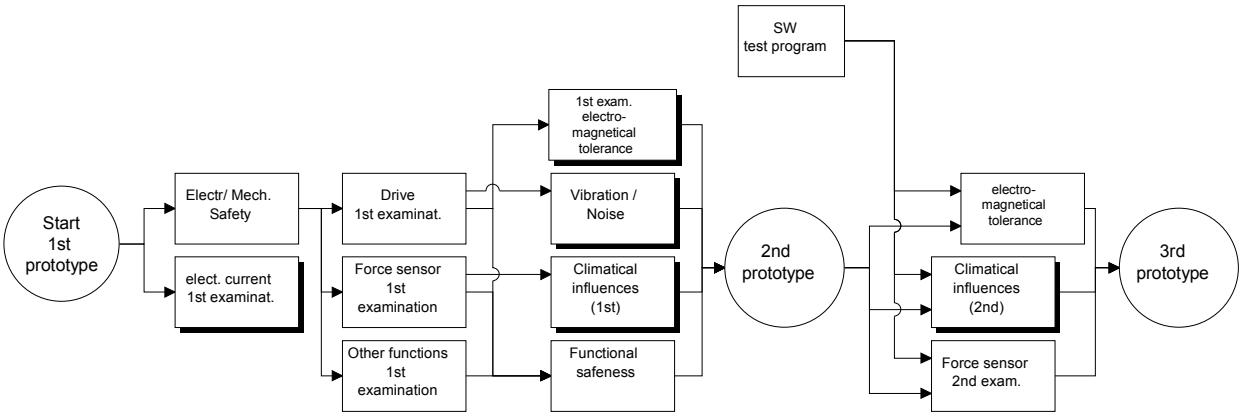


Figure 3: Structure of the integration test plan

During the **implementation phase** the different levels of the test plans were detailed. The weekly code reviews were attended by a member of the test team, too. Together with the software engineers the test team worked on the preparation of the white box tests. For the calculation of the code coverage we used CodeView. At the beginning of the project we planned an almost 100% coverage. After a while we figured out that it was more helpful to concentrate on some critical modules than to try to do all at once.

At the beginning of the implementation phase we thought that we already designed around 90 % of all needed classes (modules). But we had to figure out in the first weeks of the implementation that there was still a lot of detail work to be done. Therefore there is a steep increase of the number of classes in fig. 4 at the very beginning. After some weeks there was a stabilization of the number of modules. On the other hand there was a more or less steadily increasing number of lines of code.

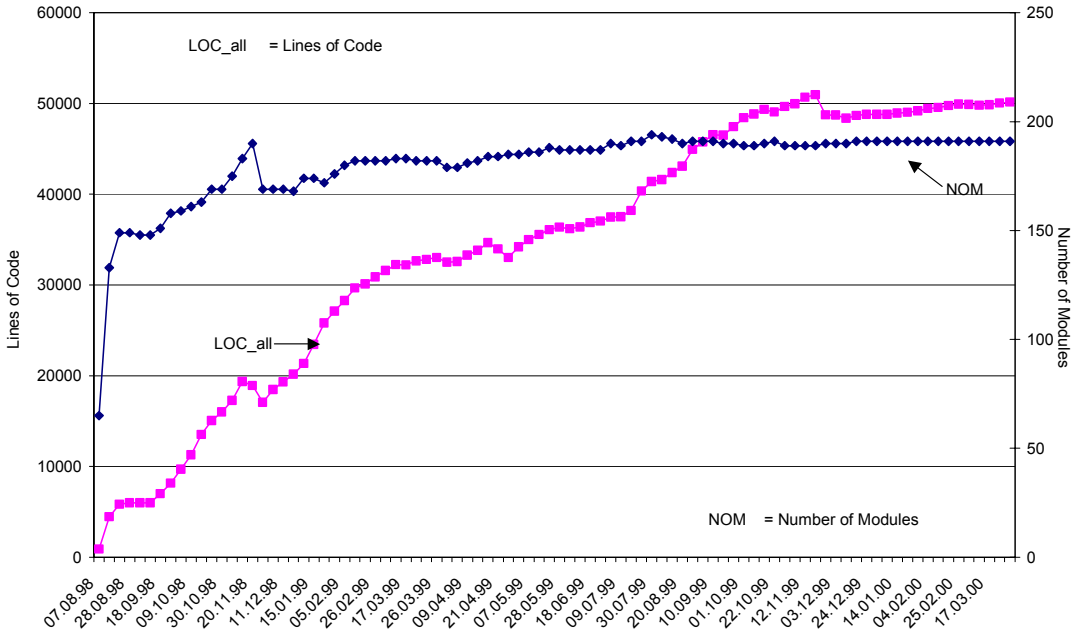


Fig.4: Number of modules (classes) and lines of code vs. time

The **test phase** started with the testing of some hardware modules. This period was quite tough because on one day the situation occurred that a hardware module was completed but the corresponding software wasn't finished. On the next day for another module the software was finished but the hardware was delayed. Looking back over this period we had to learn to accept that even with the best planning some improvisation can't be avoided. To be flexible in this situation you have to design your test procedures quite modularly. It has been our experience that the templates which are suggested by the IEEE were a good basis for tailoring our own templates.

After getting to a certain level of maturity in the basic modules we started with the integration of the system. To track the progress of the integration we used a coloured flow chart graphic.

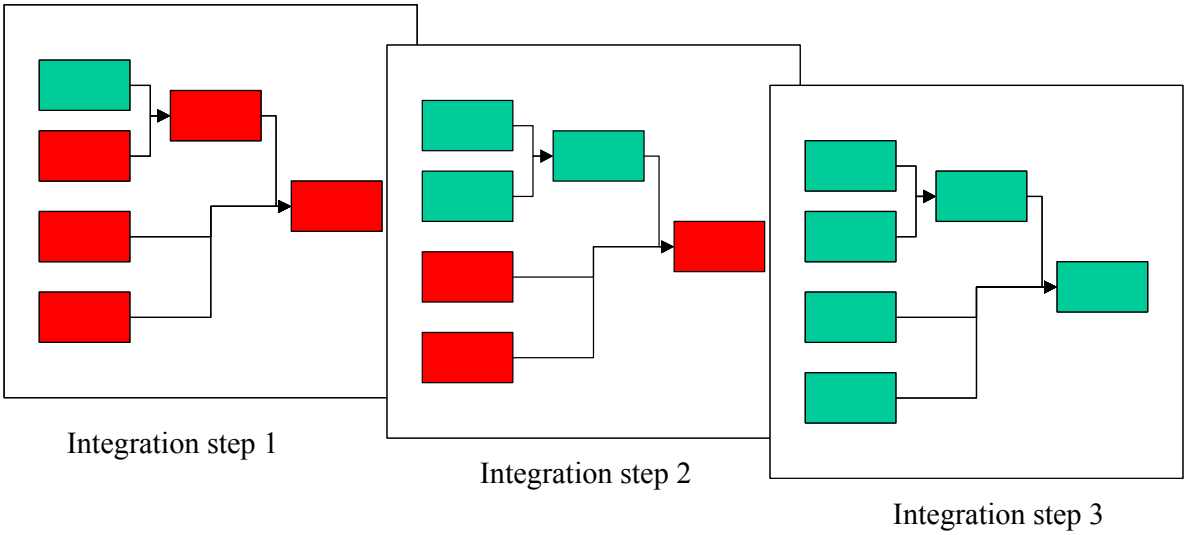


Fig.5: Progress tracking of the integration
(released modules are shown in green, unreleased modules are shown in red)

The basic idea of our approach is shown in fig. 5. For every module which is represented by a rectangle in the plan we tracked general information, such us the version, the date of release, the name of the person who released the module and the test procedures which were the basis for the release of the module. In addition we defined in advance which modules had to be tested in each integration step. In matching every integration step there were some integration test procedures which had to be performed. Through this visualization we always had a good overview of the actual situation. This was especially helpful in discussions with the management. After getting to the final point of the integration test we started with the system test. To do the system test we used a modified machine from a previous generation. After some weeks we got a pre-release of the new hardware module (mechanical and electronic components) so we could start the acceptance test concurrently with the system test.

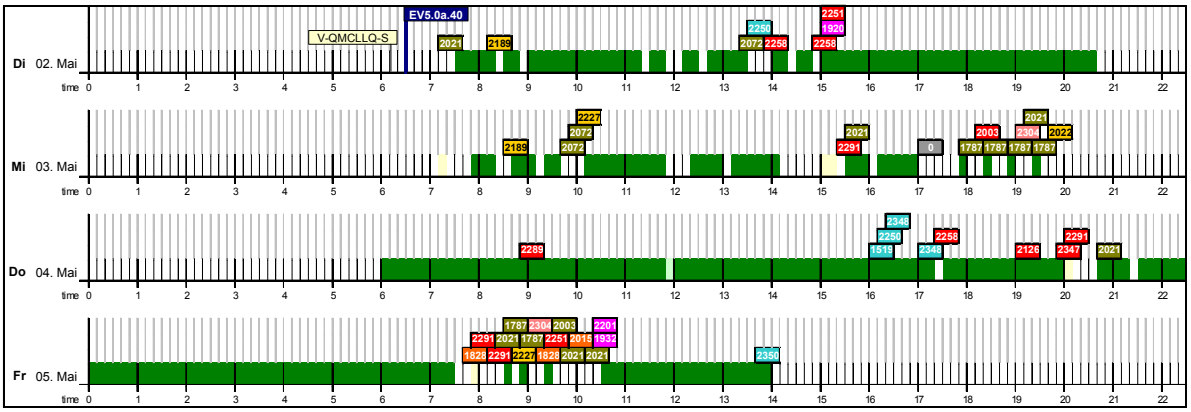


Fig 6: Tracking of the system performance and the occurrence of defects

The system performance and the defect rate were tracked by a spreadsheet with the format shown in fig. 6. The green colour shows that the system ran without any problems. All defects are marked by a rectangle. The colour of the rectangle identifies the severity of the defect and the most likely contaminated module. The number in the rectangle refers to the defect identifier in the error database.

Based on this type of spreadsheet and some other interpretations of the test results the test team could come to the decision whether or not the product was ready to release.

Effort:

The whole project was designed to learn how to manage projects of this type. One of the key questions of project management is, which effort has to be invested in which phase and of which team. According to this goal we measured not only technical aspects but also some management aspects. Figure 6 shows the progression of the ongoing project. The effort of the project staff was measured in percentage of theoretically available man power. We found out that a realistic percentage of real working on the project is for developers about 60-70%.

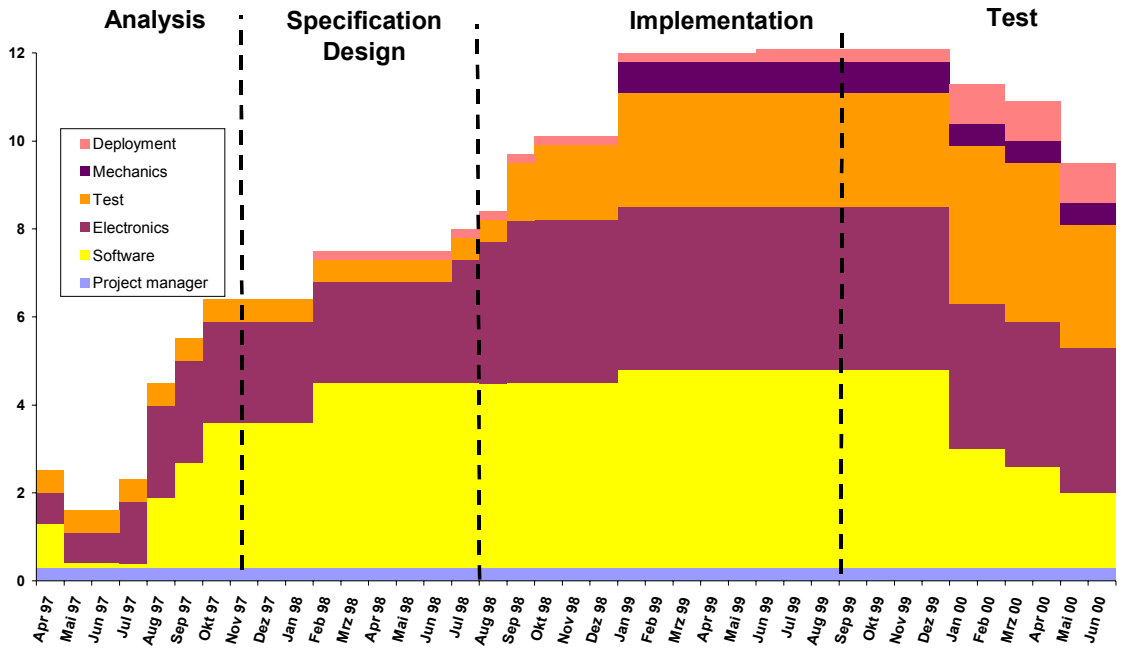


Figure 6: Overview of the efforts

The shape of the curves indicates that the product release worked quite well because there is a decline of the effort at the end of the project. Projects which end in a crisis have normally a steep increase in the effort by about two thirds of the theoretically estimated end date.

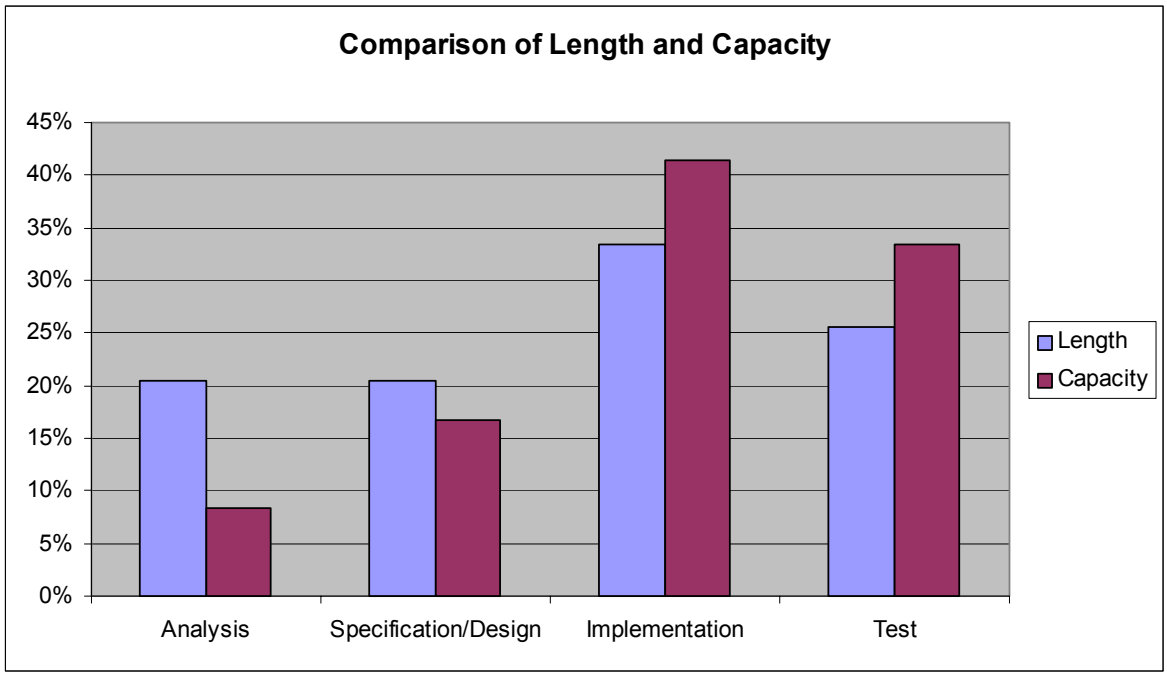


Figure 7: Length of phases vs. manpower in phases

Fig. 7 compares the lengths of the periods of time with the manpower invested in the different phases. The first phases seem to be quite long, but a look at the percentage of the manpower spent shows that the first phases needed only a small portion of the whole effort. We spent about 25% of the overall effort for requirement engineering and design but needed about 40% of the time. We think this ratio is quite realistic, because there is a big demand for interdisciplinary discussion in the first phases. These discussions are the basis for decisions and important decisions normally need time.

Finally we compared the test effort with the overall effort. In addition we figured out how much we invested in test preparation and test performance. The results are shown in fig. 8. The overall effort was about 42 man years. In test activities we spent about 42%. 20% of the test effort was needed for preparation, the rest to perform the tests.

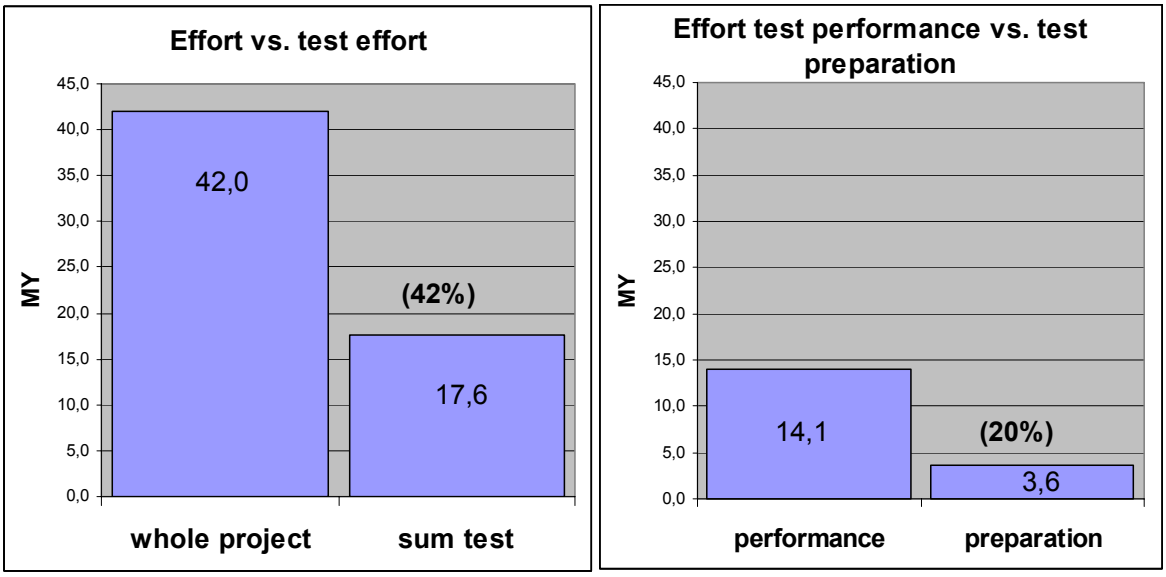


Figure 8: Test effort

Summary:

The project described ran from July 1997 to completion some months ago with a delay of only about 4 weeks. This result underlines the fact that collateral test activities conducted through the whole project are very helpful. But on the other hand it demonstrated that it may be quite hard to do meaningful test planning in a very early phase of the project. It's fairly predictable that the basic test approach might be quite wrong. Therefore it's very important that the structure of the test documents is modular to give the flexibility of adapting your test procedures very quickly to real circumstances.

Finally, we can conclude that the strategy of using a validation plan was very sensible. Firstly it gave the whole project a certain framework and some regulations. Secondly it forced us to start our test activities in a very early phase of the project which was eventually one of the preconditions for project success.

We spent a long period of time and a lot of money to prepare and plan the project in a sensible and serious way. In the eyes of the management this philosophy takes too much time for things which you can't see. Therefore it's very important to inform the management regularly and in a way which is understandable for non-insiders. Otherwise you are quite soon forced to work on getting understandable results rather than to spend time for thinking. Even though everybody knows that life is easier if think before you act.

QW2001 Paper 6T2

Mr. Keith B. Stobie
(BEA Systems, Inc.)

Automating Test Oracles and Decomposability

Key Points

- Patterns for verifying test case results: external vs. internal checking
- Patterns for how to check: external vs internal data (data driven testing)
- Test logging strategies

Presentation Abstract

All of the test patterns given below deal with test result handling.

The first patterns are complementary patterns for solving the same problem. They deal with internal (in-line check) versus external (batch check) checking of the results and internal (hard-coded) versus external (data driven) inputs and outputs. Batch check (or Benchmark) is particularly useful for changing the Judging pattern into Solved Example.

The second patterns are supplemental Test Automation Designs. Separable Tests (or Atomic Tests) deals with selection and independence of tests. All behavior (or whole function) deals with post-conditions of an Item Under Test (IUT).

All of these patterns also address test logging strategies.

About the Author

Keith directs and instructs QA and Test process and strategy. He plans, designs, and reviews software architecture and tests for BEA. His most recent project was BEA WebLogic Collaborate and previously WebLogic Exterprise. Keith was Test Architect at Informix designing tests for the Extended Parallel Server product and Manager of Quality and Process Improvement. With over 20 years in the field, Keith is a leader in testing methodology, tools technology, and quality process. He is a qualified instructor for Systematic Software Testing and software inspections. Keith is active in the software task group of ASQC, participant in IEEE 2003 and 2003.2 standards on test methods, published several articles and presented at many quality and testing conferences. Keith is an ASQ Certified Software Quality Engineer with a BS from Cornell University.

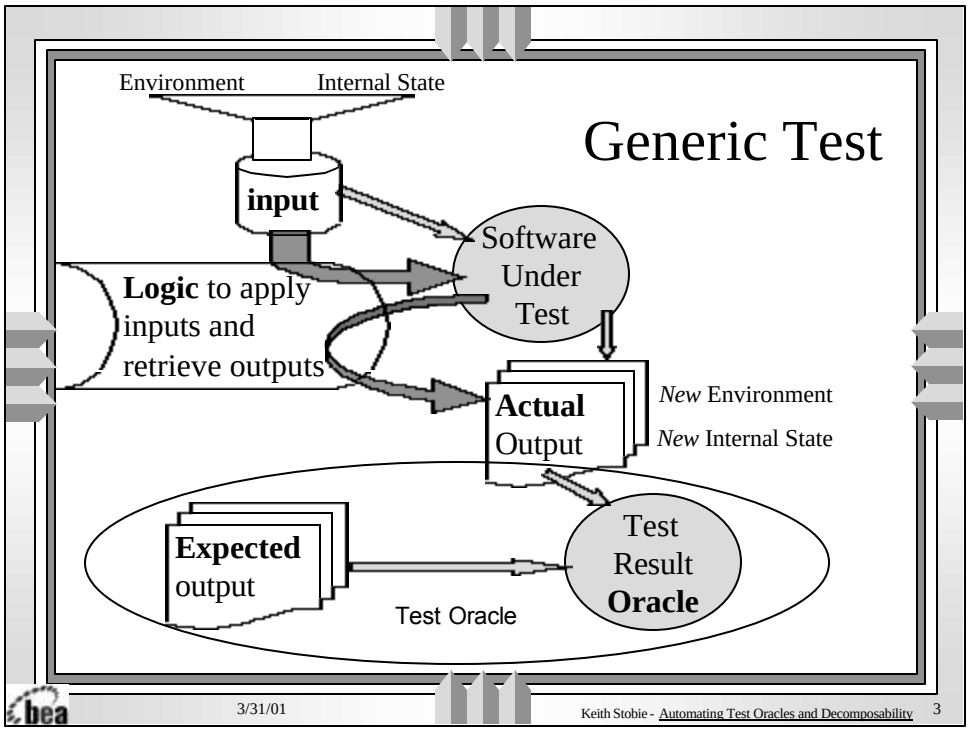
Automating Test Oracles and Decomposability

Test Patterns for
Test Result Handling

Keith Stobie
BEA Systems, Inc.

Introduction

Test Oracles	A means to determine Pass/Fail
Decomposability	A means to break down tests to smaller, more separable pieces.
Patterns	A way of describing a solution in a given context.
Anti-pattern	Tells how to go from a problem to a bad solution
Test Result Handling	How your *store*, *compare*, and *record* test results (expected and actual outputs)



Test Data Location patterns

Input Cause	Output – Effect		
	Expected	Actual	***Pattern*** / Anti-Pattern
Internal	Internal	Internal	***Self-contained data***
Internal	Internal	External	***Move actual to Internal***
Internal	External	Internal	Cause without Effect
Internal	External	External	Cause without Effect
External	Internal	Internal	Effect without Cause
External	Internal	External	Effect without Cause
External	External	Internal	***Move actual to External***
External	External	External	***Data-driven data***

Self-contained data

Context: You are creating a reusable test.

Problem: Where do you keep the test input data and test expected output data?

Forces:

- Want to read and understand a test with as few external references as possible.
- Tests are each relatively unique in their parameters (or sequences of actions).

Indications: The amount of input and output per result is relatively small and easy to understand.

Solution: Include the data in the test script or test code.

Self-contained data

Rationale: Test script logic becomes less complicated or more obvious when it is included directly with the logic.

Resulting Context/Consequences

Make it impossible to lose part of test, if it is only in one file.

Maintainability is sometimes reduced if bulk updates of expected results are needed.

Reduces code reusability if inputs and results are hard-coded or expressed as symbolic constants in the code.

Examples/Known Uses

Typically used in API tests, for example Posix Verification.

Code Samples: See *Check as you Go* using ***Self-contained data.***

Move actual to Internal

Run DB lock test (internal)

```
for I=1,numlocks {getDBlock();}
try { getDBlock(); logFail(); }
  catch (TooManyLocksException e){/*Got expected exception*/}
  catch (Exception e) { logFail(); }
```

diff DBlog expectedDBlog (external)

Run DB lock test (internal)

```
for I=1,numlocks {getDBlock();}
try { getDBlock(); logFail(); }
  catch (TooManyLocksException e){/*Got expected exception*/}
  catch (Exception e) { logFail(); };
System.exec("diff DBlog expectedDBlog");
```

See also ***Smart Dependency Checking***

Effect without Cause

Input file:

```
set onn   # should fail because it should be "set on"
set Off   # should succeed: Off should be case insensitive
```

Actual & expected outputs:

Effect without cause

```
Illegal Set argument
```

Expected output: **Effect with Cause**

```
set onn   # should fail because it should be "set on"
Illegal Set argument
set Off   # should succeed: Off should be case insensitive
```

Actual output:

```
set onn   # should fail because it should be "set on"
set Off   # should succeed: Off should be case insensitive
Illegal Set argument
```

Data-driven data

Context: You are creating a **reusable test** and you have **many** data **combinations** to be tested.

Problem: Where do you keep the test input data and test expected output data?
Hard-coding data in test scripts makes it laborious to create lots of related tests.

Forces:
- Output is voluminous.
- Output is difficult to predict and can frequently change from release to release.
- Additional, similar tests may need to be added.

Indications: 1) Test data is to be provided externally
2) You have existing legacy data.

Data-driven data

Solution: Separate test data from scripts.
This makes it easier to create multiple related tests.

Rationale: Separating data from procedure is a classic computer science technique for structuring code.

Resulting Context/Consequences

Mentally tracing through the test requires an extra level of indirection to substitute the data-driven values in the specific situation. Test script logic becomes more complicated or less obvious when data is separate.

Examples/Known Uses: Compiler tests.

SQL optimizer tests showing optimizer strategy (changing release to release).

Stub generation for distributed methods (for example CORBA IDL, or Java RMI).

Code Samples: See *Batch Check* using ***Data Driven Data*.**

Move actual to External

```
Compile <testInput >actualOutput
if [ $? != 0 ] ; then logFail(); #Internal
diff actualOutput expectedOutput # External
```

```
Compile <testInput >actualOutput
echo "ResultCode $?" >>actualOutput
diff actualOutput expectedOutput # External
```


Cause without Effect

Input file: 1 4 9 –1 0

Test code:

```
For I=1 to 3; do get num;
   if (square(squareRoot(num)) != num)
      print "fail $num";
done
For I= 1 to 2; do get num;
   if (squareRoot(num) != "illegal")
      print "fail $num";
done
```

Cause with Effect

Input file:

```
 1   1
 4   2
 9   3
-1   illegal
 0   illegal
```

Cause Effect

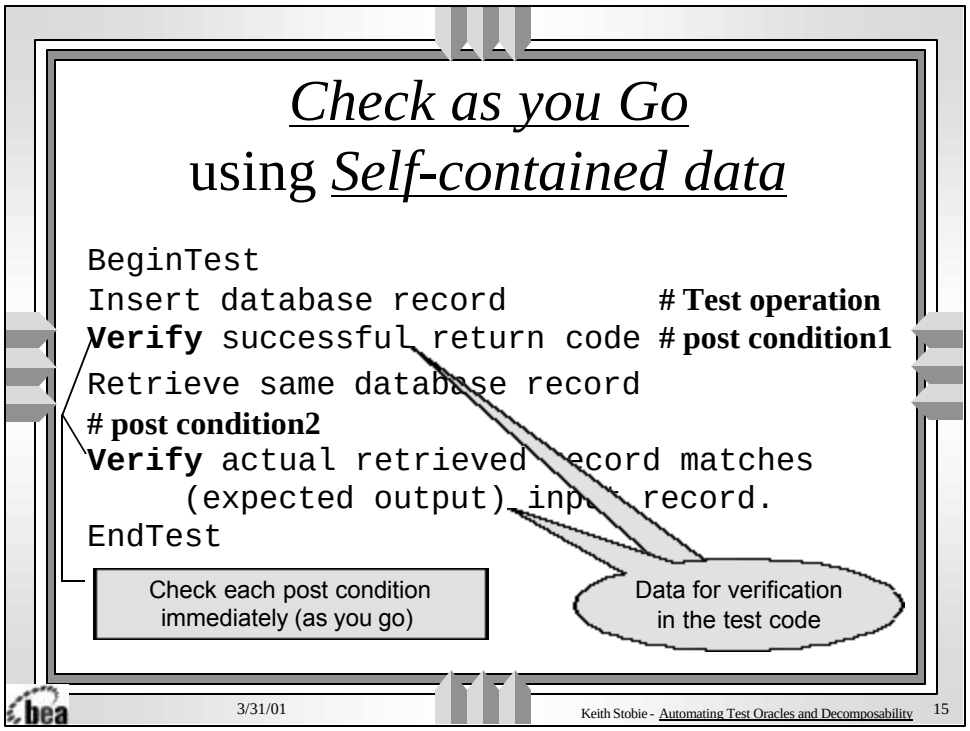
Test code:

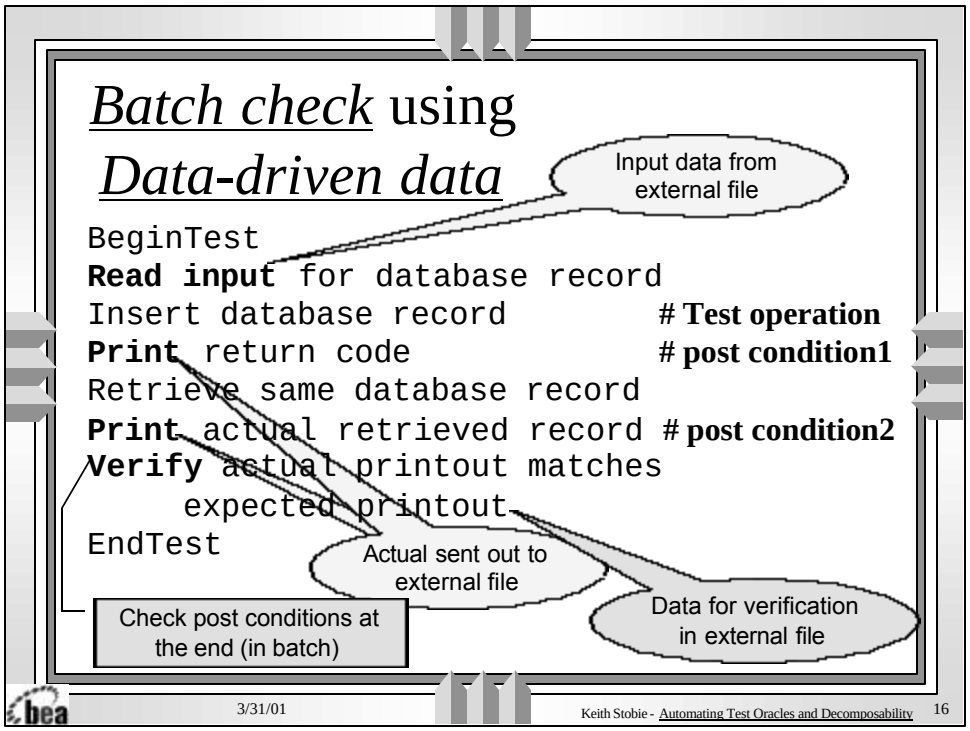
```
While read in_num, out_result; do
if (out_result = “illegal”)
   then if (squareRoot(in_num) != “illegal”)
      print “fail $in_num”;
   else if (squareRoot(in_num) != $out_result)
      print “fail $in_num”;
done
```


Comparison Timing (check patterns)

```
BeginTest
   Insert database record        # Test operation
   Verify successful return code # post condition1
EndTest

BeginTest
   Retrieve same database record
   Verify actual retrieved record matches
      (expected) input record. # post condition2
EndTest
```

Anti-Pattern: Pass Each Post-Condition


Check as you Go
using Self-contained data
BeginTest
Insert database record # Test operation
Verify successful return code # post condition1
Retrieve same database record
post condition2
Verify actual retrieved record matches
(expected output) input record.
EndTest
Check each post condition immediately (as you go)
Data for verification in the test code
3/31/01
Keith Stobie - Automating Test Oracles and Decomposability
15


Batch check using
Data-driven data
Input data from external file
BeginTest
Read input for database record
Insert database record # Test operation
Print return code # post condition1
Retrieve same database record
Print actual retrieved record # post condition2
Verify actual printout matches
expected printout
EndTest
Actual sent out to external file
Data for verification in external file
Check post conditions at the end (in batch)
3/31/01
Keith Stobie - Automating Test Oracles and Decomposability
16

Pattern Conditions

Question	Answer	***Pattern*** / Anti-Pattern
Where to store test results?	All internal	***Self-Contained Data***
	All external	***Data-Driven Data***
When to check?	At each step	***Check as you Go***
	At the end	***Batch Check***
Depend on previous test?	No	***Separable Tests***
	Yes	***Smart Dependency Checking***
Pass a test for each post-condition?	No	***Whole Function***
	Yes	Pass Each Post-Condition

Check as you Go

Aliases: *In-Line check*

Context: You have a Test Oracle for verifying the test results.

Problem: Do you compare actual results to known expected results at each step as you go, or all at the end?

Forces:

- **Future results** may be invalid if early results don't pass.
- Data from the environment is **dynamically** needed to evaluate correctness.
- Correctness requires specific relationships to occur, for example **complex data** structures like trees.
- Either the checks are very **cheap** to make or the test is **not highly performance sensitive**.

Check as you Go

Indications

- The desired results are precisely known ahead of time.
- The result of each set of inputs is easily checked.
- Test is meaningless if early discrepancy for a post-condition.

Solution

Check each result in-line as finely as possible immediately after its inputs are submitted. If a validation fails, log the failure and then don't proceed forward with the rest of the test.

Rationale

It generally aids comprehensibility if the expected results appear in the same file and as close to the inputs as possible.

Code Samples: See ***Check as you Go*** using *Self-contained data*.

Batch check

Aliases: *Benchmark, baseline, golden results, canonical results, gold master*
- a benchmark file containing expected results is used.

Context: You have a Test Oracle for verifying the test results **or** the specification may be via the pattern *Judging* actual results when expected result is not necessarily known.

Problem: Do you compare results at each step as you go or all at the end?

Forces

- The set of **inputs** is **not** easily **separable**
- The output can be **compared** easily **with** minimal **filtering**.
- The **checks are expensive** to make **or** the test is highly **performance sensitive** and it is relatively cheap to just record the results.

Batch check

Indications: A failure near the start of the test doesn't invalidate the results that follow.

Solution: Provide a benchmark file of expected results. Collect actual results as the test executes. At the end compare the expected and actual results.

Rationale: Tests are very easy to develop. Expected results can be generated by the program once, and hand-checked for accuracy once, and then reused again and again. This changes the *Judging* pattern into *Solved Example.* Expected results can be updated without affecting any code (since they are in a separate file). Batch processing may be the nature of item under test.

Batch check

Resulting Context/Consequences

Frequently **testers** get lazy when the expected output has to change and **don't scrutinize** the initial results carefully enough for correctness. In this case, the actual *incorrect output gets canonized* as the expected output.

Batch check can make maintenance more difficult *if* the relationship between inputs and outputs is not very clear.

Frequently special filtering patterns (regular expressions) are needed to ignore uncontrollable extraneous differences, for example machine names, time stamps, etc.

Examples/Known Uses

Compiler testing or any transformation type program. It is generally too expensive to test each feature completely individually, and a great deal of common setup exists to test any one feature.

Code Samples: See ***Batch Check*** using *Data Driven Data.*

Smart Dependency Checking

A test of some data storage mechanism might have three test methods, `testBind()`, `testLookup()` and `testUnbind()`. You want to run the bind test first; the lookup test second; and the unbind test last and only if the bind test passed.

If you only require a test method to have had a chance to run, but not necessarily to have passed, you can prefix the test name with a '%'.

```
public boolean testBind() { ... }
public boolean testLookup() {
        require("Bind"); ... }
public boolean testUnbind() {
         require("Bind,%Lookup"); ... }
```


Smart Dependency Checking

Extending the ***Move actual to Internal*** example

BeginTest MaxLocks

```
for I=1 to numlocks {getDBlock();}
```

EndTest

BeginTest ExceedLocks **requires(MaxLocks)**

```
  try { getDBlock(); logFail(); }
    catch (TooManyLocksException e)
          {/*Got expected exception*/}
    catch (Exception e) { logFail(); };
  System.exec("diff DBlog
  expectedDBlog");
```

EndTest

Automating Test Oracles and Decomposability

Conceptually all tests have three parts:

- the input (including the environment and internal state) is the stimulus provided by a test designer
- the expected output (including environment and internal state) is what the test requires to consider the software correct, it is generated via some Test Oracle
- the actual output (including environment and internal state) is the result of executing the software with the given input.

The output is a result of a set of post-conditions associated with the software being tested.

Where and how do you check results? What information do you record about results?
How does result checking impact test design?

Test Data Location questions: Where do the input, expected output, and actual output reside? In the test case code? In a separate set of files or a database?

Comparison Timing questions: How close in time after the execution of a test is the check of the result made? As each step of the test is made? After the entire test? After a set of tests?

If there were only one answer to these questions, they would have been solved long ago. Instead, a series of trade-offs (forces), helps determine when each method is most appropriate. This paper uses the concept of patterns[1] to describe the details of why you choose a particular test result handling method to automate a test design.

Context

You are designing tests and need to consider how much to reasonably include in a given test.
You are choosing how to automate tests that have been designed and need to make the tests as understandable and maintainable as possible.
This doesn't apply to ad hoc testing, exploratory testing or testing without an Oracle.

Test Data Location (data patterns):

The input, expected output, and actual output may be coded directly into the test (program code, test script, etc.) or apart from it (input file, expected result database, etc.). The *Co-locate Data* pattern tells us that we should make them all internal or all external resulting in the *Self-contained data* or *Data-driven data* patterns respectively. The helper patterns *Move actual to Internal* and *Move actual to External* allow transformations into the all internal or all external patterns. The anti-patterns, **Cause without Effect** and **Effect without Cause**, show what results from separating the input from the output.
The table below lists patterns in italics and anti-patterns using outlined text.

Input - Cause	Output – Effect		***Pattern*** / **Anti-Pattern**
	Expected	Actual	
Internal	Internal	Internal	***Self-contained data***
Internal	Internal	**External**	*Move actual to Internal*
Internal	**External**	Internal	**Cause without Effect**
Internal	**External**	External	**Cause without Effect**
External	**Internal**	Internal	**Effect without Cause**
External	**Internal**	External	**Effect without Cause**
External	External	**Internal**	*Move actual to External*
External	External	External	***Data-driven data***

[1] If necessary see "*A Pattern Language for Pattern Writing*" by Gerard Meszaros and Jim Doble at http://www.hillside.net/patterns/Writing/patterns.html for the more details about the meaning of the headings in the patterns used in this paper.

Comparison Timing (check patterns)

Many of the features of software that are tested result in multiple post-conditions. Each post-condition must be checked, but is each post-condition its own test? See **Pass Each Post-Condition** anti-pattern for why using only a single post-condition as a test is misleading. The *Whole Function* pattern considers a single test to include the **evaluation** of all of the relevant post-conditions.

Thus the following is considered incorrect:

```
BeginTest
Insert database record          # Test operation
Verify successful return code   # post-condition1
EndTest
BeginTest
Retrieve same database record
# post-condition2
Verify actual retrieved record matches (expected) input record.
EndTest
```

Instead the following should occur using either the *Check as you Go* pattern or the *Batch check* pattern described later:

```
BeginTest    [Check as you Go using Self-contained data]
Insert database record              # Test operation
Verify successful return code       # post-condition1
Retrieve same database record
# post-condition2
Verify actual retrieved record matches (expected output) input
record.
EndTest
```

or

```
BeginTest    [Batch check using Data-driven data]
Read input for database record
Insert database record          # Test operation
Print return code               # post-condition1
Retrieve same database record
Print actual retrieved record   # post-condition2
Verify actual printout matches expected printout
EndTest
```

Notice there are two verify steps in the *Check as you Go* case above. Either verify step can cause the test to mark the feature as failed.

However the timing and number of verifies is not prescribed. It is not even required that a verify be a direct part of the test code. Sometimes several tests will output their results before any of the comparison (verification) is done. However, each test is not considered complete *until* the results of all successful comparisons are done. It is acceptable to not complete the comparisons if a discrepancy has already been shown. The primary consideration for continuing comparisons after a discrepancy is whether it would provide additional useful information for diagnosing the failure. It does not impact the outcome of the test.

Question	Answer	***Pattern*** / **Anti-Pattern**
Where to store test results?	All internal	***Self-Contained Data***
	All external	***Data-Driven Data***
When to check?	At each step	***Check as you Go***
	At the end	***Batch Check***
Depend on previous test?	No	***Separable Tests***
	Yes	***Smart Dependency Checking***
Pass a test for each post-condition?	No	***Whole Function***
	Yes	**Pass Each Post-Condition**

The following patterns are a linkage between the Test "Oracle Micro-Patterns" for "Pre-Specification Oracles" (*Solved Example, Simulation, Approximation, and Parametric*) and "Test Automation Design Patterns" approaches of *Built-in Test* and *Test Cases* as described in *Testing Object-Oriented Systems: Models, Patterns, and Tools* (*http://www.rbsc.com/TOOSMPT.htm*). [See appendix for summaries of these patterns.]

The second pair of patterns are complementary patterns for solving the same problem. *Check as you Go* is the generally preferred method for ease of understanding. *Batch check* (or *Benchmark*) is particularly useful for changing the Oracle *Judging* pattern into *Solved Example*.

The third pair of patterns are supplemental Test Automation Designs. *Separable Tests (*or *Atomic Tests)* deals with selection and independence of tests. *Smart Dependency Checking* is used when decomposability to satisfy the *Separable Tests* pattern precludes the second test from being independent.

*Whole Function (*or *All behavior)* deals with post-conditions of an Item Under Test (IUT).

Note that patterns can be combined as the situation demands. You might use *Check as you Go* for some of the post-conditions which increases the ability to create *Separable tests* and yet use *Batch check* for voluminous post-conditions which may change frequently.

The rest of this paper presents the patterns followed by an appendix with pointers to other patterns and references.

Pattern Name: *Co-Locate Data*

Context

Reusable tests are being created and the data for the tests must be stored.

Problem

Inputs come from various sources including the test script, the environment, or internal state. Similarly output consists of various sources including the environment, internal state, test script, or output files (including standard out, standard error, log files, and database files).

Forces

Input and Output are naturally separated streams and are usually not mixed.

Solution

Put the input, expected output, and actual output either within the test code or put them all centralized external to the test code. It should be possible to see the input and expected output together (either in the code, in a file, or in an extract from a database). Transform input or output from other sources either into the code or the centralized external location.

Indications

Input data and expected output data are separated.

Rationale

Having the input and expected output in the same place (either internal or external) increases readability and understandability (including for maintenance). If you separate them, then you end up with the anti-patterns: **Cause without Effect** and **Effect without Cause**.

Generally, tests are designed to transform the cases where the expected and actual output reside in different locations, into the cases where they are all the same.

Resulting Context/Consequences

Verification of the correct expected output is easier since it is all in one place.

Related Patterns

See ***Self-contained data*** for putting the data inside the test script or program.
See ***Data-driven data*** for keeping the data outside the test script or program.
See *Move actual to Internal* or *Move actual to External* to make the actual data co-located with the input and expected output.
The anti-patterns **Cause without Effect** and **Effect without Cause** describe problems with bad solution of having the input not co-located with the expected output.

Pattern Name: *Self-contained data*

Aliases: internal data

Context

You are creating a reusable test.

Problem

Where do you keep the test input data and test expected output data?

Forces

- Want to read and understand a test with as few external references as possible.
- Tests are each relatively unique in their parameters (or sequences of actions).

Solution

Include the data in the test script or test code.

Indications

The amount of input and output per result is relatively small and easy to understand.

Rationale

Test script logic becomes less complicated or more obvious when it is included directly with the logic.

Resulting Context/Consequences

Makes it impossible to lose part of the test, if it is only in one file.
Sometimes reduces maintainability if bulk updates of expected results are needed.
Reduces code reusability if inputs and results are hard-coded or expressed as symbolic constants in the code.

Related Patterns

Contrast with *Data-driven data*.
See also *Move actual to Internal*.

Examples/Known Uses

Typically used in API tests, for example Posix Verification Suite.

Code Samples

The *Check as you Go* pattern Code Sample demonstrates simple *Self-contained data*.
The *Batch check* pattern Code Sample demonstrates input and output contained within the test script.

Pattern Name: *Move actual to Internal*

Context

The result of running a program is some external post-condition, for example writing a message in a separate log-file, but the input and other post-conditions (expected output) are internal.

Problem

The check of post-conditions is distributed between external script code and internal script code.

Forces

- Post-conditions occur in different environments.

Solution

The external actual output is read into or checked by the test program to transform it into the internal actual output case.

Indications: Input and most output are internal.

Rationale: It is most convenient to compare between expected output and actual output in the same location. Since the input and exptect output are already internal, they are the majority, and the actual output must be made to match them.

Resulting Context/Consequences: You have ***Self-contained data*** after moving actual to internal.

Related Patterns: See ***Self-contained data***.

Code Samples

A call to lock a record in a database that is out of locks might have two post-conditions:

1. an exception is returned to the caller indicating no locks (internal), and
2. a message is written to the operations log indicating the database has run out of locks (external).

The first example shows an external post-condition (DBlog output) causing comparison being done external to the test program in addition to the internal post-condition (exception raised) because of the internal input.

Run DB lock test (internal)

```
for I=1,numlocks {getDBlock();}
try { getDBlock(); logFail(); }
  catch (TooManyLocksException e){/*Got expected exception*/}
  catch (Exception e) { logFail(); }

diff DBlog expectedDBlog (external)
```

Applying this pattern results in all post-conditions being checked for internally in the code

Run DB lock test (internal)

```
for I=1,numlocks {getDBlock();}
try { getDBlock(); logFail(); }
  catch (TooManyLocksException e){/*Got expected exception*/}
  catch (Exception e) { logFail(); };
System.exec("diff DBlog expectedDBlog");
```

The above shows all post-conditions being checked from within the test code even though external actual output is involved.

Pattern Name: *Data-driven data*[2]

Aliases: external data

Context

You are creating a reusable test and you have many data combinations to be tested.

Problem

Where do you keep the test input data and test expected output data?
Hard-coding data in test scripts makes it laborious to create lots of related tests.

Forces

- Output is voluminous.
- Output is difficult to predict and can frequently change from release to release.
- Additional, similar tests may need to be added.

Solution

Separate test data from scripts. This makes it easier to create multiple related tests.

Indications

Test data is to be provided externally, for example from domain experts.
You have existing legacy data.

Rationale

Separating data from procedure is a classic computer science technique for structuring code.

Resulting Context/Consequences

Mentally tracing through the test requires an extra level of indirection to substitute the data-driven values in the specific situation. Test script logic becomes more complicated or less obvious when data is separate.

Related Patterns

Frequently used with *Batch Check*. Contrast with *Self-contained data*.
See also *Move actual to Internal*.

Examples/Known Uses

Compiler tests.
SQL optimizer tests showing optimizer strategy (which may change release to release).
Stub generation for distributed methods (for example CORBA IDL, or Java RMI).

Code Samples

If the *Batch check* pattern Code Sample had an external existing `actualOutput` file, instead of creating it on the fly, it would demonstrate *Data-driven data*.
The *Move actual to External* code sample also shows *Data-driven data*.
The **Cause without Effect** code sample also shows *Data-driven data*.

[2] Much of this material is a derivation from *Data-driven testing* pattern presented at PoST 1, Jan. 2001

Pattern Name: *Move actual to External*

Context

The result of running a program is some internal post-condition, for example an exception being raised, but the input and other post-conditions (expected output) are external.

Problem

The check of post-conditions is distributed between external script code and internal script code.

Forces

- Post-conditions occur in different environments,

Solution

The internal post-condition effect is outputted to transform it into the actual external output case.

Indications

Input and most output are external.

Rationale

It is most convenient to compare between expected output and actual output in the same location. Since the input and expected output are already external, they are the majority and the actual output must be made to match them.

Resulting Context/Consequences

You have *Data-driven data* after moving actual to external.

Related Patterns

See *Data-driven data***.**

Code Samples

For bad syntax a compiler is supposed to have two post-conditions:

1. display an error (external output), and
2. exit with a non-zero status code (internal to script check).

Below is a mixture of checking a post-condition in the script (internal) and externally.

```
Compile <testInput >actualOutput
if [ $? != 0 ] ; then logFail(); # Internal
diff actualOutput expectedOutput # External
```

After applying the pattern you get:

```
Compile <testInput >actualOutput
echo "ResultCode $?" >>actualOutput
diff actualOutput expectedOutput  # External
```

The above shows all post-conditions being checked externally to the test code even though internal actual output is involved.

Anti-Pattern Name: Effect without Cause

Aliases: output only

Context

Expected output is recorded without knowing the input.

Problem

Outputs can match, but for the wrong reasons.

Forces

Solution

Record input with the outputs. It is also possible to derive the input as an extraction from the expected output.

Rationale

It is easier to review for correctness when the inputs and outputs are together.
It is easily verified if each effect occurs due to its cause.

Resulting Context/Consequences

Test may have to echo or otherwise copy the input into the output stream.

Code Sample

Input file:

```
set onn   # should fail because it should be “set on”
set Off   # should succeed because Off should be case insensitive
```

Actual & expected outputs:

```
Illegal Set argument
```

Test marks product as passed because it got expected output. Although this has the input, expected output, and actual output all external files, this input is external to the expected output file.
Correct way:

Expected output:

```
set onn   # should fail because it should be “set on”
Illegal Set argument
set Off   # should succeed because Off should be case insensitive
```

Actual output:

```
set onn   # should fail because it should be “set on”
set Off   # should succeed because Off should be case insensitive
Illegal Set argument
```

Test marks product as failed.
The product only looks at first 2 letters (“on”) and is not case insensitive.

Notice how the expected output is easy to understand since both the cause and effect show up in the file.

Anti-Pattern Name: Cause without Effect

Aliases: input only

Context

Input is recorded externally to the expected output.

Problem

Matching the expected output with the input is error-prone during maintenance.

Forces

Solution

Record expected output with the input.

Rationale

It is easier to review for correctness when the inputs and outputs are together.
It is easily verified if each effect occurs due to its cause.
Additional inputs can be easily added since their expected output is recorded with them.

Code Samples

Input file: 1 4 9 –1 0

Test code:

```
For I=1 to 3; do get num;
      if (square(squareRoot(num)) != num) print “fail $num”;
      done
For I= 1 to 2; do get num;
      if (squareRoot(num) != “illegal”) print “fail $num”;
      done
```

Note that the test code (internal effects) is tightly tied to the input (external cause) and changing either creates test (not product) failures. This is a very brittle coding style.

Better, using *Data-driven data* pattern is:
Input file:

```
       1  1
       4  2
       9  3
      -1  illegal
       0  illegal
```

Test code:

```
While read in_num, out_result; do
      if (out_result = “illegal”)
      then if (squareRoot(in_num) != “illegal”) print “fail $in_num”;
      else if (squareRoot(in_num) != $out_result) print “fail $in_num”;
      done
```

This prevents the brittle code and is easily expandable. You can add additional test cases by changing the input file without changing the test code. This is an example of Data-Driven Data.

Pattern Name: *Check as you Go*

Aliases: In-Line check

Context

You have pre-specification of the test results.

Problem

Do you compare actual results to known expected results at each step as you go, or all at the end?

Forces

- Future results may be invalid if early results don't pass (see also *Separable Test*s).
- Data from the environment is dynamically needed to evaluate correctness.[3]
- Correctness requires specific relationships to occur, for example complex data structures like trees.
- Either the checks are very cheap to make or the test is not highly performance sensitive, that is, you can afford to spend time to do the checks during the test.

Solution

Check each result in-line as finely as possible immediately after its inputs are submitted. If a validation fails then log the failure and optionally don't proceed forward with the rest of the test. For example, when bounding the time of the result, if a connection isn't made within a timeout period, abort the test rather than waiting to get more output.

Indications

- The desired results of a test input are precisely known ahead of time.
- The test is programmable, that is, the result of each set of inputs is easily checked.
- The test is long running, and could be meaningless if there is an early discrepancy for one of the post-conditions.

Rationale

It generally aids comprehensibility of the tests if the expected results appear in the same file and as close to the inputs as possible.

Resulting Context/Consequences

Complete list of post-conditions being checked may be spread throughout the test code.

Related Patterns

See *Batch check* for the same problem, but different forces.

Examples/Known Uses

Frequently used for API/Class Drivers approach.

Junit – See http://www.junit.org/
Expect tool – See http://expect.nist.gov/
POSIX Verification Test Suite – See http://www.opengroup.org/testing/downloads/vsx-pcts-faq.html

Code Samples

Note below that the result is checked as you go in the code and not by some external entity.

[3] For example, you want to verify the timestamp on a log record. You can print the time before and after you expect the log record, but now your batch comparison requires relative checks (less than and greater than) instead of just equals. This is usually a significantly more difficult comparison algorithm.

Java/C++

```
result = squareRoot(1);
if (result != 1) {
      LogError( “squareRoot(1) resulted in “+result
              +” where 1 was expected” )
      }
result = squareRoot(4);
if (result != 2) {
      LogError( “squareRoot(4) resulted in “+result
              +” where 2 was expected” )
      }
```

Shell

```
result=`squareRoot 1`
if [ “$result” != “1” ] ; then
      echo “squareRoot 1 resulted in $result, where 1 was expected.”
fi
result=`squareRoot 4`
if [ “$result” != “2” ] ; then
      echo “squareRoot 4 resulted in $result, where 2 was expected.”
fi
```

Pattern Name: *Batch check*

Aliases: *Benchmark, baseline, golden results, canonical results, gold master*
- where a benchmark file containing expected results is used.

Context

You have pre-specification of the test results or the specification may be via the pattern *Judging* actual results when expected results are not necessarily known.

Problem

Do you compare results at each step as you go or all at the end?

Forces

- The set of inputs is not easily separable (for example compiler input file).
- The output can be compared easily with minimal filtering.
- The checks are expensive to make or the test is highly performance sensitive and it is relatively cheap to just record the results.

Solution

Provide a benchmark file of expected results. Collect actual results as the test executes. At the end compare the expected and actual results.

Indications

- A failure near the start of the test doesn't invalidate the results that follow.

Rationale

Tests are very easy to develop. Expected results can be generated by the program once, hand-checked for accuracy once, and then reused again and again. This changes the *Judging* pattern into *Solved Example*. Expected results can be updated without affecting any code (since they are in a separate file). Batch processing may be the nature of Item Under Test (IUT).

Resulting Context/Consequences

One dangerous Consequence frequently seen is testers get lazy when the expected output has to change and don't scrutinize the initial results carefully enough for correctness. In this case, the incorrect actual output gets canonized as the expected output. See Test Automation Snake Oil at http://www.satisfice.com/articles/test_automation_snake_oil.pdf

Batch check can make maintenance more difficult *if* the relationship between inputs and outputs is not very clear.

Frequently special filtering patterns (regular expressions) are needed to ignore uncontrollable extraneous differences, for example machine names, time stamps, etc. This filtering has a small risk of missing incidental problems, such as the time being reported incorrectly. Generally you rely on other tests to specifically verify what most of these types of tests ignore.

Related Patterns

See *Check as you Go* as an alternate method.

Examples/Known Uses

Compiler testing or any transformation type program. It is generally too expensive to test each feature completely individually, and a great deal of common setup exists to test any one feature.

Code Samples

The abbreviated example below shows the expected output stored in a separate file and then a batch comparison done.

Shell:

```
cat <<EOINPUT >|expectedOutput
input output
1 1
4 2
EOINPUT

# Set up for read from fd=4 with above data
exec 4<expectedOutput

read -u4 input_value?"headings " output_value
echo $input_value $output_value >| actualOutput
while read -u4 input_value?"input and output" output_value; do
   echo "$input_value \c" >> actualOutput
   squareRoot $input_value >> actualOutput
done

diff expectedOutput actualOutput
```

Pattern Name: *Separable Tests*

Aliases: ***Atomic Tests, Independent Tests***

Context

You know the nature of the tests you want to run and easily identify exactly which existing tests or subset of tests you need.

Problem

How do you run as few tests as possible and as small a set of tests as possible if you know what you want to test?
How can you run tests in any order if they each require a different setup?

Forces

- Tests may have setup dependencies.
- Tests should be as small and specific as possible.
- Tests need to be efficient and not repeat operations excessively.

Solution

Provide as fine grain a selection mechanism as possible. For API tests this means being able to select tests within a program via GUI, command line, or environmental options. Provide many ways to categorize tests (see *Test Keywords Management* in the Appendix).

Indications

- Developers or Managers request: can we run a test that just does X?
- Tests require different resources (for example network connections or database access)

Rationale

Quicker reruns of tests are usually possible if only a single small test must be run.
When development is doing defect reproduction or review, it is easier to understand small tests rather than large, multi-condition tests.

Resulting Context/Consequences

- Tests can be independently run based on each test's unique characteristics.
- Separating into very point-specific tests reduces the chances of serendipitous findings and may also avoid testing of any interactions – both of which can reduce bug-finding abilities of tests.[4]
- Running several small tests may take more time than running one large test. At a minimum there may be extra data recording time (start, stop, success or failure) for each small test.
- If a large test is broken into several smaller tests:
  - it is easier to run them in different orders, which can increase the chance of finding defects.
  - it is possible to run tests which might have been blocked by an early failure in the big test
  - there is a cleaner isolation of pass/fail for each feature

Related Patterns

Smart Dependency Checking describes methods of stating and satisfying test dependencies.

Test Keywords Management (in the Appendix) describes methods of selecting separable tests.

Examples/Known Uses

TET (http://tetworks.opengroup.org/) provides `tet_testlist` to allow "invocable components" within a program to be separately run.
Rational Test Manager – See http://www.rational.com/products/testmanager/index.jsp

[4] *Craft of Software Testing* by Brian Marick

Pattern Name: *Smart Dependency Checking*

Context

A test requires same environment setup as provided by another test. Conversely, a large test can be broken up into parts such that the early parts could be run without running the later parts and still be useful tests.

Problem

How do we allow running of any arbitrary test if the test requires other tests to run before it?

Forces

- Tests need to be efficient and not repeat operations excessively.

Solution

Provide methods of stating and satisfying test dependencies. Tests must indicate any dependencies required, and the execution harness must order tests to meet the dependencies.

Indications: Tests share a lot of common code that each must execute.

Rationale

By having each test state its dependencies, an intelligent execution harness can order the tests to satisfy the dependencies and run as few tests as possible.

Resulting Context/Consequences

Dependencies between tests are automatically accommodated.

Examples/Known Uses

TestFrameWork provides the `requires()`[5] method to automatically build the tree of required tests. For example a test of some data storage mechanism might have three test methods, `testBind()`, `testLookup()` and `testUnbind()`. Obviously you want to run the bind test first; the lookup test second; and the unbind test last and only if the bind test passed. To do this you'd write the methods this way. If you only require a test method to have had a chance to run, but not necessarily to have passed, you can prefix the test name with a '%'.

```
public boolean testBind() { ... }
public boolean testLookup() { require("Bind"); ... }
public boolean testUnbind() { require("Bind,%Lookup"); ... }
```

Code Samples

A test for exceeding the database locks could be as shown in *Move actual to Internal*.
However, the database test would typically be decomposed into two tests: the first test to verify the expected maximum number of locks can be reached and a second test to verify the error condition. This would result in the overflow test being dependent upon maximum number of locks. But, the MaxLocks test can be run independently of the ExceedLocks test if that is all that is needed for testing.

```
BeginTest MaxLocks
for I=1 to numlocks {getDBlock();}
EndTest
BeginTest ExceedLocks requires(MaxLocks)
try { getDBlock(); logFail(); }
  catch (TooManyLocksException e){/*Got expected exception*/}
  catch (Exception e) { logFail(); };
System.exec("diff DBlog expectedDBlog");
EndTest
```

[5] "*Creating a Testing Culture*" by Keith Stobie, Quality Week 1999

Pattern Name: W*hole function*

Aliases: ***All behavior***

Context

An Item Under Test (IUT) has multiple post-conditions associated with it. Is each post-condition a separate test?

Problem

What does it mean for a test to pass, or how fine-grained should comparisons be?

Forces

- Comparisons must be fine-grained.
- Tests must not report false positives (a successful execution when the product doesn't actually work).

Solution

Verify all of the post-conditions associated with a function being tested before considering a test as having shown the function passing.

Indications

Function has multiple post-conditions.

Rationale

If only one post-condition is checked, then a contradiction between post-conditions may not be detected, thus missing a defect!

Resulting Context/Consequences

A test can show a failure for multiple reasons (at least one per post-condition).

Related Patterns

Typically used in conjunction with *Separable Tests*.
See the anti-pattern **Pass Each Post-Condition**.

Examples/Known Uses

TET distinguishes between a "Test Purpose" and an "Invocable Component" which may have several test purposes, but can't be run separately.

Anti-Pattern Name: Pass Each Post-Condition

Context

An Item Under Test (IUT) has multiple post-conditions associated with it. Each post-condition is considered a separate test.

Problem

How to indicate that each post-condition has been checked?

Forces

- Management likes to see lots of tests.
- A post-condition can be thought of as a test.

Solution

Print pass or fail after each post-condition check or after each comparison.

Indications

Function has multiple post-conditions.

Rationale

The reasoning for not considering each post-condition as a test is as follows:
If the second test shows the second post-condition as failing, then it might be incorrect to say that the first post-condition in the first test succeeded. Yet, as written, the tests will indicate you should get a successful first post-condition even if the second post-condition fails.

For example:

```
BeginTest
Insert database record          # Test operation
Verify successful return code   # post-condition1
EndTest
BeginTest
Retrieve same database record
# post-condition2 or test of retrieve?
Verify actual retrieved record matches (expected) input record.
EndTest
```

The reasoning for not considering these as two tests is as follows:
If the second test shows the database record is missing, then it might be incorrect to say that the insert in the first test succeeded and that the return code should have been a successful one. Yet, as written, the tests will indicate you should get a successful return code from a failing insert and say the Retrieve (Find) function has failed!

Resulting Context/Consequences

Functions are considered partially passing when they present inconsistent post-conditions, that is, there exists a "test" of the function that succeeds.

Related Patterns

See *Whole function* for correct usage pattern.

Code Samples

A typical example looks like:

```
insert (expectedRecord)
if insert doesn't fail, then print PASS
else print FAIL
```

The other post-condition, that the insert had the desired effect is left to another test!
For example:

```
read (actualRecord)
if (expectedRecord != actualRecord)  the print PASS
else print FAIL
```

and the failure might be ascribed to the `read` instead of the `insert`. In fact, without additional tests, it is impossible to distinguish whether it is the `read` or the `insert` that failed. A good failure message in this case would be something like,

> "Read of actual record:<actual record> didn't match expected record: <expected record> that was inserted."

Appendix

To be written:
Test Keywords Management describes methods of selecting separable tests.

Logging strategies. – only logging on failure. Include and identify expected and actual results.
Known failure – providing a three way result Pass, Fail-known, Fail-unknown instead of typical Pass/Fail

From *Testing Object-Oriented Systems: Models, Patterns, and Tools* (*http://www.rbsc.com/TOOSMPT.htm*):

Oracle Patterns (micro-pattern schema)

Approach	Pattern Name	Intent
Judging	Judging	The tester evaluates pass/no-pass by looking at the output on a screen or at a listing, or by using a debugger or another suitable human interface.
Pre-Specification	Solved Example	Develop expected results by hand or obtain from a reference work.
	Simulation	Generate exact expected results with a simpler implementation of the IUT (e.g., a spreadsheet.)
	Approximation	Develop approximate expected results by hand or with a simpler implementation of the IUT.
	Parametric	Characterize expected results for a large number of items by parameters.

Test Automation Design Patterns

Capability	Pattern Name	Intent
Built-in Test	Percolation	Perform automatic verification of super/subclass contracts.
Test Cases	Test Case/ TestSuite Method	Implement a test case or a test suite as a method.
	Catch All Exceptions	Test driver generates and catches IUT's exceptions.
	Test Case / Test Suite Class	Implement test case or test suite as an object of class TestCase.

QW2001 Paper 7T1

Mr. Don Cohen
(Princeton Softech)

Requirements For A Comprehensive Testing Environment

Key Points

- Heightened Testing Need
- Automated Testing
- Creating and Maintaining Test Data

Presentation Abstract

This presentation outlines the driving forces in the business community today that are behind the increased importance of a comprehensive testing scheme, including application quality and “time to market” for customer-facing applications, and the processes and tools that need to be in place in order to accomplish the testing job.

The presentation will look back over changes to the business climate over the last five years that have precipitated a heightened awareness of application quality and ‘time to market” and why testing is becoming a “critical business need” for many IT groups today. The presentation will also look at the advent of “test factories” as a new approach to testing by large corporations, both domestically and internationally.

Once the heightened need for testing is established, the presentation will survey the processes and tools that must be present in every IT organization if they are to have a “fighting chance” to meet the new Service Level Agreements. It will further explore the “must have”, “nice to have” and “other” facilities typically embodied in these tools and the reason why such capabilities are important.

Examples of the sub-topics covered include; the key requirements when using automated testing (scripting) tools, creating meaningful test data in a relational database environment (for both new and enhanced applications), dealing with differences between the production environment and the test environment, and the value of intelligently comparing test results in an automated manner.

About the Author

Don Cohen is Vice President of Research and Development at Princeton Softech, a New Jersey-based provider of DB2 productivity software. He has been involved in the development of sophisticated system software products in the areas of

communications, operating systems, languages and relational databases for over 20 years as a developer, development manager, product manager and VP at Bell Laboratories, Applied Data Research (ADR), Computer Associates (CA), Automated Data Processing (ADP) and Princeton Softech. Mr. Cohen has been a speaker and trainer at conferences and for clients in Europe, Australia, South America and North America.

Requirements for a Comprehensive Testing Scheme in a RDBMS Environment

Don Cohen
VP Development
dcohen@princetonsoftech.com

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Today's Agenda

- **The Price of "Short Changing" Testing**
- **The Wish List….**

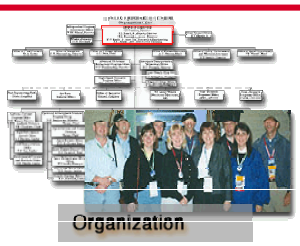
  - What you should want and why

PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

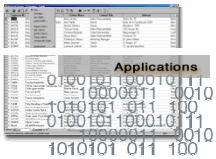
At the Core of the Business

Yesterday:

IT Systems Supported the Business….



..Today: IT is the Business!



PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Slide 3

The Importance of the Testing Process

“Application and data quality have always been important, but with the advent of ‘customer facing’ applications that expose applications to the outside world, it has become a critical issue for IT and the business as a whole. ”

Gartner
insight for the connected world

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Slide 4

Testing: The Value Proposition

- **Application Reliability**
- **Time-to-Market**
- **Cost of Quality**
- **Competitive Advantage**

Yesterday quality wasn't a business weapon -- *Today* it is

***Quality* is becoming a competitive differentiator**

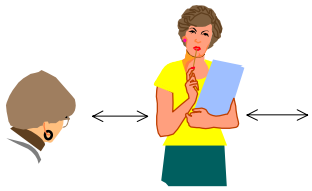
PRINCETON SOFTECH
A Computer Horizons Company

Slide 5

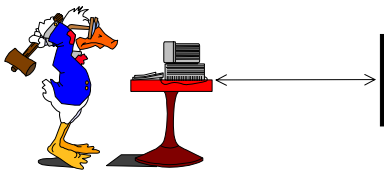
We get it right. Every time.™

Reliability

- Yesterday, since we were dealing with internal users, we could compensate for poor reliability via Internal Users and Intermediaries.



- Today we can't due to the dual impact of Direct Customer Contact and the fact that Customers have Choices.



PRINCETON SOFTECH
A Computer Horizons Company

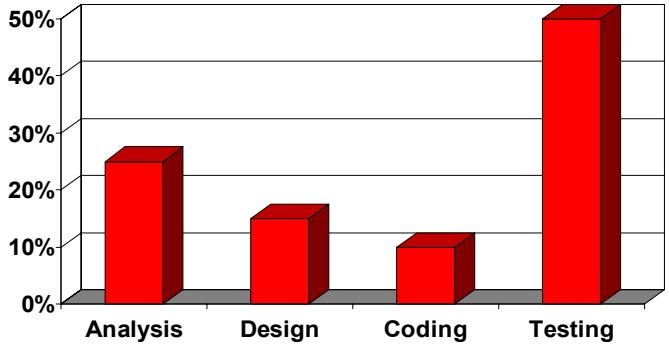
Slide 6

We get it right. Every time.™

Time-to-Market

Yesterday longer life cycles may have been tolerable,

Today there's less tolerance!



% of Time and Effort by Phase

PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 7

The "Typical" Testing Phases

Unit Testing **- Low level testing at a specific function (module) or unit (subsystem) level. Typically for new features, usually done by developer.**

Integration Testing **– Mid level testing at integration points between units or functions. Typically for new features, usually done by developer.**

System Testing **– Higher level, end-to-end testing of new functionality. Usually done by QA organization.**

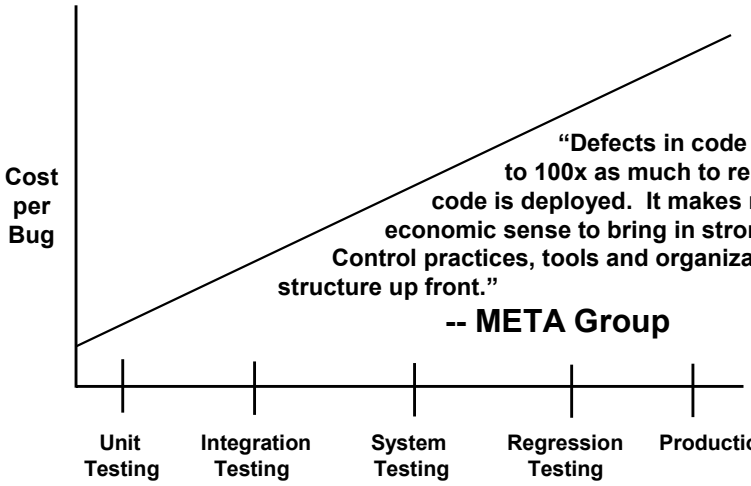
Load Testing **(or Performance, or Stress) – Focused on response time and throughput, not functionality. Usually done by QA organization.**

Regression Testing **- Higher level, end-to-end testing of previously existing functionality. Usually done by QA organization.**

PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 8

Cost of Quality



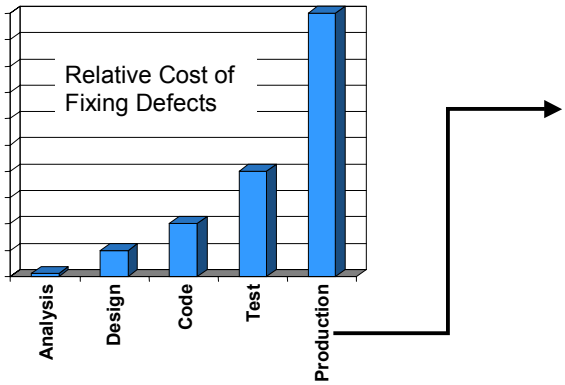
PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 9

Cost of Quality

Yesterday quality didn't have the CEO's attention -- *Today* it does!

Cost of Quality ≠ **Cost of Fixing Defects Alone**



Business Cost of Defects

- Lost Sales
- Lost Customers
- Lost Suppliers
- Contractual Penalties
- . . .

PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 10

Why is Testing a Challenge?

- Many organizations treat Testing as if it is of minor importance
- A comprehensive testing scheme can be "as challenging" of designing the application
- Requires diligence and perseverance and …

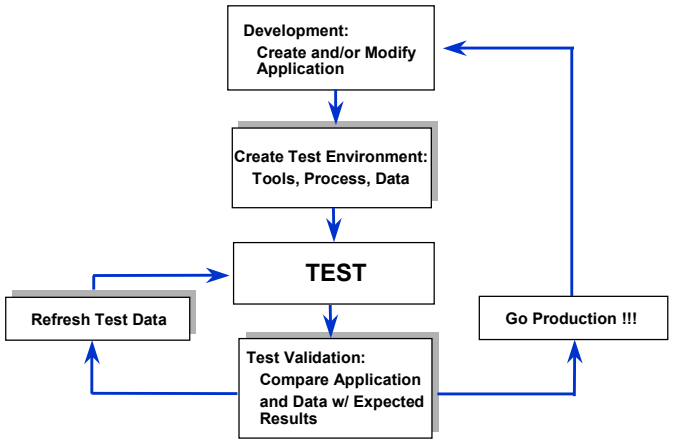
Often, by the time organizations recognize the situation they are in, they already have a problem!

PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 11

Application Lifecycle

Generalization



PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 12

The Testing Goal: Utopia

A repeatable process, which as much as possible is automated, that when utilized raises the level of application quality as close to 100% as possible, which balances the organization's requirement for "time to market", "quality" and "expense tolerance".

Sometimes this is easier said than done ...

Let's, consider the "parts of the puzzle"…

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Slide 13

A Repeatable Process

- **Test Plans** – Enables a team (or pooled resources) approach.
- **Automated Scripting Tool** – Enables repeated tests without manual intervention.
- **Automated Test Data Creation** – Enables utilization of "meaningful" data subsets with minimal manual labor.
- **Automated Comparison Tool** – Enables discovery of ALL changes – expected or not. Testing verification.

A Growing Trend:
The Automated Test Factory

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Slide 14

Scripting and Automation

- Repeatable Scripts – Minimal human intervention, speed, predictable results, "testing factory"
- Open Interfaces – Must be able to incorporate other tools, and those tools must be "open"
- Numerous Critical Features
  - Logical View of UI – (e.g. Grid Support)
  - Ability to "Record" and "Playback" THEN Augment…
  - Reusable Test Scripts – Modular, Utilize Record & Playback
  - Ability to Pinpoint Errors and Re-Execute Failed Scripts
  - Ability to Document Script Structure
  - Ability to "reasonably" administer Application Updates
  - "Good" support for all Platforms, and … needs to work in your environment w/ your equipment
  - Etc…

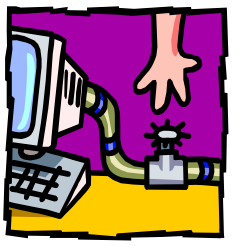
PRINCETON SOFTECH
A Computer Horizons Company

Slide 15

We get it right. Every time.™

Data, A Critical Element

- Want "meaningful" test data - critical to successful testing
- Production environment a good place to get realistic data, but it's not easy…



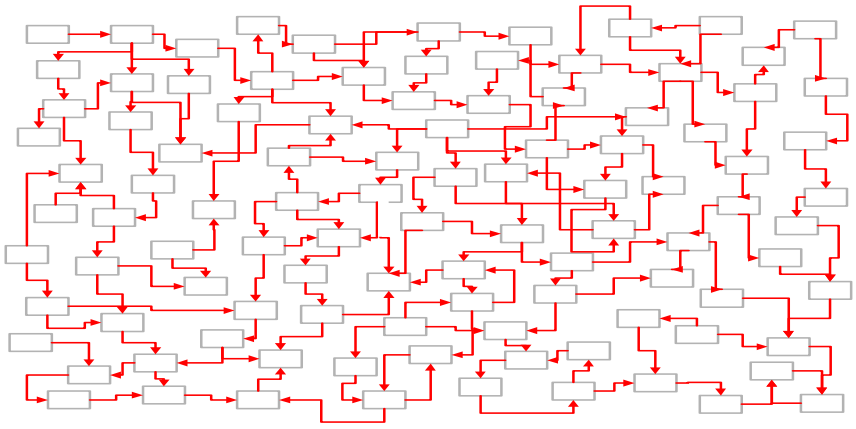
- Many organizations have used alternative approaches which have shortcomings
  - Cloning
  - Writing Extract Programs

PRINCETON SOFTECH
A Computer Horizons Company

Slide 16

We get it right. Every time.™

Capturing the "Right" Test Data



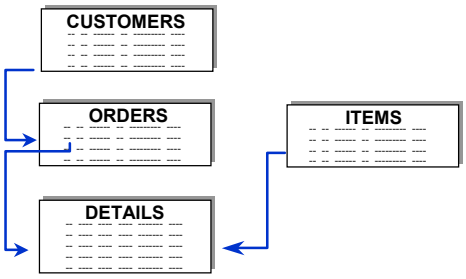
PRINCETON SOFTECH
A Computer Horizons Company

Slide 17

We get it right. Every time.™

Why is this Difficult?

Managing the Database Traversal



- **Extract: All ORDERS for all CUSTOMERS with any ORDERS outstanding for 60 days or more**
- **Extract: CUSTOMERS with any ORDERS which include the ITEMS screwdrivers**

PRINCETON SOFTECH
A Computer Horizons Company

Slide 18

We get it right. Every time.™

Test Data Creation

- **Option 1: Use Existing Data if you can…**

  Need to handle complex data models characterized by:

  - DB defined Referential Integrity, as well as Application enforced RI

  Typically the Application enforced RI does not adhere to the DB rules (e.g. "compatible" data types, composite columns, data driven)

  - Need to be able to define Test Data Criteria in a repeatable and convenient manner (i.e. automation)

- **Option 2: Synthesize New Data, if must…**

  - When there is no existing data

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Use of Existing Data

- **Selecting the Right Data – Data Partitioning**

  - Standard Selection Criteria – Of Course!
  - Random Selection
  - Partitioning / Grouping
  - Limiting the Data – By Table, By Relationship

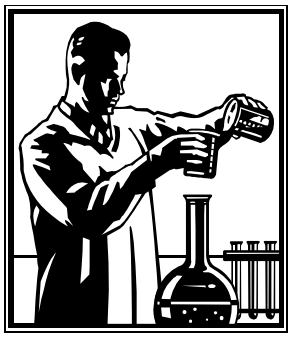
- **Transforming the Data**
  - Masking Sensitive Data
  - Altered Data Model
  - "Looking Up" Valid, Random Values

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Synthesizing Data

- Synthesizing the Right Data
  - Definition of Domains
  - Complete Subsets
- Multiplying Sets
  - Key Propagation



PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Slide 21

Creating the Test Database

- Test Database may not Exist
- Identical or Modified
- As a "fraction" of the Source
- Source and Target may be Heterogeneous
  - Need awareness of DDL and Data differences

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

Slide 22

Validating the Changes

- Finding the "needle in the haystack"
  - The "volume" problem
  - Related Changes
- Variety of Changes
  - Inserts, Deletes, Updates
  - Direct, Related
  - Ignore "expected" Changes
- Another "interesting" anomalies
  - Orphans, Duplicates, Altered Parents

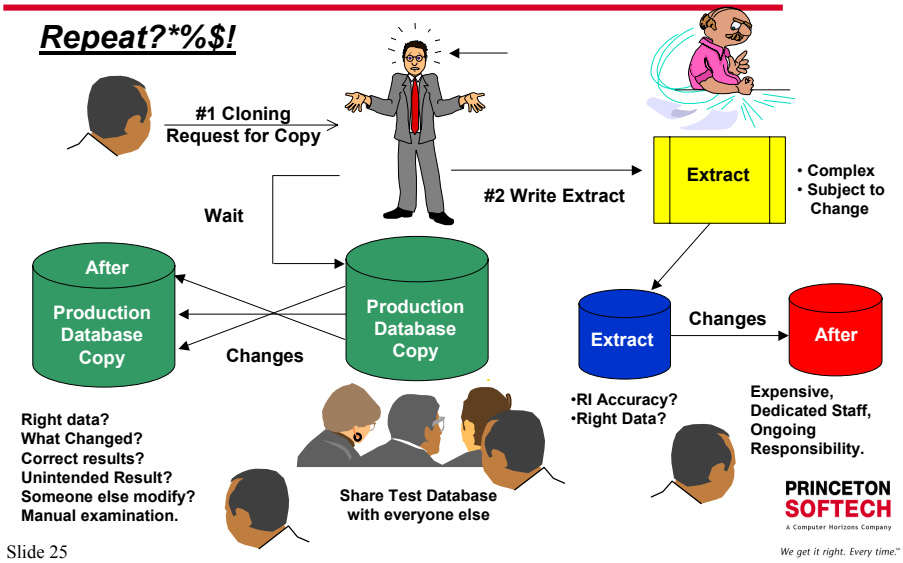
PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Miscellaneous Facilities

- Quickly Re-establishing the Test Environment
  - Testing is an Iterative Process
- Dynamic SQL vs. Load Utilities
  - Large Volumes
- Extracting Data from Image Copies
- Browse and Edit
  - Production environment may not have everything you need

PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

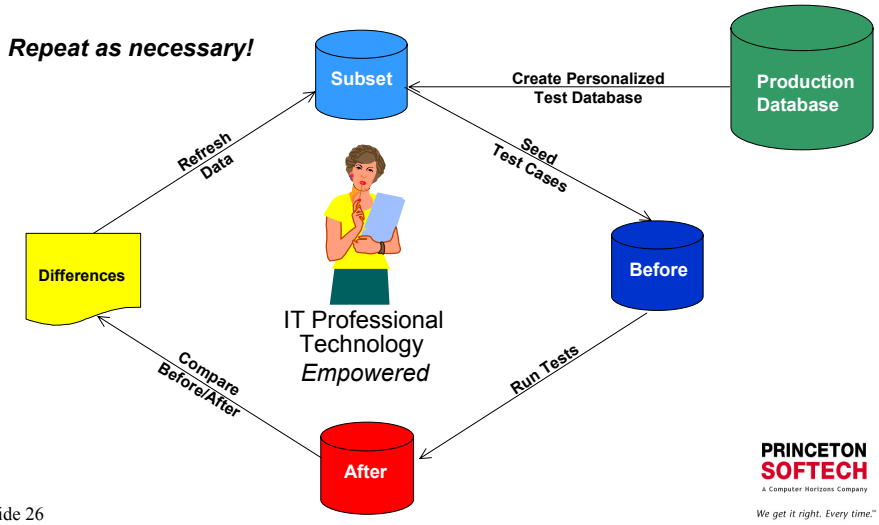
Testing: Typical "As Is" Process(es)



PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 25

Test Data: "Could Be" Process



PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 26

Gartner on "Solution Requirements"

- Automation Tools
  - Need to be able to repeat process over the course of the application life cycle
- Intelligent Test Data Generation Tools
  - Need to be able to repeatedly create realistic and manageable test data
  - Needs to understand application RI and be heterogeneous
- Intelligent Browsing and Editing Functionality
  - Needs to understand relational subsets
- Intelligent Data Comparison Tools
  - Need to understand relational subsets
- Stress Testing Tools
  - Need to accurately reflect user community

PRINCETON SOFTECH
A Computer Horizons Company
We get it right. Every time.™

Slide 27

Requirements for Testing in the DB Environment

Thank You

PRINCETON SOFTECH
A Computer Horizons Company

We get it right. Every time.™

QW2001 Paper 7T2

Mr. James Lyndsay
(Workroom Productions)

The Importance of Data In Functional Testing

Key Points

- Plan the data for maintenance and flexibility
- Know your data, and make its structure and content transparent
- Use the data to improve understanding throughout testing and the business

Presentation Abstract

A system is programmed by its data. Functional testing can suffer if data is poor. This presentation gives an understanding of the ways that data work fits into the overall test effort, and gives an overview of the ways that good data can be used to improve functional testing.

There are three kinds of test data;
* Environmental data tells the system about its technical environment.
* Setup data tells the system about the business rules.
* Input data is information input by day-to-day system functions. Some input data is fixed and available at the start of the test. Some is consumable, and forms the test input.

The presentation deals with how to recognise these types and their common problems during pre-test, testing and go-live.

Data can be loaded into the system manually, or by tools. The presentation discusses the advantages and pitfalls of various methods and suggests partitioning strategies to allow reliability and flexibility in the same dataset. The frequency and timing of data loading is also discussed.

Data maintenance is a substantial task, often comparable in size with test script maintenance. The presentation discusses common problems and possible solutions. A key solution is that good data content can help reduce the workload of data maintenance.

Good data can allow testing to carry on in areas not covered by the initial scripts and requirements. Naming conventions can help data to be accurate, and can make test results easier to interpret. A good test data structure promotes a common understanding and helps avoid mistakes. Accurate and appropriate content reduces the number of test process errors.

Data can help the business focus when requirements are vague. User involvement in data descriptions allows early insight into possible problems. The presentation

discusses the pitfalls and advantages of sourcing data from the business.

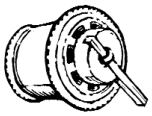
Before winding up, a brief mention is made of operational profiles, non-functional testing, and data verification before/during live operation.

About the Author

James Lyndsay is an independent test consultant with ten years experience. Specialising in test strategy, he has worked in a range of businesses from banking and telecomms to the web, and pays keen attention to the way that his clients' focus is shifting away from functional testing.

The Importance of Data in Functional Testing

James Lyndsay
Workroom Productions



Data is important to functional testing.

A system is programmed by its data.

Functional testing suffers if data is poor.

Good data is vital to reliable test results.

Good data can help keep testing on schedule.

Problems caused by poor data

The following problems can be caused by poor data;

- Unreliable test results
- Degradation of test data over time
- Increased test maintenance budget
- Reduced flexibility in test execution
- Obscure results and bug reports
- Larger proportion of problems can be traced to poor data
- Less time spent hunting bugs
- Confusion between developers, testers and business
- Requirements problems can be hidden in inadequate data
- Simpler to make test mistakes
- Unwieldy volumes of data
- Business data not representatively tested
- Inability to spot data corruption caused by bugs
- Poor database integrity

Topics

1) Recognising types of data

2) Avoiding common problems

Types of Data

Environmental data

tells the system about its technical environment.

Setup data

tells the system about the business rules.

Input data

information input by day-to-day system functions.

- **Fixed input data** is available at the start of the test.
- **Consumable input data** forms the test input.

Avoiding problems

Systems are programmed by their data . . .

If the functionality of your system is at all affected by the setup data;

TEST THE DATA

Why?

Problems found will;

1) Improve testing
2) Help get the data right before live operation
3) Help pinpoint bugs in live operation

How?

Data testing can be incorporated in functional testing, by looking at;

Data load and maintenance

Organising the data

Data and the business

Ways of loading data

- Using the system you're trying to test
  - Manually entered
  - Automated tool, making keystrokes into application
- Using a data load tool
  - All new data, created for testing
  - Old data, selected for testing / filtered and migrated
  - Complete set, migrated and loaded, identical but for personal details
- Not loaded
  - Already set up for testing / Left in the system
  - You are working on the live system

Frequency of data load

- At the start of testing
- With each release
- First thing Monday
- Whenever I want
- Before every test

NB: Straw poll of 'When does the data usually get loaded' also came back with the following answers;

- Before I ever got involved
- The developers left it there
- The last testers / tests left it
- Whenever we get enough time
- After we've found out what shape the database is
- When we know what it means

Data Maintenance

When and why?

Replacing consumed data

Repairing broken data

Responding to change - database schema, code, requirements

New test requirements

Problems:

Sizeable task - can be a substantial fraction of overall test maintenance

Prone to error

Performed by more than one group

Solutions to Loading and Maintenance headaches

1) **Automate data load and maintenance where possible**
2) **Control / measure data change**
3) **Recognise and prepare for problems**
4) **Use good data**

Good Data

Good data increases data reliability, reduces data maintenance time and can help improve the test process.

Good data assists testing, rather than hinders it.

Good data is based on;

1) **Permutations**
2) **Clarity**
3) **Partitions**

Data Permutations 1

Permutations are familiar from test planning

Utility Customer Care / Billing Example:

A customer can have one of three products (1,2,3). They may be billed Monthly or Quarterly, be High or Low value, and their last bill was either Paid or Unpaid. There are 3x2x2x2 = 24 combinations - and so the number of possible permutations climbs rapidly as system complexity increases.

By requiring that the list holds not all possible combinations, but all possible pairs, the list can be reduced. All possible pairs; M1, M2, M3. Q1, Q2, Q3. H1, H2, H3. L1, L2, L3. P1, P2, P3. U1, U2, U3. MH, ML, QH, QL, MP, MU, QP, QU. HP, LP, HU, LU.

The following six permutations contain all the pairs;

Customer	Account	Product	Care	Bill
1	M	1	H	P
2	M	2	L	P
3	M	3	H	U
4	Q	1	L	U
5	Q	2	H	U
6	Q	3	L	P

Data Permutations 2

Permutation is appropriate when:

Fixed input data consists of many rows

Fields are independent

You want to do many tests without loading / you do not load fixed input data for each test.

Permutation helps because:

Achieves good test coverage without having to construct massive datasets

Can perform investigative testing without having to set up more data

Can be used to test other data - particularly setup data

Permutation is familiar from test planning.

Reduces the impact of functional/database changes

Clarity

Developers and the Business don't need to understand test data / data requirements - so some of them won't. We can make our data clearer by using available free text fields;

Customer Name	Account	Product	Care	Bill
HP1 Monthly	M	1	H	P
LP2 Monthly	M	2	L	P
HU3 Monthly	M	3	H	U
LU1 Quarterly	Q	1	L	U
HU2 Quarterly	Q	2	H	U
LP3 Quarterly	Q	3	L	P

Clarity helps because:

Improves communication within and outside the team

Reduces test errors caused by using the wrong data

Helps when checking data after input

Helps in selecting data for investigative tests

Partitioned data

Data load/reload can be inconvenient. Data can be partitioned into:

1) **Safe area**

   Used for enquiry tests, usability tests etc. No test changes the data, the area can be trusted. Many testers can use simultaneously

2) **Change area**

   Used for tests which update/change data. Data must be reset or reloaded after testing. Used by one tester at a time.

3) **Scratch area**

   Used for investigative update tests and those which have unusual requirements. Existing data cannot be trusted. Used at own risk!

Data can be partitioned by machine / database / instance. Can also be partitioned by disciplined use of text / value fields.

Data and the Business

'The Business' is good at looking at data;

- Easier to understand than tests
- Can be compared with existing systems

Advantages

- Helps focus when requirements are vague
- Helps UAT
- Increases trust and understanding
- Helps early user identification of problems

Disadvantages

- Data creep
- Vague requirements can lead to vague data
- Incomplete data can lead to incomplete testing

Further data issues

Operational Profiles

Non-functional testing

Data verification

Topics: conclusion

1) Recognising types of data

Environmental Data
Setup Data
Input Data - Fixed and Consumable

2) Avoiding common problems

Recognise the problems
Automate loading and maintenance
Test your data
Use good data
Involve the Business

Summary

- Plan the data for maintenance and flexibility
- Know your data, and make its structure and content transparent
- Use the data to improve understanding throughout testing and the business

James Lyndsay
Workroom Productions
www.workroom-productions.com
jdl@workroom-productions.com

QW2001 Paper 8T1

Mr. J. D. Brisk
(Exodus Communications)

Peer-to-Peer Computing: The Future Of Internet Performance Testing

Key Points

- Web testing: Load/Stress testing
- Peer-to-Peer Computing
- Distributed Computing

Presentation Abstract

In this presentation, JD Brisk, Managing Director of Exodus Performance Labs, will discuss how Peer-to-Peer Computing will revolutionize Web performance testing by creating the world's largest, most realistic testing environment possible. Specifically, he will address the opportunities and challenges of using the Peer-to-Peer Computing method of Web testing.

Peer-to-Peer Computing, - individual computers exchanging data without a central server - has been around for more than 20 years in various forms. It works by taking large tasks and dividing them into many smaller tasks, all of which are disseminated to many computers running simultaneously via a network such as a private corporate network, or the Internet. After the tasks are processed, block-by-block via individual computers, the data is transmitted back to a central server that then assembles an answer.

Most recently, Napster and the SETI@home project have helped increase awareness of the Peer-to-Peer, also known as Distributed Computing. However, a more powerful use of the technology is not to exchange music, but to combine the processing power of thousands of networked PCs to create a virtual supercomputer. This testing environment provides an unprecedented level of reality-based testing that is expected to propel the limits of Web site performance, capacity and scalability.

Peer-to-Peer Computing opens a new door of possibilities for organizations to use distributed bandwidth resources to realistically perform large-scale stress, load and scalability testing of e-commerce Internet sites. With this technology, organizations can improve the efficiency and accuracy of Internet testing methods.

Exodus Performance Labs is using Peer-to-Peer Computing to create a testing environment that utilizes hundreds of thousands, and potentially millions, of real web clients that characterize the variants found in real life. This real-world testing

model uses the power of the Internet to test the Internet.

Hundreds, even thousands of simulated users created on one machine are generally not representative of real world situations. The Distributed Computing model greatly expands testing capabilities by utilizing real user machines in diverse locations and introduces the actual variants found daily on the Internet.

Recent studies report that about one billion personal computers, each with an average processing speed of 500 megahertz, are now connected to the Internet. Leveraging this user base gives creates a large, diverse pool of resources at a fraction of the cost of buying the machines and building the environment. With this technology, Web testing companies can create and utilize the worlds largest test environment.

The Peer-to-Peer Computing testing model not only exercises the transaction processing systems, in-route components like firewalls and load balancers, but also provides the flexibility to test based on select user demographics incorporating the "last mile" which previously has not been addressed. Companies now have access to load testing that's as close to live as you can get because it uses real client machines in homes and small businesses around the world with real variants found in real life with the advantage of repeatability from scripted applications.

About the Author

J.D. Brisk is Managing Director of Exodus Performance Labs, formerly KeyLabs. Prior to its acquisition by Exodus, J.D. was President and CEO of KeyLabs, based in Linden, UT. Initial COO and one of the original founders of KeyLabs, J.D. became President and CEO after he and two colleagues formed another new company in August 1998 called Altiris and spun out their highly successful software business. Somewhat of a maverick in the industry, JD is known for his unique ability to get things done. He worked as a hardware/software test engineer and technical manager for 5 years prior to joining Novell in 1985 where he spent 11 years in various technical management positions. At Novell, JD managed Sustaining Engineering and most of the other core Testing Departments. He pioneered the concept of behavioral testing. He designed and implemented the Corporate Interoperability Testing programs and as Director of Engineering in Novell Labs, designed and implemented the "YES" and "Tested and Approved" Software Certification Programs. He was also responsible for the world's largest network test facility.

Exodus Performance Labs

Peer-to-Peer Computing: The Future of Internet Performance Testing

J.D. Brisk, Vice President

Exodus Performance Labs

THE INFRASTRUCTURE FOR THE DIGITAL ECONOMY™

EXODUS®

Topics for Discussion

- Evolution of Website testing
- Simulated Load Testing
- In-Lab Load Testing
- Expanded IDC Load Testing
- Distributed P2P Load Testing
- Case Study – Follow the Sun
- Methodology Study – Simulated vs. Distributed Load Testing
- Customer Story – Zoom Culture
- Future –other uses-
- Exodus Performance Labs
- Questions

2

Concern for Site Performance

The success of a Website depends largely on its performance.

- Page download times
- Functionality of the Web-based applications
- Thoroughness of the transaction processing systems

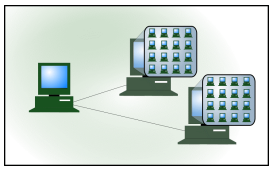
3

Evolution of Web Site Testing

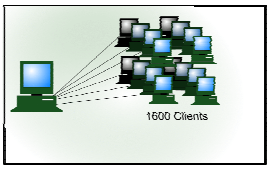
- **Simulated Load Testing**
  - Using 1 or 2 machines to simulate thousands of users either over the net or in-house
- **Lab Environment Load Testing**
  - Use multiple machines to create an environment of realistic of users
- **Expanded IDC Load Testing**
  - Utilize distributed machines to re-recreate real users from various locations
- **Distributed Peer to Peer Load Testing**
  - Using actual users, spread across the world, to provide real load and stress testing.

4

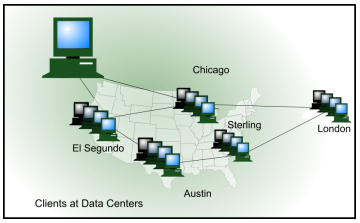
Evolution of Web Site Testing



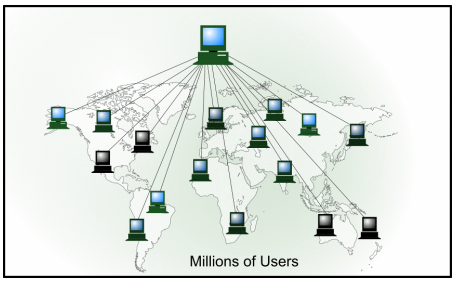
Simulated Load



Lab Environment



Extended IDC

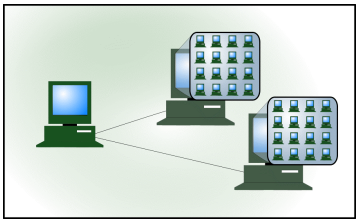


Distributed Peer-to-Peer

5

Simulated Load Tests

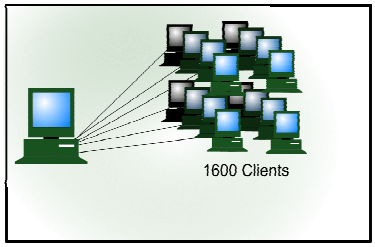
- **Pro**
  - Allows the use of existing resources
  - Synthetic simulation of many users
  - Low barrier to entry
- **Con**
  - Unrealistic connectivity (mostly inside the firewall)
  - Equipment Under Test is usually local
  - Requires machines, expertise, and time
  - Work load at server not accurate representation of real users
  - Inaccurate correlation between baseline and heavy loads
  - Generally located at single location
  - Tools often misused
  - False feeling of security
  - Limited scalability
  - Last mile not taken into account



6

In-Lab Load Testing Environment

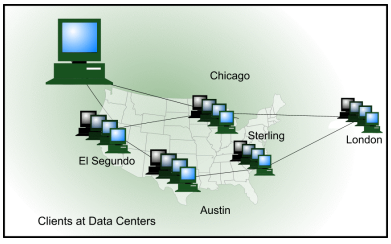
- **Pro**
  - Target Website remain at customer site or in IDC
  - More realistic in terms of re-creating the environment of the Internet
  - Scalable for most websites
  - Considerably less expensive than setting up own lab
- **Con**
  - Single route
  - No distributed testing capabilities
  - Limited scale; number of simultaneous tests
  - Last mile not taken into account



7

Expanded IDC

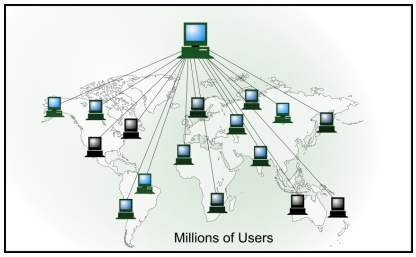
- **Pro**
  - Website locations not an issue
  - Even more realistic in terms of recreating the environment of the Internet
  - Not limited by bandwidth
  - Greater number of multiple routes
  - Able to run multiple, simultaneous tests
  - More cost effective
  - Semi-distributed
- **Con**
  - Equipment is always "down the hall"
  - Last mile not taken into account



8

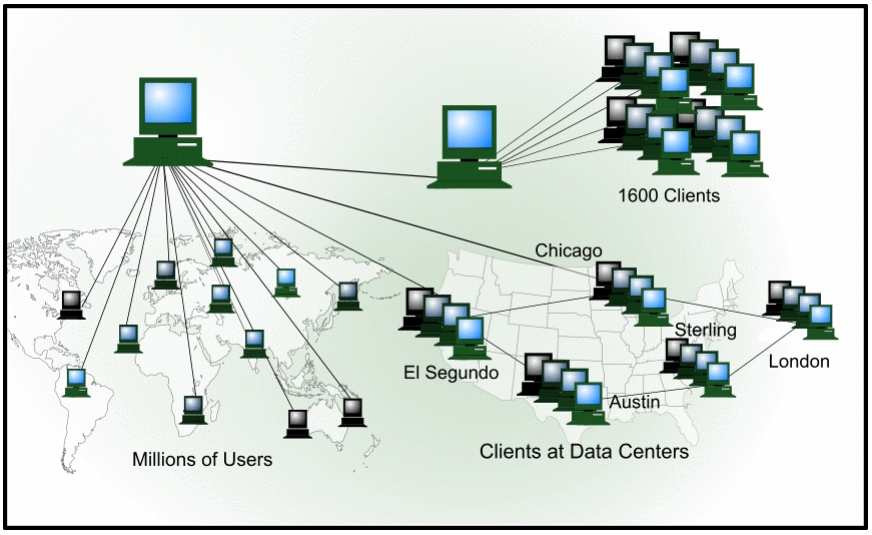
Distributed P2P Model

- **Pro**
  - Unlimited testing capabilities
  - REAL: using the Internet to test the Internet
  - Utilizing various and diverse ISPs
  - Ability to test target site using machines in target site area worldwide
  - Multiple, simultaneous tests from any location around the world
  - Monitoring from any geographic location
  - Allows for testing of the Last Mile
  - Cost effective
- **Con**
  - Machines not under lab environment



9

Cumulative Load Generation Capabilities



10

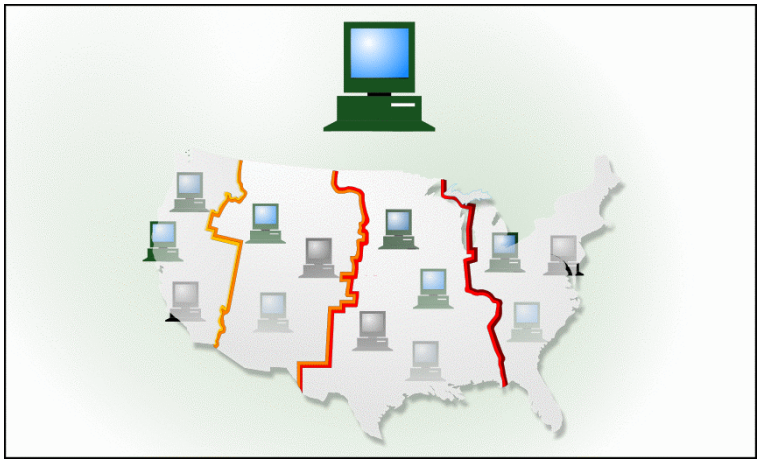
EXODUS®

Case Study –Follow the Sun-

- A Website sees traffic patterns which indicate excessive numbers of users are accessing the site early in the morning from each time zone throughout the United States.
- The customer wants to understand the end user experience in each of the time zones and make modifications to the site during these heavy loads, but cannot afford to have the site perform worse or go down during business hours.

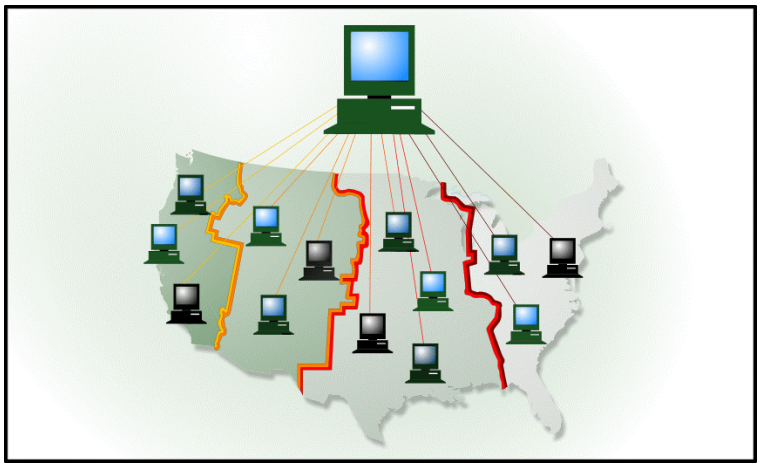
11

EXODUS®



12

EXODUS®



13

EXODUS®

Methodology Study – Simulated vs. Distributed Load

Testing the load testing tools

- **Test Constants**
  - 15 to 150 users browsing the site
  - Pentium II 450 with 128MB RAM
- **Distributed Load**
  - 15 machines driving load
- **Simulated Load**
  - 1 machine driving load

14

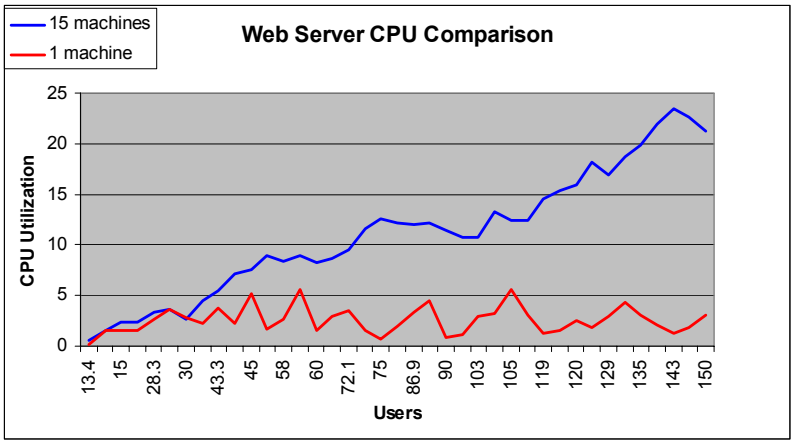
Methodology Study – Areas Monitored

- **Web Server CPU Utilization**
  - The web server CPU utilization shows how much work is being exerted on the server
- **Script Execution Time**
  - The execution time is the time that it took to execute the script
- **Get Requests Per Second**
  - The get requests are monitored to determine how many get requests are being issued against the web site by the test tool
- **Throughput**
  - The throughput is monitored to determine how many Bytes per Second are being requested from the server by the test tool

15

Methodology Study – Test Results cont.

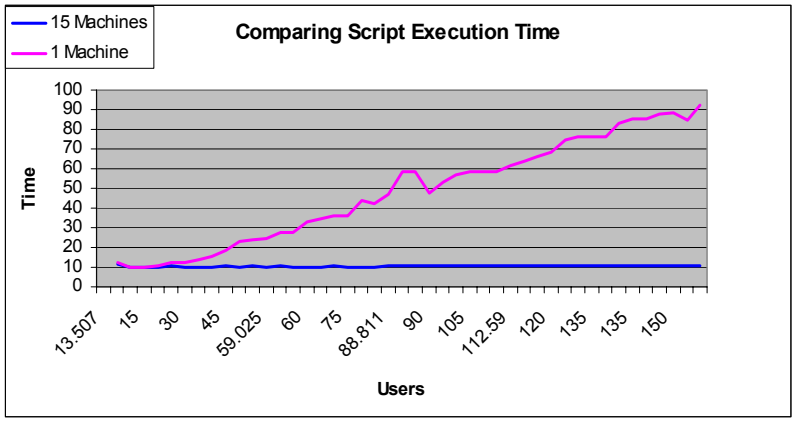
- 150 users running on 1 machine do not exert the same amount of work on the web server as 150 users distributed between 15 machines



16

Methodology Study – Test Results

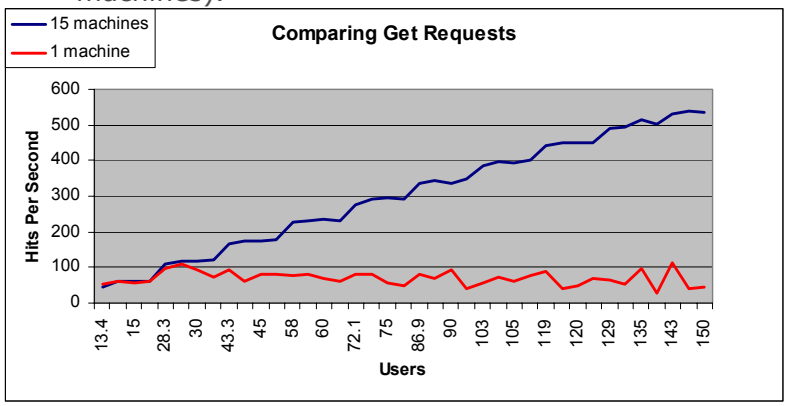
- 1 machine running 150 users is not able to process the scripts a efficiently as 15 machines running the same script distributing the (90 second average on 1 machine, 10 second average on 15 machines).



17

Methodology Study – Test Results cont.

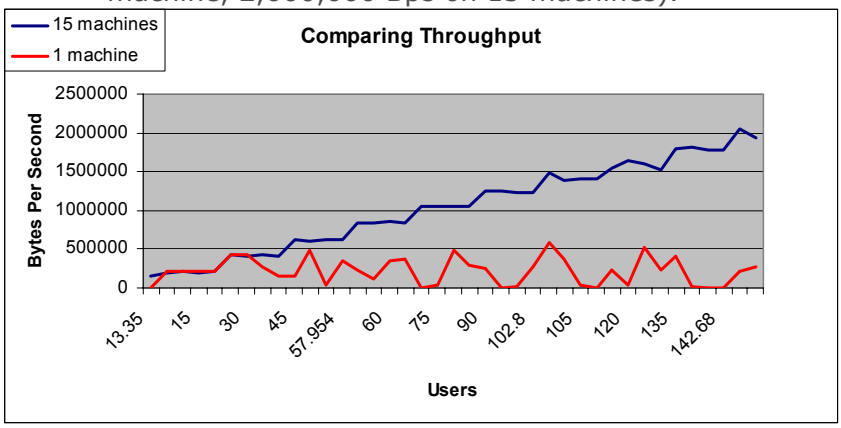
- 1 machine running 150 users cannot issues get requests as efficiently as 15 machines running the same script distributing the users (111 hits per second on 1 machine, 536 hits per second on 15 machines).



18

Methodology Study – Test Results cont.

- 1 machine running 150 users cannot transfer data a efficiently as 15 machines running the same script distributing the users (215,000 Bps on 1 machine, 2,000,000 Bps on 15 machines).



19

Methodology Study – Conclusions

- **Unrealistic simulated loads on a single machine will not provide accurate results**
- **Machine limitations**
- **Distributing the load will provide a more accurate and realistic user load**
- **"Realistic" users vs. "Simulated" users**
- **Which testing methodology are you using now?**
- **Which is right for your application?**

20

Case Study – Zoom Culture

- **North Carolina based media company**
- **"Zoom Directors" shoot digital video content broadcast by ZC.TV, Fox and Sports Networks**
- **Board raised questions of site robustness… simply wanted to know the "breaking point"**
- **Considered buying tool and using simulated user service from a competitor**
- **Our environment matched well their needs,**
  - Unique streaming video capabilities negated options for simulated users
  - Connect, register and download metrics
  - ramped tests
  - fix as we go
- **Time was of the essence**

21

Case Study – Zoom Culture

Initial results:

- **Site designed for concurrent 250 users**
  - **failed at 5 concurrent requests**

After project completion:

- **700% increase** in the number of concurrent streaming users

22

Case Study – Zoom Culture

"Working with Exodus Performance Labs was great. The load testing process was interactive so while it was in progress we could talk about what was happening as it happened. The test engineer made recommendations that we agreed upon and within a couple of hours, they were implemented. It was tremendous. We have plans to test with Performance Labs again."

-Bill Graham, Vice President of Technology Operations

23

Exodus Performance Labs Background

- KeyLabs formed in January 1996
- Acquired by Exodus in February 2000
- We've been doing this for over 5 years
- Focus on networking gear. HW, SW, Systems and Infrastructure components and Internet testing
- Full service test lab
- Certification programs
- PEN Testing, Vulnerability Scans

24

The feeling you will have…

25

Summary

- **Risk Taker?**
- **Use the right methodology for your application**
- **Consequences (as per Bill at Zoom Culture)**
  - Unwilling to burden potentially huge financial consequences
  - Annoyed, even lost customers
  - Decreased investor' confidence
  - Negative perception in public eye
- **Unrivaled capabilities/resources/expertise to understand your needs and architect tests accordingly**
- **We're your Partner and extension of your resources**
- **Provides "peace-of-mind" or "controlled fear"**

26

Questions?

QW2001 Paper 8T2

Mr. Erik Simmons
(Intel Corporation)

Quantifying Quality Requirements using Planguage

Key Points

- Designed to quantify qualitative statements in plans, specifications, and designs, Planguage is a keyword-driven language that allows measurable, testable quality requirements to be written.
- Planguage has many benefits; it is easy to learn, compact, extensible, and provides a consistent way to specify quality requirements.
- Examples and experiences introducing Planguage at Intel are provided.

Presentation Abstract

"Planguage" is a new industrial engineering language, designed for planning, projects and processes. Developed by Tom Gilb and others, it is a new language for communicating about engineering and management work. Users of planguage include Intel, IBM, HP, Ericsson and Boeing.

Planguage is a keyword-driven language. Simple to learn, it also prevents omission of critical information when specifying quality requirements. The resulting requirements are less ambiguous and more measurable than requirements written using other syntaxes or methods.

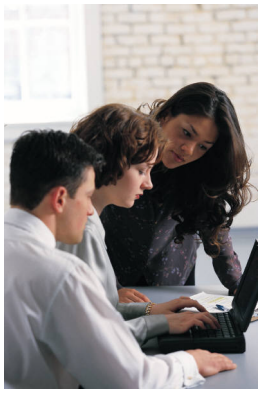
This workshop presents Planguage keywords and syntax, and then uses actual examples from Intel to illustrate how Planguage can simplfy, clarify, and improve quality requirement specification using a "before and after" format. Students participate in several exercises to help drive home the concepts, and are invited to bring their own requirements documents for use during the final exercise.

About the Author

Erik Simmons has 15 years experience in multiple aspects of software and quality engineering. Erik currently works as a Platform Quality Engineer within the Corporate Quality Network at Intel Corporation. He leads the corporate Software Engineering Process Team that is charged with improving software development capabilities across Intel's product development groups, and is responsible for Intel's product requirements engineering practices.

Quantifying Quality Requirements Using Planguage

Erik Simmons, Intel Corporation



TAG: {*collection*}

GIST:

SCALE:

METER: <*fuzzy concept*>

MUST: RECORD:

PLAN: ←*source*

[*qualifiers*]

PAST:

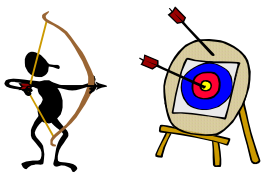
TREND:

Quantifying Qualitative Requirements

The system must be easy to learn. ← *Can you test this?*

How about:

The system must be used successfully to place an order in under 10 minutes without assistance by at least 80% of test subjects with no previous system experience.



Can we do even better?

What is Planguage?

Created by Tom Gilb, Planguage stands for **P**lanning **Language**, a simple but powerful set of keywords and syntax that can be used within:

- Requirements Specifications
- Business Plans, Success Criteria, Vision Statements
- Design documents, Strategies, etc.

Planguage aids communication about complex ideas.

Why use Planguage?

Planguage:

- Permits quantification of qualitative statements
- Parameterizes those statements to prevent omissions
- Offers better clarity and comprehension than ordinary structured English
- Captures lots of information in a small space
- Excels at expressing quantified quality requirements
- Can be customized or extended for different environments and new uses
- Is intuitive and simple enough to be used with almost any audience

Common Planguage Keywords

TAG: A unique, persistent identifier

GIST: A short, simple description of the concept contained in the Planguage statement

STAKEHOLDER: A party materially affected by the content of the statement

SCALE: The scale of measure used to quantify the statement

METER: The process or device used to establish location on a SCALE

MUST: The minimum level required to avoid failure

PLAN: The level at which good success can be claimed

STRETCH: A stretch goal if everything goes perfectly

WISH: A desirable level of achievement that may not be attainable through available means

Common Planguage Keywords

PAST: An expression of previous results for comparison

TREND: An historical range or extrapolation of data

RECORD: The best known achievement

DEFINED: The official definition of a term

AUTHORITY: The person, group, or level of authorization

Fuzzy concepts requiring more details: <*fuzzy concept*>

Qualifiers (used to modify other keywords): [*when, which, …*]

A collection of objects: {*item1, item2, …*}

The source for a statement: ←

Learnability

TAG: Learnable

GIST: The ease of learning to use the system.

SCALE: Time required for a Novice to successfully complete a 1-item order using only the online help system for assistance.

METER: Measurements obtained on 100 Novices during user interface testing.

MUST: No more than 7 minutes 80% of the time

PLAN: No more than 5 minutes 80% of the time

WISH: No more than 3 minutes 100% of the time

PAST [our old system]: 11 minutes ← *recent site statistics*

Novice: DEFINED: A person with less than 6 months experience with Web applications and no prior exposure to our Website.

Planguage in Practice

In practice, the TAG keyword is often dropped and the Tag used in place of the GIST keyword. For example, the learnability requirement could be written:

LEARNABLE: The ease of learning to use the system

instead of

TAG: Learnable

GIST: The ease of learning to use the system

This convention is used in most of the examples that follow. You may use either format in your work.

Using Qualifiers

Qualifiers allow for precise description of conditions and events. They add richness,precision, and utility to Planguage.

Examples (not related to each other):

PLAN **[Q1 '00]**: 20,000 units sold

MUST **[First year]**: 120,000 units sold

WISH **[First release, enterprise version]**: 1 Dec. 2000

PLAN **[US market, first 6 months of production]**: Defects Per Million < 1,000

METER **[Prototype]**: Survey of focus group

METER **[Release Candidate]**: Usability lab data

Finding Scales

Scales exists for just about any concept. Here are some helpful hints for locating/defining scales:

- Divide the measured quality into its elementary components first if possible
- Use known, accepted scales of measure when possible
- Derive new scales from known scales by substituting terms
- Incorporate qualifiers in the scales to increase usefulness and specificity
- Don't confuse scale with meter
- Share effective scales with others

Examples of Scales

Software Security: Time required to break into the system

Software Maintainability: Average engineering time from report to closure of defects reported prior to release

Software Reliability: The Mean Time to Failure (MTTF) of the system

Software Learnability: Average time for <novices> to become <proficient> at a defined set of tasks (this can be measured on competing prototypes)

Meters

First, study the scale carefully. If no meter comes to mind:

- Look at references, handbooks, examples, etc. for ideas
- Ask others for their experience with similar methods
- Look for examples within test procedures

Once you have a candidate, check to see that:

- The meter is adequate in the eyes of all stakeholders
- There is no less-costly meter available that can do the same job (or better)
- The meter can be measured *before* product release or completion of the deliverable

Examples of Meters

Software Security: An attempt by a team of experts to break into the system using commonly available tools

Software Maintainability: Analysis of at least 30 consecutive defects reported and corrected during development

System Reliability: A Probability Ratio Sequential Test demonstration with α=10%, β=10%, Discrimination Ratio = 3

Software Learnability: UI testing & HCI usability tests, survey responses from focus groups, etc.

An Actual Requirement

As written:

The third key requirement is power consumption. Generally, the power consumption requirements are driven by noise requirements, or CE compatibility. The customers expressed the need for lower active power consumption so that passive cooling can be used. However, this is one possible implementation, and other implementations need to be addressed by engineering. Standby power consumption should meet the levels obtained by CE devices; 5-10W, and be achievable with the fan off. Cost is a factor. 10W standby is acceptable if the implementation cost is less than that of 5W standby. These requirements were articulated by *Company1*, *Company2*, *Company3*, *Company4*, and *Company5*.

An Actual Requirement

Rewritten using Planguage:

STANDBY: Standby Power Consumption ←{*Company1, Company2, Company3, Company4, Company5*}

GIST: The amount of power consumed by the system with the fan off and the HDD not spinning

SCALE: Watts

METER: Measurement on 3 units for 10 seconds at 23°C, ± 2°C

MUST: 10W

PLAN[*CostOK*]: 5W

CostOK: Design and manufacturing costs do not exceed 10W cost by more than 25%

NOTE: Relates to noise and CE compatibility requirements. Passive cooling within the system is desired.

Defect Correction

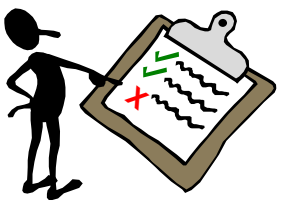
Maintainability.Debug: The ease of correcting defects in the system.

SCALE: Average engineering hours needed to correct defects once they are located.

METER: Measurement of 50 defects corrected during system testing (all severities).

MUST: Less than 12 hours average

PLAN: Less than 6 hours average



Scalability

Scalability.CPU: The CPU usage pattern under increasing application stress.

SCALE: Minimum application transactions per second required to sustain 100% CPU utilization for at least 15 seconds.

METER: Stress testing of the application using automated software drivers and a representative operational profile.

MUST [Single Processor, 500MHz]: At least 45 TPS

PLAN [Single Processor, 500 MHz]: At least 60 TPS

Security

Security.Access: The resistance of the system to <unauthorized access>.

SCALE: Time required to obtain <unauthorized access> to the system using commonly available tools and techniques.

METER: Attempted <access> by a team of two skilled security engineers with no special knowledge of the system.

MUST: At least 8 hours

PLAN: At least 16 hours

Usability/Documentation Quality

Usability.UserGuide: The usefulness of the user guide.

SCALE: Average response to usability survey questions, scored on a 5-point scale.

METER: Surveys administered to end-users who are members of the product focus group.

MUST [Gold]: Average response > 4

PLAN [Gold]: Average response > 4.5

Planguage Templates

Planguage can be used to create requirements templates for reuse:

Usability: The ease of use of the system under stated conditions

SCALE: Minutes on average for <target users> to complete <a defined set of tasks> correctly using the system

METER: Testing prior to release using <*n* target users>

MUST [first release]: *z* minutes or less, 90% of the time

PLAN [prototype complete]: *x* minutes or less, 80% of the time

PLAN [first release]: *y* minutes or less, 90% of the time

Sub-Parameters for METER

As one example of Planguage extensibility, the METER keyword has been detailed using four sub-parameters.

METHOD: The method for measuring to determine a point on the Scale

FREQUENCY: The frequency at which measurements will be taken

SOURCE: The people or department responsible for making the measurement

REPORT: Where and when the measurement is to be reported

An Example with Sub-Parameters

Before

METER: Measurement of 50 randomly-selected defects corrected during system testing (all severities).

After

METER: Measurement of 50 defects corrected during system testing (all severities).

- METHOD: Random selection from defect logs
- FREQUENCY: Once prior to Beta milestone; repeated if needed based on outcome
- SOURCE: SQA Lead, based on data from Test Lead
- REPORT: Weekly product development team meeting

For More Information

The following are all authored by Tom Gilb:

Competitive Engineering, available free at http://www.result-planning.com.

Requirements-Driven Management using Planguage, available free at http://www.result-planning.com.

Principles of Software Engineering Management, Addison Wesley 1988

Quantifying the Qualitative, available free at http://www.result-planning.com.

A Requirements Engineering Language, available free at http://www.result-planning.com.

intel

Quantifying Quality Requirements Using Planguage

Erik Simmons
Intel Corporation
JF1-46
2111 NE 25th Ave.
Hillsboro, OR 97214-5961
erik.simmons@intel.com

Version 1.1, 03/30/01

Prepared for Quality Week 2001

Key Words: Product Quality; Quality Requirements; Quantified Quality

Author Biography

Erik Simmons has 15 years experience in multiple aspects of software and quality engineering. Erik currently works as Platform Quality Engineer in the Platform Quality Methods group, part of the Corporate Quality Network at Intel Corporation. He is responsible for Requirements Engineering practices at Intel, and lends support to several other corporate software and product quality initiatives. Erik is a member of the Pacific Northwest Software Quality Conference Board of Directors. He holds a Masters degree in mathematical modeling and a Bachelors degree in applied mathematics from Humboldt State University in California.

Abstract

Within the last decade, requirements engineering has benefited from increased attention. Several good books are now available, from general textbooks on requirements engineering to specific monographs on advanced topics. Among the many benefits has been an increased awareness of the importance of specifying quality requirements. However, outside of structured English, few methods for specifying quality requirements have been established. Planguage, created by Tom Gilb, is one notable exception. Designed to quantify qualitative statements in plans, specifications, and designs, Planguage is a keyword-driven language that allows measurable, testable quality requirements to be written. Planguage has many benefits; it is easy to learn, flexible, compact, extensible, and prevents omissions by providing a consistent set of parameters for quality requirements. In this paper, Planguage keywords and syntax are introduced. Examples of quality requirements before and after using Planguage are given, and the experiences of introducing Planguage within a product engineering environment are discussed.

Introduction

The last decade has seen an increased focus on the methods, process, and benefits of good requirements engineering. In the past few years alone, several very good books have been published on the topic. Undergraduate and graduate programs now more commonly introduce students to the fundamental concepts and techniques of requirements engineering.

Despite these and other advances, few techniques are taught for properly specifying quality attributes like performance, reliability, scalability, and ease of use. In most cases, structured English sentences are used to express the underlying requirements using terms that are difficult or impossible to test adequately. Qualitative terms like *easy, fast, reliable, secure, scalable, efficient, robust,* and a host of others are fertile ground for misunderstandings between product stakeholders.

Planguage was created by Tom Gilb in order to overcome these problems by quantifying qualitative terms [Gilb01, Gilb97a, Gilb97b]. Planguage is a keyword-driven language whose name is derived from a contraction of the words planning and language[1]. Planguage can be used in requirements specifications, design documents, plans, and other places where qualitative statements are common. Its primary benefits are quantifying the qualitative and improving communication about complex ideas. In addition to these, Planguage has several other desirable features and benefits:

Ease of Learning and Use

Planguage can be taught effectively to individuals and groups in a short period. At Intel, Planguage is covered in only a few hours as part of the requirements engineering curriculum. Although this brief exposure is not enough to guarantee successful adoption and use of Planguage, when combined with a small amount of follow-up mentoring and a catalog of examples the results have been quite good. More than 1,200 students at Intel have been exposed to Planguage within the past 12 months, and Planguage has made its way into many product development efforts. It is used by engineering, quality assurance, marketing, and program management alike in a widening array of documents, plans, and designs.

Flexibility and Extensibility

Planguage is designed to be extensible and customizable to fit local needs. This includes the addition of keywords and the rich structure of Planguage, with its ability to create and label statements, collections, and other internal structures for reuse. These properties have made Planguage popular and useful across differing product development efforts – an essential capability in order to obtain broad adoption and use in as diverse an environment as Intel.

Prevention of Omissions

One of the most powerful benefits of Planguage is its ability to prevent omissions when quantifying qualitative statements. Because keywords are prescribed for all the important dimensions, users of Planguage are less likely to omit necessary information. Planguage is equally effective in this regard whether implemented as a table within a document or as part of an automated requirements repository. In both cases, users praise its ability to bring issues to light through its complete, separate, and consistent treatment of the important dimensions of quantification.

Separation of Success and Survival

When considering qualitative concepts, there are usually many levels of achievement (or a range of achievement) possible. The question is not whether a system is reliable or secure, but *how* reliable or secure. Planguage excels at expressing these ideas through its use of more than one level of achievement. By allowing for specification of the best recorded level of performance, the

[1] The term Planguage is also used as the name of some programming languages for parallel processors, but that use is not related to its use in this paper.

optimum level, the planned level, and the level below which financial or political failure occurs, Planguage paints a detailed and complete picture of success and survival, allowing for informed, due-diligent decision making.

Planguage Keywords & Syntax

Planguage has a rich set of keywords. The commonly used keywords are given in Table 1.

Table 1: Planguage Keywords

TAG	A unique, persistent identifier
GIST	A short, simple description of the concept contained in the Planguage statement
STAKEHOLDER	A party materially affected by the requirement
SCALE	The scale of measure used to quantify the statement
METER	The process or device used to establish location on a SCALE
MUST	The minimum level required to avoid failure
PLAN	The level at which good success can be claimed
STRETCH	A stretch goal if everything goes perfectly
WISH	A desirable level of achievement that may not be attainable through available means
PAST	An expression of previous results for comparison
TREND	An historical range or extrapolation of data
RECORD	The best-known achievement
DEFINED	The official definition of a term
AUTHORITY	The person, group, or level of authorization

As an example of the extensibility of Planguage, four sub-keywords have been created for the keyword METER. The sub-keywords are designed to add precision and specificity to the METER statement, and are given in Table 2.

Table 2: Sub-keywords for the METER Keyword

METHOD	The method for measuring to determine a point on the Scale
FREQUENCY	The frequency at which measurements will be taken
SOURCE	The people or department responsible for making the measurement
REPORT	Where and when the measurement is to be reported

Besides keywords, Planguage also offers several convenient and useful sets of symbols:

- Fuzzy concepts requiring more details are marked using angle brackets: <fuzzy concept>
- Qualifiers, which are used to modify other keywords, are contained within square brackets: [when, which, …]
- A collection of objects is indicated by placing the items in braces: {item1, item2, …}
- The source for a statement is indicated by an arrow: Statement ← source

Using Qualifiers

Qualifiers allow for precise description of conditions and events. They add richness, precision, and utility to Planguage. Here are several (unrelated) examples of qualifier use:

PLAN **[Q1 '00]**: 20,000 units sold
MUST **[First year]**: 120,000 units sold

WISH **[First release, enterprise version]**: 1 Dec. 2000
PLAN **[US market, first 6 months of production]**: Defects Per Million < 1,000

METER **[Prototype]**: Survey of focus group

METER **[Release Candidate]**: Usability lab data

A Basic Application of Planguage

Requirements often contain statements like the following:

> *" The system must be easy to learn."*

When presented with this first requirement, nearly everyone would agree that it is not testable as written. It is up to the tester or someone else downstream to decide what "easy" is, what "learn" means, and how to test whether the product meets minimum levels of goodness.

A second common form of the statement of usability is made in structured English:

> *" The system must be used successfully to place an order in under 10 minutes without assistance by at least 80% of test subjects with no previous system experience."*

This is an improvement over the first requirement, and represents the typical state of the practice. The second wording gets a better response for testability, and many believe that they could write and execute tests for it.

Here is the Planguage version:

TAG: Learnable
GIST: The ease of learning to use the system.
SCALE: Time required for a Novice to complete a 1-item order using only the online help system for assistance.
METER: Measurements obtained on 100 Novices during user interface testing.
MUST: No more than 7 minutes 80% of the time
PLAN: No more than 5 minutes 80% of the time
WISH: No more than 3 minutes 100% of the time
PAST [our old system]: 11 minutes ← *recent site statistics*
Novice: DEFINED: A person with less than 6 months experience with Web applications and no prior exposure to our Website.

This statement provides a great deal of information in a compact format. Additionally, it is testable and far less ambiguous than the previous structured English statement.

Finding Scales and Meters

Scales exists for just about any concept. Here are some helpful hints for locating/defining scales:

- Divide the measured quality into its elementary components first if possible
- Use known, accepted scales of measure when possible
- Derive new scales from known scales by substituting terms
- Incorporate qualifiers in the scales to increase usefulness and specificity
- Don't confuse scale with meter
- Share effective scales with others

Examples of scales for several situations are given in Table 3:

Table 3: SCALE Examples

Environmental Noise	dBA at 1.0 meter
Software Security	Time required to break into the system
Software Maintainability	Average engineering time from report to closure of defects reported prior to release

System Reliability #1	The Mean Time To Failure of the system
System Reliability #2	The time at which a certain percentage of the system failures have occurred (known as the B-life). For example, at the B10 life, 10% of the units have failed.
System Learnability	Average time for <novices> to become <proficient> at a defined set of tasks (this can be measured on competing prototypes)
Vendor Of Choice	Gaps between customer's expressed importance and satisfaction for various product and service attributes
Revenue	Total sales in US$, Average Selling Price, etc.
Market Share	Percentage of Total Available Market (TAM)

To locate a meter, study the scale carefully. If no meter comes to mind:

- Look at references, handbooks, examples, etc. for ideas
- Ask others for their experience with similar methods
- Look for examples within test procedures

Once you have located a candidate meter, be sure that:

- The meter is adequate in the eyes of all stakeholders
- There is no less-costly meter available that can do the same job (or better)
- The meter can be measured *before* product release or completion of the deliverable

Examples of Meters for several situations are given in Table 4:

Table 4: METER Examples

Environmental Noise	Lab measurements performed according to the Environmental Test Handbook
Software Security	An attempt by a team of experts to break into the system using commonly available tools
Software Maintainability	Analysis of at least 30 consecutive defects reported and corrected during development
System Reliability #1	A Probability Ratio Sequential Test demonstration with α=10%, β=10%, Discrimination Ratio = 3
System Reliability #2	Weibull analysis of 50 sample units bench tested to failure

Planguage Examples

In practice, the TAG keyword is often dropped, as is the GIST keyword. Instead, the tag itself is placed before the text of the gist, like this:

LEARNABLE: The ease of learning to use the system

instead of

TAG: Learnable
GIST: The ease of learning to use the system

Most of the examples that follow use the shorter format combining the tag and gist.

Example 1: Power Consumption
Before Planguage, here is an actual requirement as written. Only the company names have been altered:

"The third key requirement is power consumption. Generally, the power consumption requirements are driven by noise requirements, or CE compatibility. The customers expressed the need for lower active power consumption so that passive cooling can be used. However, this is one possible implementation, and other implementations need to be addressed by engineering. Standby power consumption should meet the levels obtained by CE devices; 5-10W, and be achievable with the fan off. Cost is a factor. 10W standby is acceptable if the implementation cost is less than that of 5W standby. These requirements were articulated by *Company1*, *Company2*, *Company3*, *Company4*, and *Company5*."

The same requirement written using Planguage:

STANDBY: Standby Power Consumption ←{Company1, Company2, Company3, Company4, Company5}
GIST: The amount of power consumed by the system with the fan off and the HDD not spinning
SCALE: Watts
METER: Measurement on 3 units for 10 seconds at 23°C, ± 2°C
MUST: 10W
PLAN[CostOK]: 5W
CostOK: Design and manufacturing costs do not exceed 10W cost by more than 25%
NOTE: Relates to noise and CE compatibility requirements. Passive cooling within the system is desired.

This rewritten statement is traceable (since it is uniquely and persistently identified by its TAG), measurable (and testable), and more precise than the original while taking up less space and using fewer words than before.

Example 2: Acoustic Noise

Another actual requirement, as originally written:

"The second key requirement is that the acoustic noise generated by the PC be at levels similar to common consumer electronics equipment. Based on OEM feedback, this acoustic noise level while the PC is active (HDD active) needs to be in the range of 25-33dB. *Company1* shared the progress they have made in this area. They have moved from 38dB active in 1996 to 33dB active in 1997. Their goal is to maintain less than 33dB. *Company2' s*requirement is 25dB during active state."
Rewritten using Planguage:

NOISE: Acoustic Noise ←{Company1, Company2}
GIST: The amount of acoustic noise generated by the system with the fans running and HDD spinning.
SCALE: dBA
METER: Acoustic Sound Pressure test from the current Environmental Test Handbook, measured on 3 units
MUST [Company1]: 33dBA
MUST [Company2]: 25dBA
PLAN: 25dBA
TREND [1996 – 1997, Company2]: 38dBA → 33dBA

Note that other solutions are possible. The original requirement does not make clear whether the PLAN should be 33dB, 25dB, or some other value. Similarly, the MUST statement(s) could be written in several other ways. It is the conversations required to determine which expression is correct that are valuable.

Example 3: Development Process Efficiency

This example makes use of the optional sub-keywords for the Meter (Method, Frequency, Source, and Report).

EFFICIENT: The efficiency of the development process
SCALE: Rework as a percentage of total effort expended
METER: Examination of defect logs and project data
METHOD: Total Rework (defect logs) divided by total effort (project tracking database)
FREQUENCY: Measured monthly
SOURCE: Software Process Engineering Team data
REPORT: Senior Staff Meeting
MUST: No more than 45%
PLAN: No more than 35%
PAST: 50-60% ← guess, based on industry averages.

Example 4: Software Scalability

Scalability.CPU: The CPU usage pattern under increasing application stress.
SCALE: Minimum application transactions per second required to sustain 100% CPU utilization for at least 15 seconds.
METER: Stress testing of the application using automated software drivers and a representative operational profile.
MUST [Single Processor, 500MHz]: At least 45 TPS
PLAN [Single Processor, 500 MHz]: At least 60 TPS

Example 5: Security

This example illustrates how fuzzy concepts can be marked as needing clearer definition. The requirement could be used as a template for several projects, with the terms and achievement levels defined as needed for each one:

Security.Access: The resistance of the system to <unauthorized access>.
SCALE: Time required to obtain <unauthorized access> to the system using commonly available tools and techniques.
METER: Attempted <access> by a team of two skilled security engineers with no special knowledge of the system.
PLAN: At least 16 hours
MUST: At least 8 hours

Example **6: Memory Use**

TAG: MemoryUse
GIST: The amount of memory used by the application.
SCALE: Megabytes
METER: Performance Log observations made during system testing.
PLAN [Peak committed memory, Representative Operational Profile]: No more than 24 MB
PLAN [Peak committed memory, Stress Profile]: No more than 40 MB
PLAN [Average committed memory, Representative Operational Profile]: No more than 16 MB
PLAN [Average committed memory, Stress Profile]: No more than 24 MB
Representative Operational Profile: DEFINED: An operational profile that is likely to occur during use of the system after deployment. Specifically not a profile designed to stress the application in ways not possible or rarely encountered in actual use.
Stress Profile: DEFINED: An operational profile designed to cause extreme resource consumption or challenge the system's performance, regardless of whether the profile is likely or even possible to occur in actual use.

Lessons Learned Introducing Planguage at Intel

Planguage has been among the most popular topics in the requirements engineering coursework taught at Intel. The material has been presented to a broad cross section of the company, in terms of both job function and geographic location. Students embrace Planguage because it solves a real problem with elegance and simplicity. Most teams have felt the pain of mismatched

expectations that stemmed from weak, qualitative terms. Planguage presents an opportunity to avoid those problems from the start. Test teams and quality assurance personnel also like the clarity and accountability that comes with Planguage requirements.

If students have any difficulty as they learn Planguage, it is usually when they first attempt to locate scales and meters for Planguage statements. Students sometimes confuse scale and meter, so a simple example such as natural gas service or residential water supply is useful and provides a way to clarify thinking for less-obvious situations.

Although Planguage is a simple concept that has innate appeal, students typically require some additional assistance before they become independently proficient with the techniques involved (especially scales and meters). Two strategies work well to provide this assistance: follow-on mentoring from experienced Planguage users and a catalog of example Planguage requirements from which to draw ideas and templates. This catalog can be extended with new material as it is developed, and could be nicely implemented as a Website.

Planguage is designed for a much broader application than just quality requirements. Once Planguage use has been established on a team or in a business unit, others pick the language up for roadmaps, marketing objectives, vision statements, plans, and other uses. The positive benefits of such cross-pollination are significant.

References

Gilb01	Gilb, Tom, *A Handbook for Systems & Software Engineering Management using Planguage*, Addison Wesley 2001
Gilb97a	Gilb, Tom, *Requirements-Driven Management: A Planning Language*, Crosstalk, June 1997
Gilb97b	Gilb, Tom, *Quantifying the Qualitative*, available at http://www.result-planning.com

QW2001 Paper 9T1

Mr. Greg Berger
(Lawson Software)

Creating A Tool-Independent Test Environment

Key Points

- Discussion of some of the problems and concerns of developing tests using a specific automated test tool.
- How Object Oriented Programming Techniques can minimize the impact of tool specific changes.
- How integrating multiple technologies can create a test environment that makes use of the strengths of 3rd Party automated test tools, yet keeps your test environment and regression tests independent of the test tool.

Presentation Abstract

Our company has gone through many growing pains, starting from manual testing to automating much of the testing. In our situation, we had built up a regression base of client/server tests using a 3rd party test tool when our focus expanded to web testing. We now had to add another test tool for the web portion of our testing, while continuing to support our client/server testing. In order to accomplish this, we developed a testing environment that is tool independent.

With this idea in mind, we integrated many types of technologies and used as much of the Object-oriented programming techniques as possible. At present, we have common code that will run using Rational Robot and Mercury Interactive's WinRunner. We have implemented a VB ActiveX library and a C DLL for this purpose. We have also written our scripts/code in such a way that we have been able to isolate test tool specific calls (interface layer). We have written much of our functionality using objects and encapsulation.

Our final goal was to create a test environment that could use common code and functionality across several test tools and not be tied to a specific test tool. This has allowed us to use the strengths of each test tool and not have to re-develop those functionalities. The end result should give us flexibility to change, allow us to use our regression base in many other ways and cut our testing costs.

The talk discusses

* Present ways companies use automated test tools and how their test regression bases are being built up
* Problems and concerns that need to be addressed with developing tests using the current methods
* Solutions to minimize the impact of changes to the test tool or applications under

test
* Solutions to make your test environment more independent of a specific test tool
* Advantages to using the new test development methods

About the Author

Greg Berger is a Senior Systems Quality Engineer at Lawson Software (www.lawson.com) with responsibility for functional testing and test architectural design/implementation. Greg has been in the software test area for seventeen years, the last five of which using and developing in a wide variety of automated test tools, including Autotester, Mercury Interactive's WinRunner and Rational Robot.

Before Lawson, Greg worked in a variety of areas and technologies at companies such as Unisys and Pitney Bowes. Greg can be reached at greg.berger@lawson.com (651) 767-4061.

Creating a Tool-Independent Test Environment

Greg Berger
Lawson Software

14th International Software/Internet Quality Week (QW2001)

Overview

- Standard automation test methods
- Problems encountered
- Solutions

Standard Automation Test Methods

- Automation Decision
- Test tool selection
- Create Test bed
  - Record & Playback
  - Code development/scripting
  - Combination of R&P and Code Development
  - Problems & Concerns

Record & Playback

- Advantages
  - Quick & easy
- Problems & Concerns
  - Platform/OS dependency
  - No dynamic window captioning
  - No support for stress testing
  - No code re-use
  - High Maintenance

Code/Script Development

- Advantages
  - Code re-use
  - Less Maintenance
  - Robust
- Problems & Concerns
  - Changes within the test tool
  - AUT changes

Combination of Record & Playback and Code Development

- Advantages
  - Less expensive than total code development
- Problems & Concerns
  - Changes within the test tool
  - AUT changes

Changes in the Test Tool

- Problem:
  - Syntax changes
- Solution:
  - Interface functions
    - Test tool specific calls
    - Protects against syntax changes

Interface Functions

Script Without Using Interface Functions

```
Sub Main
   Dim result As Integer

   SQALogMessage sqaNone, "Test Message", "Sub
      Main"
   Window SetContext, GV_captiontitle, ""

   ' Clear out the window
   InputKeys "clear"

   ' Bring up the application screen
   InputKeys "execute Form100 {ENTER}"
End Sub
```

Script Using Interface Functions

```
Sub Main
   Dim result As Integer

   PRT_Print (PRT_MESSAGE, "Test Message", _
            "Sub Main")
   Window_SetContext ( CAPTION, GV_captiontitle, "")

   ' Clear out the window
   CMD_Execute ( CMD_CLEARSCREEN)

   ' Bring up the application screen
   CMD_Execute ( CMD_GENERAL, "execute Form100")
   CMD_Execute ( CMD_ENTER)
End Sub
```

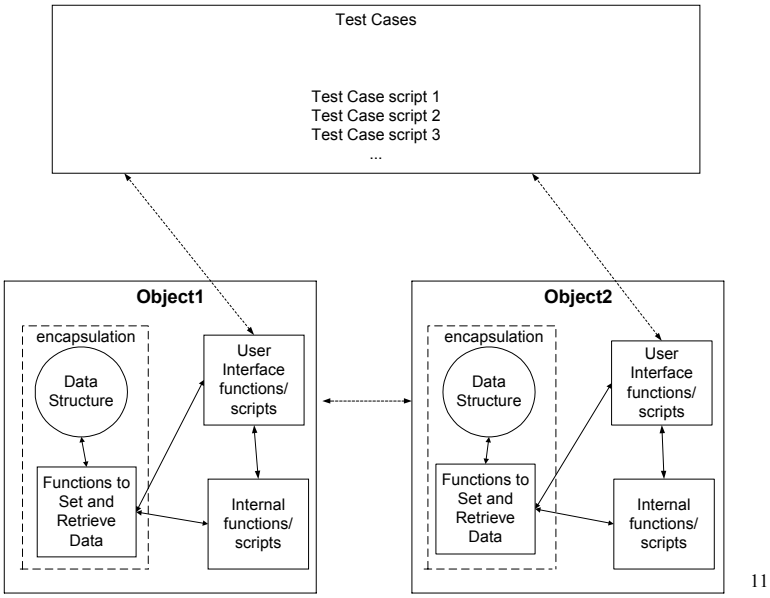
Changes by the AUT

- Problems:
  - User Interfaces
  - Functionality
- Solution:
  - Object Oriented Design (OOD) approach

Object Oriented Design

- Most tools do not support OOD
- Emulate some OOD features:
  - User Interface functions
  - Private functions
  - Assessor functions
  - No global variables

Object Oriented Design



11

Lawson Software's Automation Process

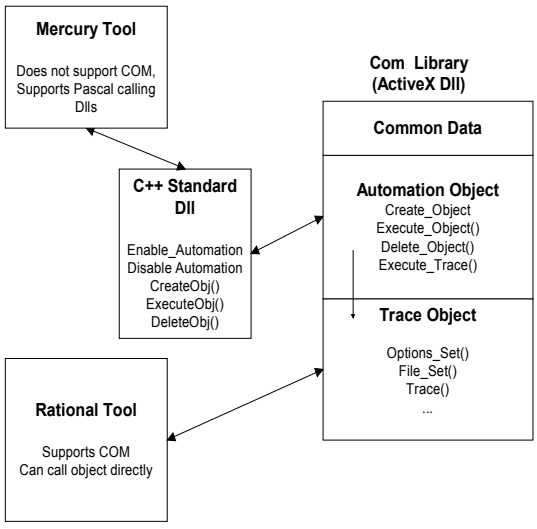
- Automation Decision
- Test tool selection
- *Test bed*
  - *Interface functions*
  - *OOD*
- **Second test tool needed**
  - Problems & concerns

12

Adding Test Tools

- Problems and Concerns:
  - Expense of creating a new test bed
  - Maintenance of two test beds
  - Addition of future test tools
- Solution:
  - Component Object Model (COM)

Lawson's **COM** Implementation



Disadvantages to COM Approach

- Requires up-front investment
  - Additional development tools
  - More training
- Initial Test development
- More code development expertise

Advantages to COM Approach

- Changes in technology and industry trends
- Code re-use
- Less maintenance
- Cost effective

Lawson Software's Automation Process

- Automation Decision
- ***Test tools selection***
- *Test bed*
  - *Test tool specific code base*
  - *COM objects*

17

Conclusions

- Automated with standard methods
- Encountered problems
- Developed a tool-independent approach
- Benefits
  - Better code re-use
  - Less Maintenance
  - Less Cost

18

Questions & Answers

Creating a Tool-independent Test Environment

Greg Berger
Senior Systems Quality Engineer
Lawson Software

greg.berger@lawson.com
http://www.lawson.com
(651) 767-4061

International Quality Week, 2001

Abstract

Lawson Software has experienced many growing pains, starting from manual testing to automating much of the testing. In our situation, we had built up a regression base of client/server tests using a 3rd party test tool when our focus expanded to web testing. Because the current tool did not have some functionality we needed, we added a second test tool for the web portion of our testing, while continuing to support our client/server testing. In order to accomplish this, we developed a testing environment that is tool independent.

With this idea in mind, we integrated many types of technologies and used as many of the Object-oriented programming techniques as possible. At present, we have common code that will run using Rational Robot and Mercury Interactive's WinRunner, implemented a VB ActiveX library and a C DLL for this purpose. We have also written our scripts/code in such a way that we have been able to isolate test tool specific calls (interface layer). We have written much of our functionality using objects and encapsulation.

Our final goal was to create a test environment that could use common code and functionality across several test tools and not be tied to a specific test tool. This has allowed us to use the strengths of each test tool and not have to re-develop those functionalities. The end result should give us flexibility to change, allow us to use our regression base in many other ways and cut our testing costs. This paper reflects the author's perspective gained from experience in the testing field and from helping to develop the existing test architecture at Lawson Software.

Overview of Automated Test Tool Usage

In today's market, companies have a wide range of automated test tools to choose from. Each tool has its strengths and weaknesses. For the sake of this discussion, an application that is tested by an automated test tool will be called Application Under Test (AUT).

Companies also have the option of writing their own automated test tools. This can lead to high development and maintenance costs.

The initial step a company will typically take when it wants to start automating its testing is to evaluate and select an automated test tool. Once a tool has been selected that best fits its needs, a regression test bed can then start to be built up.

A test bed is usually made up of the following:

- Test Cases
  A test case is a process that is performed against the AUT and the results of the process are checked and reported.
- Verification points
  A verification point is a checkpoint in a test case that results in a pass or fail.
- Supporting functionality such as windowing, AUT functionality and control flow.

These tests and supporting functionality are coded in scripts. A script is be made up of functions, sub-procedures and calls to other scripts.

Common methods of building a test bed are:

- Record and Playback
- Code/Script Development
- Combination of Record and Playback and Code/Script Development

Method: Record and Playback

Usage

Record and playback simply records an operator's keystrokes and actions as they are using the AUT. These scripts can then be played back. This method can produce scripts fast and with little programming expertise needed.

Problems and Concerns

- Does not support multiple hardware configurations/operating systems
  When multiple servers and operating systems are tested (such as Unix and NT), there can be differences in execution, such as system commands and responses. This creates a situation where scripts have to be recorded for each operating system.
- Does not support dynamic window captioning.
  Dynamic captioning is when the AUT window's caption can change during execution depending on circumstances and data input. For example, the caption for a Microsoft Word window contains the file name. If two files are open, the caption name depends on which file is active at the time. Because most test tools use the window caption attribute for their windowing, record and playback will not work.
- Does not support stress testing
  To perform stress testing, actions and functionality are executed multiple times through the use of looping. Record and playback does not support looping.
- No code re-use

All scripting is done as straight line coding. If there are actions that are done multiple times, they are recorded in multiple places. Record and playback does not promote code re-use.

- High maintenance

  If changes are made to the application, scripts will need to be updated or recorded again. The changes can affect many or all of the scripts depending on how many places the code reside in.

Method: Code/Script Development

Usage

Coding scripts, instead of recording them, can stabilize the regression bed. Code can be written as straight-line, or can be coded with modularity (functions and sub-procedures). Code containing modularity takes advantage of code re-use.

Problems and Concerns with Straight Line Code

- Does not support multiple hardware configurations/operating systems
- Does not support dynamic window captioning
- Does not support stress testing
- No code re-use
- High maintenance

Problems and Concerns with Scripts Coded with Modularity

- Test tool changes such as syntax changes, added functionality, or the elimination of functionality necessitates changes in the test scripts and functions.
- If an additional test tool is needed along with your current test tool, another test bed must be built and maintained with no re-usable code between the two.
- If the current test tool is no longer adequate for your testing, and another test tool replaces it, your entire test bed must then be rebuilt.

Method: Combination of Record and Playback and Code/Script

Usage

A mixture of recording and coding can be used. In this method, recording is done initially to create a template or shell of basic logic flow. Code is then added to the recorded scripts making them more robust.

Problems and Concerns

- Test tool changes such as syntax changes, added functionality, or the elimination of functionality necessitate changes in the test scripts and functions.
- If an additional test tool is needed along with your current test tool, another test bed must be built and maintained with no re-usable code between the two.
- If the current test tool is no longer adequate for your testing, and another test tool replaces it, your entire test bed must then be rebuilt.

Minimizing the impact of changes to the test tool and A

Changes in your automated test tool or AUT can result in high maintenance costs and downtime of your regression test bed. The following two sections discuss methods to reduce these obstacles.

Minimizing the Impact of Changes to the Automated Test Tool

The initial step is to minimize the impact of any test tool changes. This can be accomplished by creating interface functions that perform all test tool specific commands. All other scripts would then call those interface functions instead of using the tool's commands directly.

The following example is given using the Rational Robot test tool.

Script Without Using Interface Functions

```
Sub Main
   Dim result As Integer

   ' Make the host window the active window
   Window SetContext, GV_captiontitle, ""
   Window SetPosition, "", "Coords=" +
                       GV_termcoords +
                       ";Status=NORMAL"

   ' Clear out the Unix window
   InputKeys "clear"

   ' Bring up the application screen
   DelayFor 2000
   InputKeys "execute Form100 {ENTER}"
End Sub
```

Script Using Interface Functions

```
Sub Main
   Dim result As Integer

   ' Make the host window the active window
   WM_Window_SetContext(WM_Window,
                        GV_captiontitle)
   WM_Window_SetPosition(WM_Window
                GV_captiontitle,
                GV_termcoords)

   ' Clear out the Unix window
   CMD_Execute(CMD_CLEARSCREEN)

   ' Bring up the application screen
   Call WM_Window_Delay(2000)
   CMD_Execute(CMD_GENERAL,"execute ")
   CMD_Execute(CMD_GENERAL,
                "Form100")
   CMD_Execute(CMD_ENTER)
End Sub
```

In this example, the functions WM_Window_SetContext, WM_Window_SetPosition, WM_Window_Delay and CMD_Execute are the interface functions. By structuring your scripts this way, you have now protected a majority of those scripts from any test tool changes that might occur.

Minimizing Changes to Your AUT

Scripts written using object-oriented methods can help minimize the impact of changes made to your AUT. Even though most automated test tools do not support object-oriented

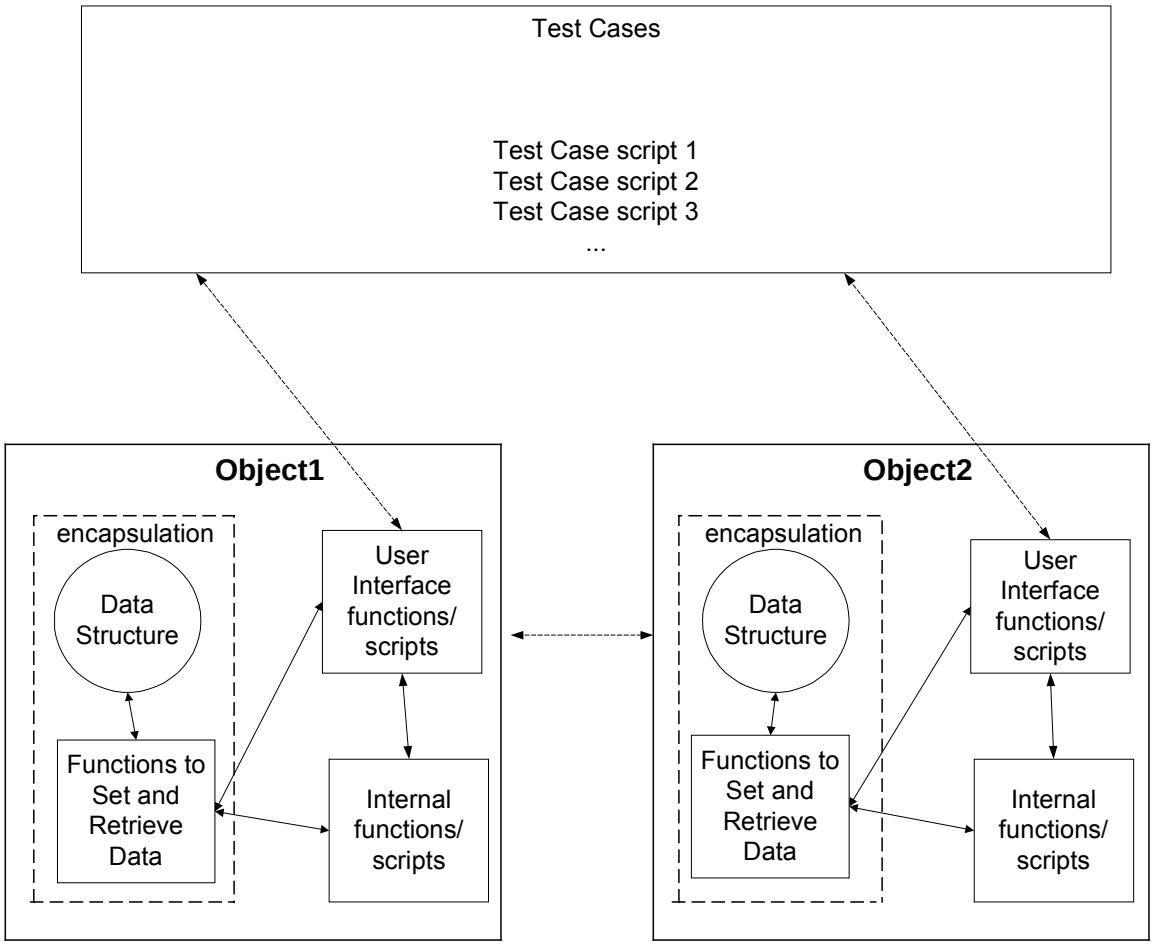
design (OOD); you can still emulate some of the OOD features. High cohesion should be the goal to strive for in OOD.

High cohesion has the following attributes:

- A well defined set of related functions that are exposed to the user (Public functions)
- A set of functions that implement the functionality of the object that are not exposed to the user (private functions)
- A set of assessor functions used to access the data of the object (the data is private)
- No global variables

All of the attributes listed above are used to create an object. The object contains scripts and functions of common functionality. All of the data needed for this object and its implementation of that data is hidden from all external scripts (encapsulation). External scripts access the object's data through a function (assessor). Functions and scripts within the object would also use those functions to access its data. This allows the object's data structures to be changed or added to without affecting any of the functions or scripts accessing that data. By encapsulating the data, the module also has control on what data can be modified and viewed. This is in contrast to using global variables to hold your data where there is no control.

Here is a diagram of what an OO design might look like:



Here is a sample of scripts and functions using the OOD approach with Rational Robot used as an example:

Master Shell to control test execution (script)

```
Option Explicit
'$Include "SetupD.sbh"
'$Include "WindowD.sbh"
'$Include "PrintD.sbh"

Sub Main
    Dim iReturn As Integer
    Dim iWindowID As Integer
    Dim iCurrentSession As Integer
    Dim sServer As String

    Call Setup          ' Get test parameters from user and open up terminal emulation window

    sServer = TTY_Session_GetServer()          ' Retrieve server that testing is being done on
    CALL PRT_PRINT(PRT_MESSAGE, "** START testing on Server: " & sServer,  _
```

```
                        "Sub Main")
   iWindowID = WM_Window_GetID()              ' Get the current window that has focus
   iCurentSession = WM_Session_GetCurrent()   ' Get the current emulation session

   iReturn = TTY_Session_Connect(iWindowID, iCurrentSession)
   Callscript "Test_case_1"                                    ' Run Test Case 1
   Callscript "Test_case_2"                                    ' Run Test Case 2
   iReturn = TTY_Session_Disconnect(iWindowID)    ' Disconnect emulation session
End Sub
```

Test Case Scripts

```
' Test Case 1
Option Explicit
'$Include "WindowD.sbh"
'$Include "PrintD.sbh"
'$Include "CommandD.sbh"

Sub Main
Dim iReturn As Integer

    CALL PRT_Print(PRT_MESSAGE, "** START Test Case 1 " , "Sub Test Case 1")
     iWindowID = WM_Window_GetID()      ' Get the current window that has focus
    CMD_Execute(CMD_CLEARSCREEN)
    CMD_Execute(CMD_GENERAL,"cd TestDir")
    VP = WM_Window_Check("File YYY")            ' Perform Verification Point check
    CMD_Execute(CMD_OSTRANSMIT)
End Sub
```

```
' Test Case 2
Option Explicit
'$Include "WindowD.sbh"
'$Include "PrintD.sbh"
'$Include "CommandD.sbh"

Sub Main
Dim iReturn As Integer

    CALL PRT_ Print (PRT_MESSAGE, "** START Test Case 2 ", "Sub Test Case 2")
     iWindowID = WM_Window_GetID()      ' Get the current window that has focus
    CMD_Execute(CMD_CLEARSCREEN)
    CMD_Execute(CMD_GENERAL,"ls TestDir")
    VP = WM_Window_Check("File XXX")            ' Perform Verification Point check
    CMD_Execute(CMD_OSTRANSMIT)
End Sub
```

Objects

```
'**** PRINT OBJECT - QUICK REFERENCE ************
'
'**** ASSESSORS
' PRT_Get
' PRT_Set

'**** METHODS
' PRT_Print
        …
'**** PRIVATE METHODS
' PRT_RobotLog_Write
        …
'*********************************************

'$Include "Print.sbh"
Option Explicit

'*********************************************
'***** Data Definitions for the Print Object      *****
'*********************************************
Type m_PrintData
      iState            As Integer            'State of Printing
      sFileName         As String             'File name Parameter Setting
        …
End Type

Global m_aTableSettings(PRT_LOWERDIM To PRT_UPPERDIM) As m_PrintData
'*********************************************
'***** Assessors for this Object                  *****
'*********************************************

'******************** Function PRT_Get  ***********
Function PRT_Get(iDevice As Integer, _
                      sAttribute As String) As Variant

   Select Case sAttribute
        Case PRT_STATE
             PRT_Get = m_aTableSettings(iDevice).iState
        Case m_FILENAME
             PRT_Get = m_aTableSettings(iDevice).sFilename
   End Select
End Function

'******************** Sub PRT_Set  ***************
Sub PRT_Set(iDevice As Integer, _
                sAttribute As String, _
                 vValue As Variant)
```

```
   Select Case sAttribute
         Case PRT_STATE
               m_aTableSettings(iDevice).iState = vValue
         Case m_FILENAME
               m_aTableSettings(iDevice).sFilename = vValue
   End Select
End Sub

'***********************************************
'***** Methods for Print Object                 *****
'***********************************************

'******************** Function PRT_Print  **********
' *** User Interface to the Print Object ***
Sub PRT_Print(sType As String, _
                  sMessage As String, _
                  sFunctionName As String)
   Dim iReturn As Integer
   Dim iTraceFile As Integer

   iTraceFile = PRT_GET (PRT_TRACEFILE, PRT_STATE)
   If (iTraceFile = PRT_ON) Then
      iReturn = PRT_File_Message(PRT_TRACEFILE, sType, sMessage,
                                 sFunctionName)
   End If

    iReturn = PRT_RobotLog_Write(sType, sMessage, sFunctionName)
End Sub
```

Test Tool Interface Functions

```
'***** FUNCTION PRT_RobotLog_Write ********
Function PRT_RobotLog_Write(sType As String, _
                                 sMessage As String, _
                                 sFunctionName As String) as Integer
   Dim iReturn As Integer

   iReturn = PRT_PASS

   Select Case sType
      Case "PASS"
         SqaLogMessage SqaPass, sMessage, sFunctionName
      Case "FAIL"
         SqaLogMessage SqaFail, sMessage, sFunctionName
      Case "WARNING"
         SqaLogMessage SqaWarning, sMessage, sFunctionName
```

```
        Case "MESSAGE"
            SqaLogMessage SqaMessage, sMessage, sFunctionName
    End Select

    PRT_RobotLog_Write = iReturn
End Function
```

Making Your Test Environment Independent of a Test

Developing your scripts and functions using Object Oriented methods better protects your test bed from test tool and AUT changes. This also does a good job of utilizing code-reuse.

However, what happens when your company now shifts paradigms and goes to web applications instead of client server applications, or adds web applications to the client server applications already being used and tested? The test bed that was built up for the client server testing can still be used for that testing. The scripts and functions developed for that test bed would most likely not work on the web applications without some modifications. If the scripts and functions were designed properly, the only scripts and functions that may need updating are the test tool interface and lower level functions. In some cases, two test beds will be needed and maintained, one for client server and one for web applications.

A more severe problem occurs when the current test tool you are using cannot be used against the new applications because the tool does not support some functionality that is needed. In this situation, a new test tool will be needed, causing a new test bed to be created. If you are still supporting the existing applications, then you will have to maintain both test beds.

In our situation, we were using Rational Robot for our client-server testing. Lawson Software then began developing web applications in addition to the client-side applications. We found that Robot did not support some functionality that we needed for the Netscape web-browser at that time, therefore, we added the Mercury's WinRunner tool that did support that functionality.

Both a change in testing and a change in tools result in modifications of scripts and functions or replacement of those scripts and functions. There is also the task of maintaining multiple sets of code. This can become a very large and expensive process, especially if the entire test bed must be rebuilt. It would be nice if all the common functionality between the two technologies could be shared, keeping the code base minimized and the maintenance low. The good news is that they can.

The use of mixed development environments and languages can be used to accomplish code re-use between automated test tools or for the replacement of test tools. Our test group at Lawson Software chose the Component Object Model (COM) as our framework to create common code objects. COM is based on components that can be instantiated from any programming language (Visual Basic, Visual C++, etc.). We chose Visual Basic as our main programming language. Common code functions were written as an ActiveX DLL using VB. ActiveX components allow an application to use objects supplied by another application, or

expose its own objects for use by another application. An ActiveX DLL is an in-process component that runs in another application's process.

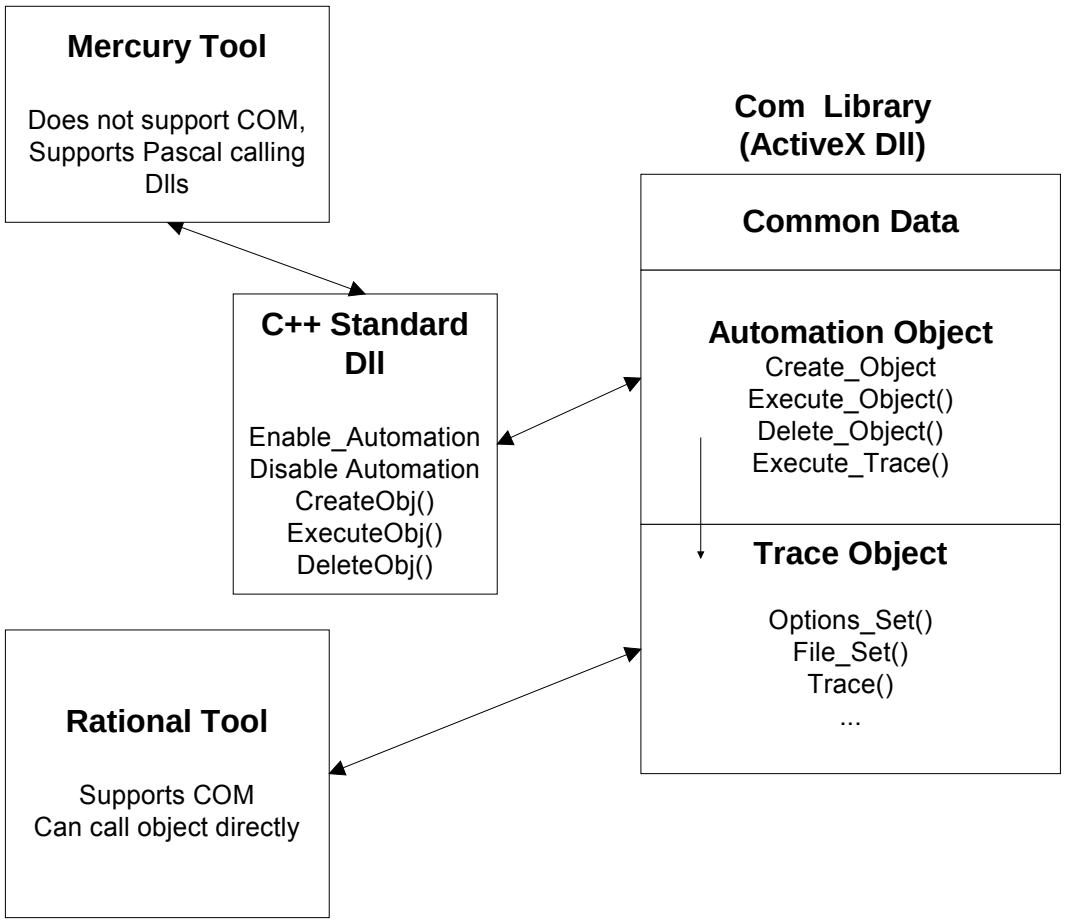
We initially moved the common functionality from Rational Robot to the ActiveX DLL. Rational Robot supports COM calls therefore the test runs using the ActiveX DLL ran without problems. We now wanted to take the ActiveX DLL and make calls into the WinRunner test scripts. This would give us the code re-use we were looking for.

We found that WinRunner does not support COM. Therefore, we could not create the ActiveX Object and run it directly from WinRunner. WinRunner was able to call standard 'C' DLLs however, therefore we wrote a standard 'C' DLL using VC++ which contained the functions necessary to access and manipulate the ActiveX Objects. By using this indirect approach, we were able to run successfully with WinRunner.

We now had an environment where we could share functionality between the two test tools. In addition, we are currently writing our startup routine (input test parameters, runtime tool parameters, etc.) as a VB application that will run with either Robot or WinRunner.

Our goal is to produce as much of a common code base as possible for AUT specific functionality and support functionality. This in turn helps keep maintenance low and re-use high. We still want to take advantage of the strengths of each test tool however, whether it is managing the running of tests, logging of test results or verifying object properties. This gives us a distinct division of code between AUT and test tool functionality.

Here is a diagram to help illustrate this approach:



Here is a sample of scripts and functions from Rational Robot and Mercury using the COM approaches. In this example, the common code is tracing functionality used for debugging. Note that these functions are incomplete and are used for demonstration purposes only.

Mercury Test Script

```
# *******  Initialize test  **************
load_dll ("c:\Winnt\system32\Comlib.dll");
load("trace");

extern int Enable_Automation();
extern int Disable_Automation();

iReturn = Enable_Automation();
TRC_SetTraceObject();
#*********************************
TRC_File_Set(TRC_TRACEFILE,
                    TRC_FILENAME,
                    "c:MercuryTest.txt");
TRC_OptionsSet(TRC_TRACEFILE,
                     TRC_STATE,
                     TRC_ON);

# *******  Main section of testing *******
TRC_Trace(TRC_DEBUG_ LVL1,
                MSG_ENTERREC,
```

Rational Robot Test Script

```
'$Include "TraceD.sbh"

Sub Main

'*******  Initialize test  ******************
   Call TRC_SetObjectActive()



'****************************************
   Call TRC_File_Set(TRC_TRACEFILE,
                        TRC_FILENAME,
                        "c:RationalTest.txt")
   Call TRC_OptionsSet(TRC_TRACEFILE,
                           TRC_STATE,
                           TRC_ON)

'*******  Main section of testing  **************
    Call TRC_Trace(TRC_DEBUG_ LVL1,
                      MSG_ENTERREC,
```

```
                "tracetest");

TRC_Trace(TRC_MESSAGE,
                "Test Log message #1",
                "tracetest");

TRC_Trace(TRC_DEBUG_ LVL1,
                MSG_EXITREC,
                "tracetest");
# ************************************
# *******  Clean-up test  ***************
TRC_File_Close(TRC_TRACEFILE);
TRC_DeleteTraceObject();

iReturn = Disable_Automation();

unload_dll ("c:\Winnt\system32\ Comlib.dll");
unload("trace");
# ************************************
```

```
                    "tracetest")

    Call TRC_Trace(TRC_MESSAGE,
                    "Test Log message #1",
                    "tracetest")

    Call TRC_Trace(TRC_DEBUG_ LVL1,
                    MSG_EXITREC,
                    "tracetest")
'****************************************
'*******  Clean-up test  ********************
   Call TRC_File_Close(TRC_TRACEFILE)
   Call TRC_DeleteTraceObject()
'****************************************

End Sub
```

Mercury Interface Functions

```
extern int CreateObj(string iString);
extern int ExecuteObj(int iObject,
                        string sMethod, string sParm1,
                        string sParm2, string sParm3,
                        string sParm4);
extern int DeleteObj(int iObject);

#***** Data Definitions for the Trace Object *****
public oTrace = -1;

#***** FUNCTION TRC_SetTraceObject  ******
public function TRC_SetTraceObject()
{
        oTrace = CreateObj("TRACE");
}

#***** FUNCTION TRC_DeleteTraceObject  ***
public function TRC_DeleteTraceObject()
{
        auto iReturn;
        iReturn = DeleteObj(oTrace);
}

#***** FUNCTION TRC_Trace  **************
public function TRC_Trace(sType,
                            sMessage,
                            sFunctionName)
{
   auto iReturn;

   iReturn = ExecuteObj(oTrace, "TraceItem", sType,
                        sMessage, sFunctionName,
                        "*WR");
   iReturn = TRC_MercuryLog_Write(sType,
                                    sMessage,
                                sFunctionName);
}
```

Rational Robot Interface Functions

```
'$Include "Trace.sbh"

Option Explicit



'***** Data Definitions for the Trace Object *****
Global oTrace As Object

'***** SUB TRC_SetObjectActive **************
Sub TRC_SetObjectActive()

   Set  oTrace = CreateObject("ComLib.cTrace")
End Sub

'***** FUNCTION TRC_DeleteTraceObject ****
Sub TRC_DeleteTraceObject()

   Set  oTrace = Nothing
End Sub


'***** FUNCTION TRC_Trace  ***************
Sub TRC_Print(sType As String, _
                sMessage As String, _
                sFunctionName As String)
   Dim iReturn As Integer

   Call oTrace.TraceItem(sType,
                        sMessage,
                        sFunctionName,
                        "*RR")
   iReturn = TRC_RobotLog_Write(sType,
                                sMessage,
                                sFunctionName)
End Sub
```

```
#***** FUNCTION TRC_MercuryLog_Write ***
static function TRC_MercuryLog_Write(sType,
                                     sMessage,
                                   sFunctionName)
{
   tl_step(sFunctionName, 0, sMessage);
}
```

```
'***** FUNCTION TRC_RobotLog_Write ********
Function TRC_RobotLog_Write(sType As String, _
                    sMessage As String, _
                    sFunctionName As String) as Integer

   SqaLogMessage SqaPass, sMessage, sFunctionName
End Function
```

C++ DLL Interface Between WinRunner and the ActiveX DLL

```
_cAutomation oAutomation;

/***** FUNCTION Enable_Automation ****/
int Enable_Automation()
{
   oAutomation.CreateDispatch(
                        "lawsonLib.cAutomation");
   return 1;
}

/***** FUNCTION Disable _Automation ***/
int Disable_Automation()
{
   oAutomation.DetachDispatch();
   return 1;
}

/***** FUNCTION CreateObj ***********/
int CreateObj(const char *sObject)
{
   short iReturn;
   BSTR bMode = NULL;
   wchar_t *bString;
   bString = (wchar_t *)malloc(strlen((
                const char *)sObject));

   mbstowcs(bString,sObject,strlen((
                const char *)sObject)+1);
   bMode = NewBSTR(bString);
   iReturn = oAutomation.Create_Object(
                        &bMode);
   SysFreeString(bMode);

   return (int) iReturn;
}

/***** FUNCTION ExecuteObj **********/
int ExecuteObj(int iObject, char *sMethod,
                char *sParm1, char *sParm2,
                char *sParm3, char *sParm4)
{
   int iReturn;
```

ActiveX DLL

Class cAutomation

```
Dim oObject As Object

'***** FUNCTION Create_Object ***********
Public Function Create_Object(sObject As String)
                                As Integer

   Set oObject = New cTrace
End Function

'***** FUNCTION Execute _Object *********
Public Function Execute_Object(iObject As Integer,
          sMethod As String,  sParm1 As String,
          sParm2 As String, sParm3 As String,
          sParm4 As String) As Integer
   Dim iReturn As Integer

   iReturn = Execute_Trace(iObject, sMethod,
                sParm1, sParm2, sParm3, sParm4)

   Execute_Object = iReturn
End Function

'***** FUNCTION Delete _Object ***********
Public Function Delete_Object(iObject As Integer)
                                As Integer

   Set oObject = Nothing
   Delete_Object = iObject
End Function

'***** FUNCTION Execute_Trace ***********
Private Function Execute_Trace(iObject As Integer,
            sMethod As String, sParm1 As String,
            sParm2 As String, sParm3 As String, _
            sParm4 As String) As Integer
   Dim iReturn As Integer
   Dim iDevice As Integer
   Dim iMode As Integer

   iReturn = 0
```

```
   short shObject;

   iReturn = oAutomation.Execute_Object(
                  &shObject, &bMethod,
                  &bParm1, &bParm2, &bParm3,
                  &bParm4);

   return (int) iReturn;
}

/***** FUNCTION DeleteObj **********/
int DeleteObj(int iObject)
{
   short iReturn;
   short shObject;

   shObject = (short) iObject;
   iReturn = oAutomation.Delete_Object(
                     &shObject);
   return (int) iReturn;
}
```

```
   Select Case UCase(sMethod)
      Case "OPTIONSSET"
         iReturn = iObject.OptionsSet(iDevice,
                                 sParm2, iMode)
       Case "TRACEITEM"
         Call iObject.TraceItem(sParm1, sParm2,
                                 sParm3, sParm4)
   End Select

   Execute_Trace = iReturn
End Function
```

Class cTrace

```
'***** FUNCTION OptionsSet
************************
Public Function OptionsSet(iDevice As Integer,
                           sAttributes As String,
                           iMode As Integer) As
Integer

         … Code Here
End Function

'***** FUNCTION TraceItem
*************************
Public Sub TraceItem(sType As String,
                     sMessage As String,
                     sFunctionName As String,
            Optional sPrefix As Variant)

         … Code Here
End Function

         …
```

Sample Output for Mercury

```
"*********************************************************************"
"            TEST RUN STATUS LOG"
"*********************************************************************"
" LOG CREATED:  9/13/00 9:42:15 AM"
" LOG LOCATION: i:MercuryTest.txt"
"*********************************************************************"
"9:42:15 AM: STATUS LOG OPENED"
"9:42:15 AM: tracetest          *WR*LEVEL1*      Enter Rec"
"9:42:15 AM: tracetest          *WR*MESSAGE*   Test Log message #1"
"9:42:15 AM: tracetest          *WR*LEVEL1*      Exit Rec"
"9:42:15 AM: STATUS LOG CLOSED"
```

Sample Output for Rational Robot

```
"*********************************************************************"
"            TEST RUN STATUS LOG"
"*********************************************************************"
" LOG CREATED:  9/13/00 8:42:15 AM"
" LOG LOCATION: i:RationalTest.txt"
"*********************************************************************"
"8:42:15 AM: STATUS LOG OPENED"
"8:42:15 AM: tracetest          *RR*LEVEL1*       Enter Rec"
"8:42:15 AM: tracetest          *RR*MESSAGE*    Test Log message #1"
"8:42:15 AM: tracetest          *RR*LEVEL1*       Exit Rec"
"8:42:15 AM: STATUS LOG CLOSED"
```

Conclusion

Under most circumstances record and playback scripts are the least desirable. These scripts are sensitive to application changes, test tool changes, multiple environments, dynamic captioning and many other factors. In most cases, these scripts will have to be maintained regularly or even replaced periodically. The only times I would recommend using this method would be for demos and one time only runs.

Scripting (coding) is more robust and allows for code re-use. For this to be effective however, the scripts should be written in an object-oriented fashion. Interface functions should also be used to isolate test tool specific calls.

If scripts are developed using these techniques, then they should be protected from most AUT changes. These scripts will require less maintenance and be more robust. Code re-use will also be higher for use with different hardware and operating system configurations. Under some circumstances, these scripts may also be used for different types of testing, such as client server vs. web testing. These scripts do not however go across test tools. If for some reason you are using two test tools, or you have to switch to another test tool, and you need the same functionality in both, then the scripts will have to be converted.

By developing your common test functionality using COM objects, ActiveX objects, etc., you can use the same code for all test tools. This method can be more expensive up front, and more technical and code development expertise is needed in your test development group than with using the other methods. However, the savings from code re-use and lower maintenance will pay off in the long run. You will also have more control over your test bed and will have more flexibility in selecting alternative test tools or other solutions if the situation should arise.

QW2001 Paper 9T2

Mr. Timothy Kelliher, Dr. Daniel Blezek, Mr. William Lorensen & Dr. James Miller
(GE Corporate R&D)

The Frost Extreme Testing Framework

Key Points

- The frost software testing framework
- Test result collection
- Test result storage
- Test reporting

Presentation Abstract

It has been our experience that small to medium software development efforts in a distributed environment do not have the resources to staff a full software quality assurance department. Indeed, having a SQA for a small or medium development team can significantly impede the development and release process. Software quality should not be ignored in these cases, rather it should be built in to the development process itself, effectively ensuring high quality software. To address this need, we have built a non-intrusive framework to collect, summarize, and trend software and system testing results. Our system, named Frost (Frequent Regular, On-Demand System Testing), allows the developer in the small project to easily deploy a software testing infrastructure in the initial phases of the project. Incorporating software testing using Frost in the early stages of development leads to increased project transparency, and quality.

About the Author

Timothy P. Kelliher is a Computer Scientist at GE's Corporate Research and Development Center in Schenectady, NY. He has over 15 years experience in systems and software engineering. At the center he has worked on Software Engineering CASE tools, Human Computer Interaction, and Software Quality systems. He is a Six Sigma Master Black Belt and spends much of his time instructing and mentoring the GE software development community. He is co-author of “Engineering Complex Systems with Models and Objects” published by McGraw-Hill, 1997.

Daniel Blezek is a Computer Scientist in the Visualization and Computer Vision Program at GE's Corporate Research and Development center. He holds a Ph.D. from the Mayo Graduate School in Biomedical Engineering. His research interests

include medical image segmentation, advanced rendering algorithms, and practical software quality techniques.

William Lorensen is a Graphics Engineer at GE's Corporate Research and Development Center in Schenectady, NY. He has over 30 years of experience in computer graphics and software engineering. William is currently working on algorithms for 3D medical graphics and scientific visualization. William is the author or co-author of over 60 technical articles on topics ranging from finite element pre and postprocessing, 3D medical imaging, computer animation and object-oriented design. He is a co-author of "Object-Oriented Modeling and Design" published by Prentice Hall, 1991. He is also co-author with Will Schroeder and Ken Martin of the book "The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics" published by Prentice Hall in November 1997. Mr. Lorensen holds twenty seven US Patents on medical and visualization algorithms.

James Miller is a Computer Scientist at GE's Corporate Research and Development Center in Schenectady, NY. He joined GE after receiving his PhD from Rensselaer Polytechnic Institute in 1997. His thesis topic was in Computer Vision. At GE James has become a primary contributor to vtk software algorithm development and testing. For the past year, he has been the project leader on a research project for Lockheed Martin involving the inspection of large airframes using laser ultrasonic techniques.

The Frost Extreme Testing Framework

Frequent, Regular, On Demand System Testing

Dan Blezek, Tim Kelliher,
Bill Lorensen, Jim Miller
GE CRD
1 Research Circle
Niskayuna, NY 12309

What is Frost?

- Software Testing Data Collection Framework
  - Data Collection
  - Archive
  - Presentation
- Goal and Objectives
  - Improve Productivity and Quality of Software Projects
    - Reduction in Rework
    - Early Error Detection
    - Reduction in Administration Time
  - Apply Six Sigma Principles
    - Quantitative Management
    - Requirements Validation

Extreme Testing

- "If testing is good, everybody will test all the time, even the customers" -- Kent Beck
- Qualities of an Extreme Testing Framework
  - Reusable
  - Distributed
  - Persistent
  - Structured
  - Executed Regularly
  - Independent of presentation
  - Dynamic

Frost: Extreme Testing Framework

- Frequent, Regular, On Demand System Testing

Two roads diverged in a wood, and I --
I took the one less traveled by,
And at has made all the difference.
-- Robert Frost

- Software testing is often the road less traveled, but it does make all the difference

Extreme Testing

- Short software engineering life cycle
  - Design, implement, test
- The Software should **ALWAYS** work
- Find and fix defects in hours not weeks
  - Bring SQA inside the development cycle
  - Break the cycle of letting users find bugs
- Automate everything
- All developers are responsible for testing

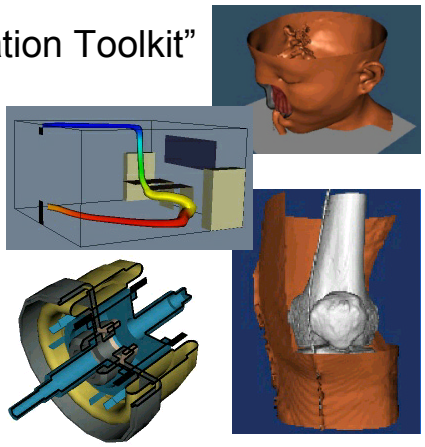
The Importance of Early Testing

- Testing early and often is critical to high quality software
- Retain measurements to assess progress and measure productivity
- Present results in concise, informative ways
- Know and Show the status of the system at any time

- Our customers expect it to be the way we work

Motivation (Historical Perspective)

- We develop the “Visualization Toolkit”
- Open Source
- 600 C++ classes
- 240,000 Lines of Code
  - 100,000 executable
- 20+ developers
- 6 years of development



“We don’t sell VTK, we sell what we do with VTK.”

Testing Constraints

- We only have 15 active developers, spread across many projects and sites
  - Can’t afford separate SQA division
- We don’t have dedicated testing hardware
  - Testing cannot hinder project work
- We are “algorithm developers” not “software engineers”
  - Testing must be automated and concise

Testing Design Goals

- Frequent testing
  - Identify defects as soon as they are introduced
  - Too hard to find cause if not done frequently
- Minimally invasive to daily activities
- Automated testing
- Automated report generation/summaries
  - Must be concise yet informative
- Track testing results over time

A day in the life of VTK quality

- The day starts at 8pm (EST)
- Determine what has changed in the system
- Update the testing system's version of the software
- On 11 different system configurations, we
  - Build the software
  - Run over 500 regression tests
- Dynamic memory analysis
- Coverage analysis
- Check on coding style and documentation

A Good Day

There were 32 files changed since the last dashboard. Changes for the month.

Platform	Build VTK		Test Tcl				Test Cxx	
	Errors	Warnings	Passed	Failed	Faster	Slower	Passed	Failed
irix65	0	22	416	0	3	5	49	0
irix6n32	0	5210	416	0	0	0	49	0
irix6n64	0	445	416	0	0	0	49	0
solaris	6	19815	394	2	0	13	44	0
solaris27	0	4064	416	0	8	3	49	0
solarisCoverage	0	679	412	2	0	5	49	0
WinNT	0	2	414	2	0	0		
hp	0	11729	413	3	1	2	48	1
linuxRH52	0	298	415	1	0	4	49	0
solaris26m6	0	4070	408	8	0	3	47	2
solaris26	0	0	402	1	0	0		

Key:	Internal Test System	External Test System	System or Network Error	Build/Test Successful	Build/Test Error	Timing Change

- Real work begins at 7:05am

7:00am - A Bad Day

Platform	Build VTK		Test Tcl				Test Cxx	
	Errors	Warnings	Passed	Failed	Faster	Slower	Passed	Failed
irix6	0	286	355	0	7	4	44	2
irix6n32	2	2419	355	0	1	0	46	0
solaris	0	16273	341	2	3	1	41	0
solarisCoverage	0	10	353	2	0	8	45	0
WinNT	Catastrophic failure.		Catastrophic failure.					
hp	0	4955	0	396	0	0	0	630
linux	0	10	Catastrophic failure.					
solaris26	0	0	310	9	0	0		

Key:	Internal Test System	External Test System	System or Network Error	Build/Test Successful	Build/Test Error	Timing Change

- We are “prisoners of quality”...

How do we use VTK SQA?

- Track the effects of major changes
- Identify what needs to be changed
- Portability leads to quality
- Navigate software features
- Build and test on future OS releases
- Test new 3rd party software (e.g. OpenGL) for compliance

Has truly “Changed the way we work.”

The road to Frost

- VTK SQA exceeded our expectations
- VTK SQA process generated a lot of interest
- VTK SQA is
  - “vtk-centric”
  - daily snapshot of quality
  - a collection of csh, tcl, awk, grep, sed scripts
  - hard to port to new software projects
  - hard to navigate through time
- Step back, re-load, … Frost

Critical To Quality

- Reusable
  - Test frameworks are often custom, and hard to transition
  - VTK testing is an example
  - Frost is adaptable to many different projects
  - Contains concepts that capture the essence of testing

Critical To Quality

- Distributed
  - Anyone should be able to contribute testing data from anywhere, on any device.
  - Test results should be available from anywhere
  - Frost allows collection through a variety of mechanisms
    - TCP/IP, HTTP, SMTP, FTP
  - Language independent XML format for reporting

Critical To Quality

- Persistent
  - Observations should be retained over the life of the project, and throughout the product life
  - Readily available for trending and analysis
  - Frost is built on relation database technologies
    - Provides persistence of observations
    - Query capabilities
    - Facilitates data import and export

Critical To Quality

- Structured
  - The collected data must be structured
  - Can immediately detect missing, or incomplete test data
  - Facilitates efficient summarization
  - The core concepts of Frost completely characterize the testing structure
    - Gauges collect Measurements during testing
    - Tests group Gauges into logical clusters
    - TestGroups organize Tests and other TestGroups into a hierarchy

Critical To Quality

- Executed Regularly
  - Frequent: encourages frequent code changes
  - Regular: collects overall view of incremental development activities
  - On Demand: enables each developer to make local changes with immediate evaluation
  - Frost supports all of these modes

Critical To Quality

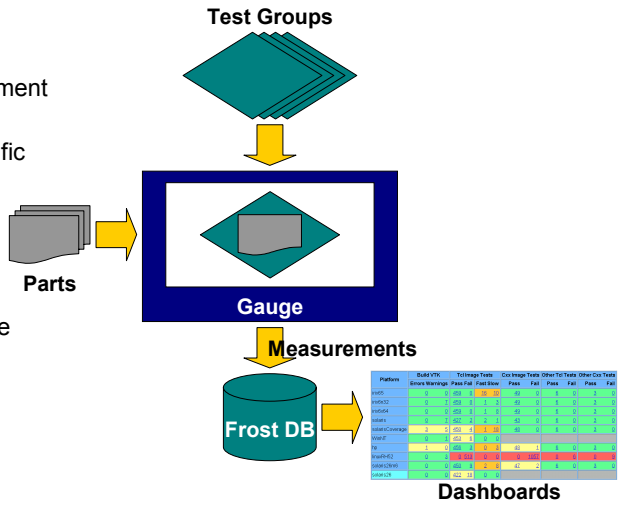
- Independent of Presentation
  - The usual case is one tool - one log
    - Need to look in multiple places
      - Build log
      - Purify
      - Coverage
      - Regression test log
  - Frost collects from each tool into a central location
  - Presents a comprehensive dashboard
  - “Quality-at-a-glance”

Critical To Quality

- Dynamic
  - Must always have up to date dashboard
  - If Extreme Testing is the goal, it must be supported with immediate feedback
  - Frost always presents most recent test results

Frost Concepts

- Gauge
  - Takes a measurement during a TestRun
  - May be Part specific
- Test
  - Organized in TestGroups
- Part
  - A component to be tested
  - Organized in a hierarchy



Gauge

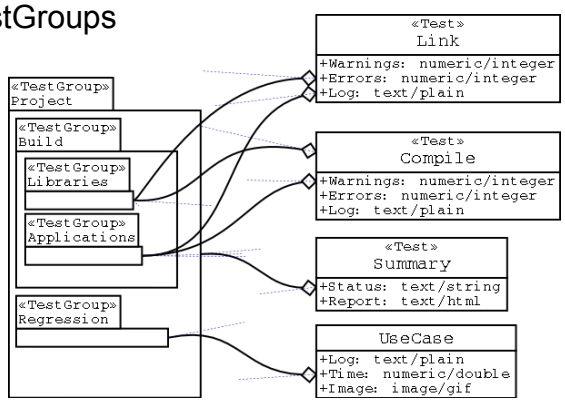
- Core component of Frost
  - Associated type
    - integer, double
    - text, HTML, XML, etc…
  - Records Measurements during TestRuns
  - Organized into Tests
  - Are named
    - i.e. TimeInSeconds of type double
    - BuildLog of type text

Test

- Collection of Gauges
- Executed to produce a TestRun
- Produce a pass/fail status
  - Frost automatically records a NotRun status
- Organized into TestGroups
  - May be in more that one TestGroup

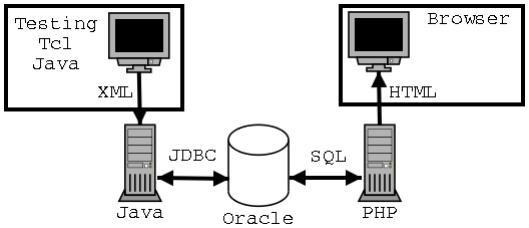
TestGroup

- Hierarchical
  - May contain Tests
  - May contain TestGroups
- Example



Data Flow

- Inbound Test Data
  - Testing
  - XML
  - Server
  - Oracle
- Queried results
  - Web server
  - PHP
  - SQL query
  - Oracle



Frost Schema XML

```
$Frost DefineGauge "Report" "" "text/html"
$Frost DefineGauge "Time" "" "numeric/double"
$Frost DefineGauge "Image" "" "image/gif"


$Frost DefineTest "Link" "" [list Errors Warnings Log]
$Frost DefineTest "Compile" "" [list Errors Warnings Log]
$Frost DefineTest "Summary" "" [list Status Report]
$Frost DefineTest "UseCase" "" [list Log Time Image]

set Project [$Frost NewTestGroup Project ""]
$Project AddTest Summary
set Build [$Project AddTestGroup Build ""]
$Build AddTestGroup Libraries "" [list Link Compile]
$Build AddTestGroup Applications "" [list Link Compile]
$Project AddTestGroup Regression "" UseCase

$Frost DefineTestGroup $Project


$Frost Complete
```

- Simple format
- Describes structure
- Language independent

```
<Gauge Name="Report" Type="text/html" PartSpecific="f" />
<Gauge Name="Time" Type="numeric/double" PartSpecific="f" />
<Gauge Name="Image" Type="image/gif" PartSpecific="f" />
- <Test Name="Link">
  <Gauge Name="Errors" PartSpecific="f" />
  <Gauge Name="Warnings" PartSpecific="f" />
  <Gauge Name="Log" PartSpecific="f" />
 </Test>
+ <Test Name="Compile">
- <Test Name="Summary">
  <Gauge Name="Status" PartSpecific="f" />
  <Gauge Name="Report" PartSpecific="f" />
 </Test>
- <Test Name="UseCase">
  <Gauge Name="Log" PartSpecific="f" />
  <Gauge Name="Time" PartSpecific="f" />
  <Gauge Name="Image" PartSpecific="f" />
 </Test>
- <TestGroup Name="Project" Cardinality="1">
  <Test Name="Summary" />
 - <TestGroup Name="Build" Cardinality="1">
   <Test Name="1" />
  - <TestGroup Name="Libraries" Cardinality="1">
    <Test Name="Link" />
    <Test Name="Compile" />
   </TestGroup>
  - <TestGroup Name="Applications" Cardinality="1">
    <Test Name="Link" />
    <Test Name="Compile" />
   </TestGroup>
```

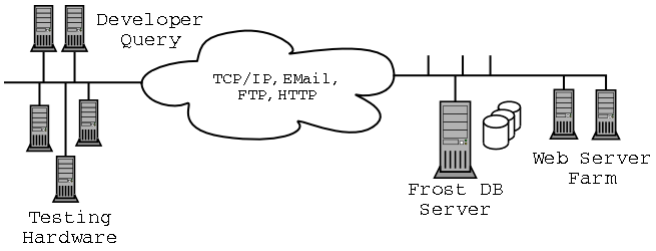
Frost Testing XML

```
public class Paper
{
  static void main ( String[] args )
  {
    Frost Frost = null;
    Frost = new Frost ( ... );

    Frost.beginTestGroupRun ( "Project" );
      Frost.beginTestGroupRun ( "Build" );
        Frost.beginTestGroupRun ( "Libraries" );

          Frost.beginTestRun ( "Compile" );
            // ... Processing Omitted ...
            Frost.setMeasurement ( "Errors", 0 );
            Frost.setMeasurement ( "Warnings", 10 );
            Frost.setMeasurement ( "Log", "...Omitted..." );
          Frost.endTestRun ( true );

          Frost.beginTestRun ( "Link" );
            // ... Processing Omitted ...
            Frost.setMeasurement ( "Errors", 0 );
            Frost.setMeasurement ( "Warnings", 0 );
            Frost.setMeasurement ( "Log", "...Omitted..." );
          Frost.endTestRun ( true );
        Frost.endTestGroupRun (); // Libraries
      Frost.endTestGroupRun (); // Build
    Frost.endTestGroupRun (); // Project
    Frost.complete();
  }
}
```

- Example of Test XML
- Java API

```
- <TestGroupRun Name="Project">
   <StartDateTime>Thu Apr 05 08:26:11 2001</StartDateTime>
  - <TestGroupRun Name="Build">
     <StartDateTime>Thu Apr 05 08:26:11 2001</StartDateTime>
    - <TestGroupRun Name="Libraries">
       <StartDateTime>Thu Apr 05 08:26:11 2001</StartDateTime>
      - <TestRun Name="Compile">
         <StartDateTime>Thu Apr 05 08:26:11 2001</StartDateTime>
        - <Measurement Gauge="Errors">
           <Value>0</Value>
          </Measurement>
        - <Measurement Gauge="Warnings">
           <Value>10</Value>
          </Measurement>
        - <Measurement Gauge="Log">
           <Value>...Omitted...</Value>
          </Measurement>
          <EndDateTime>Thu Apr 05 08:26:11 2001</EndDateTime>
          <Passed>t</Passed>
        </TestRun>
      - <TestRun Name="Link">
          <StartDateTime>Thu Apr 05 08:26:11 2001</StartDateTime>
        - <Measurement Gauge="Errors">
           <Value>0</Value>
          </Measurement>
        - <Measurement Gauge="Warnings">
           <Value>0</Value>
          </Measurement>
        - <Measurement Gauge="Log">
           <Value>...Omitted...</Value>
          </Measurement>
          <EndDateTime>Thu Apr 05 08:26:11 2001</EndDateTime>
          <Passed>t</Passed>
        </TestRun>
```

System Architecture



- No live connection required
  - FTP & email can be used to transport test results
  - Test data is time stamped
  - Provides flexibility
  - Allows for global test data to be collected

Case Studies

- VTK Testing
  - Features cross-platform comparison
  - Instant trending
- Help Desk
  - Summary of data in remote database
  - Highlevel view and trending
- Frost performance
  - Drill down capabilities to find the source of problems
  - Tracks source code changes

VTK Testing

VTK Dashboard for Wed May 24 20:03:30 EDT 2000

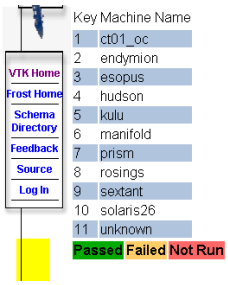
There were 15 files changed since the last dashboard. Changes for the month.

Platform	Build VTK		Tcl Image Tests				Cxx Image Tests		Other Tcl Tests		Other Cxx Tests	
	Errors	Warnings	Pass	Fail	Fast	Slow	Pass	Fail	Pass	Fail	Pass	Fail
irix65	0	18	478	5	3	26	51	0	6	1	7	1
irix6n32	0	16	480	3	0	3	51	0	6	1	7	1
irix6x64	0	18	480	3	0	2	51	0	6	1	7	1
solaris	1	20	446	7	0	7	45	0	6	1	7	1
solarisCoverage	1	3	467	10	0	8	50	0	6	1	7	1
WinNT	0	3	473	10	0							
hp	2	0	475	8	0							
linuxRH52	0	5	476	5	0							
solaris26m6	0	7	478	5	0							
solaris26	0	0	436	33	0							

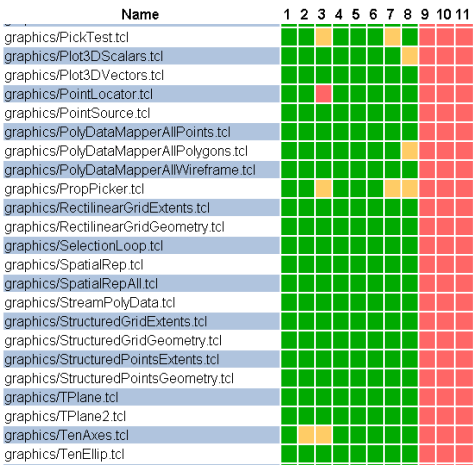
- Transition from static to dynamic
- Comparison between yesterday and today

Platform	Build VTK		Tcl Image Tests		Cxx Image Tests		Other Tcl Tests		Other Cxx Tests	
	Errors	Warnings	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail
hp	1	0	540	2	53	0	8	1	10	0
irix65	0	0(1)	541	2(1)	53	0	8	1	10	0
irix6n32	1	2	541	1	53	0	8	1	10	0
irix6x64	1(0)	0(1)	541	2(1)	53	0	8	1	10	0
linuxRH52	0	74	530	7	54	0	7	2	10	0
solaris	0	3(4)	510	2(1)	48	0	8	1	10	0
solaris26	0	0								
solaris26m6	0	0(1)	521(534)	22(7)	53(51)	0	8(NA)	1(NA)	10(NA)	0(NA)
solarisCoverage	0	3(4)	531(519)	8(7)	52	0	8	1	10	0
unknown	No Data									
WinNT	0	1	313	1						

Cross platform comparison



- Graphical overview
- "Quality-at-a-glance
  - Green: Passed
  - Yellow: Failed
  - Red: Not Run



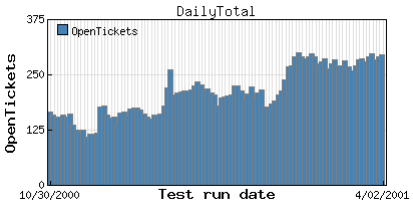
Help Desk

QUEUES	TICKETS				
	Opened Last 24 Hrs	Closed Last 24 Hrs	Remaining Open	Missed SLA	View
DailySummary	113 Created (11 MTD)	104 Closed (7 MTD)	284 Open	17 Missed SLA	View all History

Level	On-Time	%	Avg	Std Dev	MTD OT	MTD %	MTD Avg	MTD Std Dev
Critical History	0 of 0	0%	0:00 Hrs	0:00 Hrs	0 of 0	0%	0:00 Hrs	0:00 Hrs
Urgent History	0 of 0	0%	0:00 Hrs	0:00 Hrs	0 of 0	0%	0:00 Hrs	0:00 Hrs
Normal History	68 of 85	80%	7:21 Hrs	18:49 Hrs	7 of 7	100%	0:03 Hrs	0:07 Hrs
Other History	19 of 19	100%	67:34 Hrs	57:38 Hrs	0 of 0	0%	0:00 Hrs	0:00 Hrs

DailySummary ADS Desktop/Printers EHPC IMAC Network-Data Network-Voice Servers ServiceDispatch Ser
WebMaster iProcess

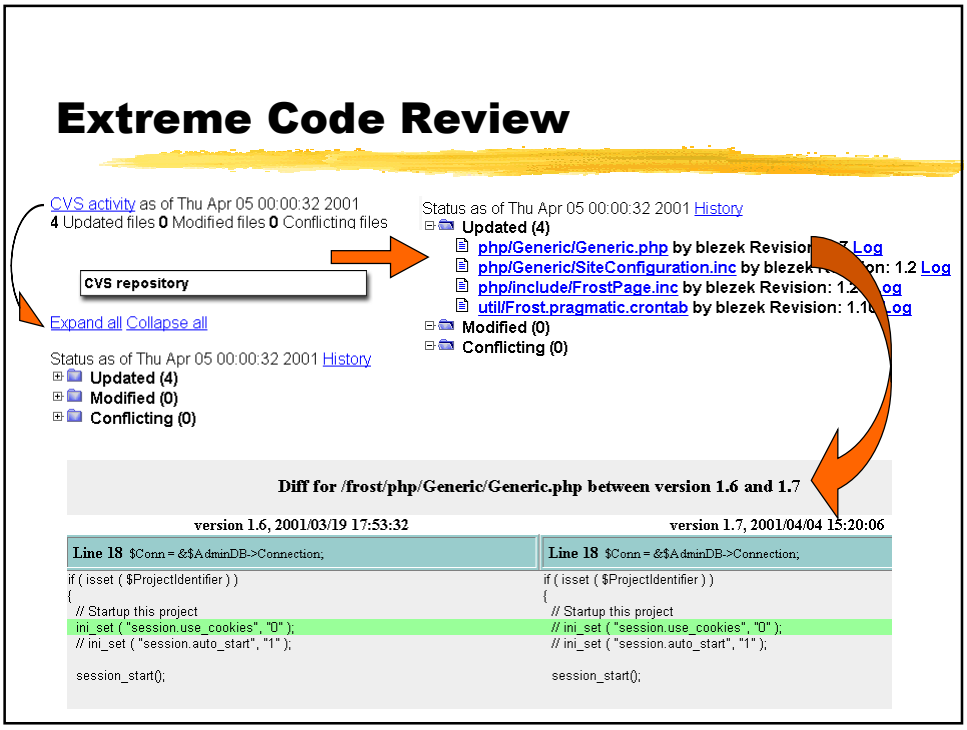
- Captures ticket statistics
- Continuous open ticket log
- Trends



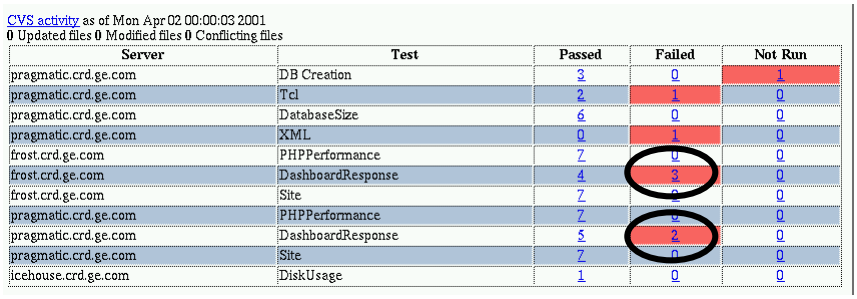
Muse: Testing Frost

- Collect summary of source code changes
  - For instantaneous code review
  - Links to code changes
- Checks 7 critical dashboards for time to load
  - 8 seconds is upper limit
  - Detailed drill down information
- Monitors database size
- Checks pages for errors and warnings

Extreme Code Review



Muse



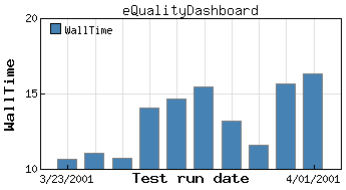
Server	Test	Passed	Failed	Not Run
pragmatic.crd.ge.com	DB Creation	3	0	1
pragmatic.crd.ge.com	Tcl	2	1	0
pragmatic.crd.ge.com	DatabaseSize	6	0	0
pragmatic.crd.ge.com	XML	0	1	0
frost.crd.ge.com	PHPPerformance	7	0	0
frost.crd.ge.com	DashboardResponse	4	3	0
frost.crd.ge.com	Site	7	0	0
pragmatic.crd.ge.com	PHPPerformance	7	0	0
pragmatic.crd.ge.com	DashboardResponse	5	2	0
pragmatic.crd.ge.com	Site	7	0	0
icehouse.crd.ge.com	DiskUsage	1	0	0

- Not so good day
  - Drill down for details

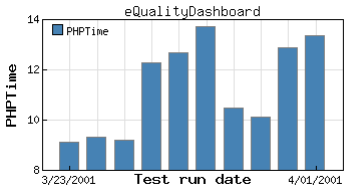
Dashboard Response Details

eQualityDashboard: Failed

- URL: http://pragmatic.crd.ge.com/Continuous/eQuality/project/eQuality.php
- WallTime: 16.354874
- SizeInK: 292.798000
- PHPTime: 13.376000
- Test Run Detail

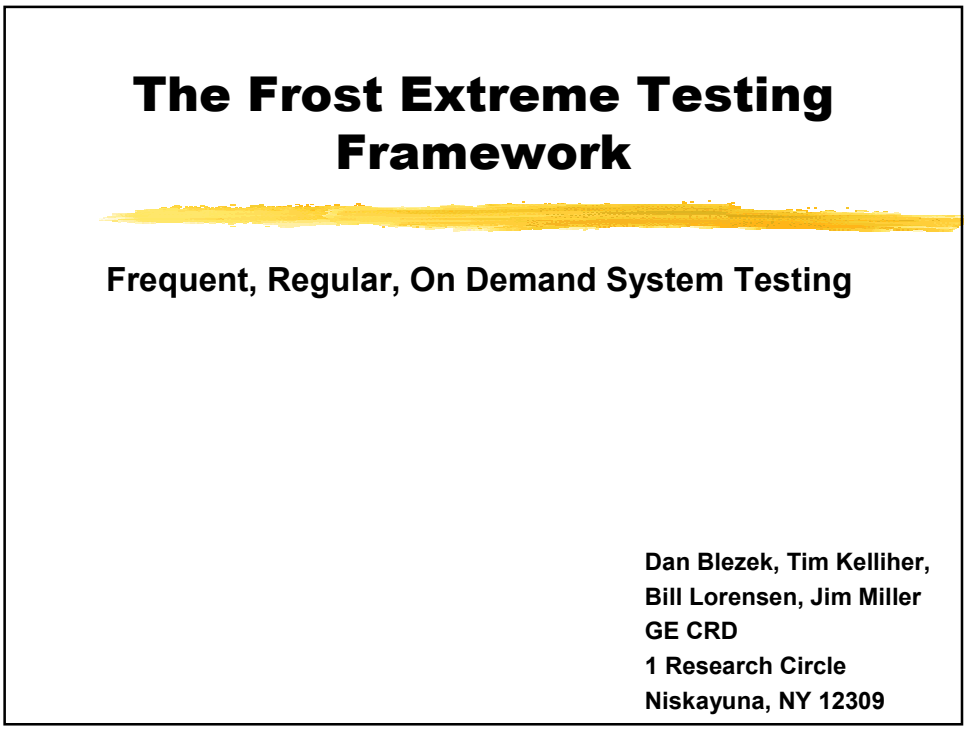


- eQuality Dashboard failed
  - Wall time is 16.3 seconds
  - Generation took 13.3 seconds
- Is this a trend?
  - Yup!



Conclusions

- Frost is our Extreme Testing tool of choice
  - Based on open standards
  - Flexible, scalable architecture
- Concepts are simple, powerful and elegant
  - Gauge, Test, TestGroup, Part
- Frost provide a solid foundation for Extreme Programming


The Frost Extreme Testing Framework
Frequent, Regular, On Demand System Testing
Dan Blezek, Tim Kelliher,
Bill Lorensen, Jim Miller
GE CRD
1 Research Circle
Niskayuna, NY 12309

The Frost Extreme Testing Framework

Frequent Regular On-demand System Testing

Daniel Blezek
518-387-5481, blezek@crd.ge.com

Timothy Kelliher
518-387-6691, kelliher@crd.ge.com

William Lorensen
518-387-6744, lorensen@crd.ge.com

James Miller
518-387-4005, millerjv@crd.ge.com

GE Corporate R&D
KW/C218C
1 Research Circle
Niskayuna, NY 12309

Introduction

It has been our experience that small to medium software development efforts in a distributed environment do not have the resources to staff a full software quality assurance department. Indeed, having a SQA for a small or medium development team can significantly impede the development and release process. Software quality should not be ignored in these cases, rather it should be built in to the development process itself, effectively ensuring high quality software. To address this need, we have built a non-intrusive framework to collect, summarize, and trend software and system testing results. Our system, named Frost (Frequent Regular, On-Demand System Testing), allows the developer in the small project to easily deploy a software testing infrastructure in the initial phases of the project. Incorporating software testing using Frost in the early stages of development leads to increased project transparency, and quality.

Extreme Programming

Extreme Programming (XP) is an emerging philosophy aimed at introducing a lightweight methodology for small-to-medium development teams. XP takes commonsense principles and moves them to extreme levels. One of the tenants of XP is testing. To quote Kent Beck: "If testing is good, everybody will test all the time (unit testing), even the customers (functional testing)".

The Extreme Programming (XP) tenet of testing is often overlooked in software development efforts. Frequently, this is due to tight development schedules, and reluctance on the part of developers to expend effort in areas that are perceived to have no value to the final software product. It is only when testing is an integral part of the software development processes that it becomes effective. Only when the developer has an easy mechanism of creating, running, and verifying tests does the Extreme Testing (XT) aspect of XP become a reality.

A significant barrier to implementing XT within a project is the overhead of designing, creating, and maintaining a XT framework. Usually, these test frameworks are hastily put together by a single developer, without a thought to reuse in future projects. In our experience testing frameworks prove to be highly non-portable, even with significant effort on the part of the framework developer or developers.

To overcome this barrier in our group, we have obtained a set of items that are Critical To Quality (or CTQ's) of a testing framework. The framework must be:

- **Reusable**
  A general framework that can be adapted by different projects.
- **Distributed**
  Anyone should be able to contribute testing data from anywhere on any device.
- **Persistent**
  Testing observations must be preserved for future analysis, and for trending information.
- **Structured**
  Structured test observations facilitate efficient summarization of the data.
- **Executed regularly**
  To be effective, an XT framework must allow several levels of execution: frequent, regular, and on-demand. Frequent testing encourages frequent code changes. Regular testing is performed at scheduled intervals, often nightly, and collects the overall view of software development activities for the past 24 hours. On-demand testing allows each developer to make local changes and immediately evaluate them against the established gold standard.
- **Independent of presentation**
  Often testing results are created as part of a web page, or in some difficult to use file format.

- **Dynamic**

Test observations should be summarized on-demand, as it is collected, and available in many different formats.

The Frost framework addresses these CTQ's, and has proven to be a robust and capable infrastructure. The name Frost has dual meanings, first it is an acronym for Frequent Regular On Demand Systems Testing, and secondly honors the poet Robert Frost who wrote:

> ***Two roads diverged in a wood, and I--***
> ***I took the one less traveled by,***
> ***And that has made all the difference.***
> Robert Frost

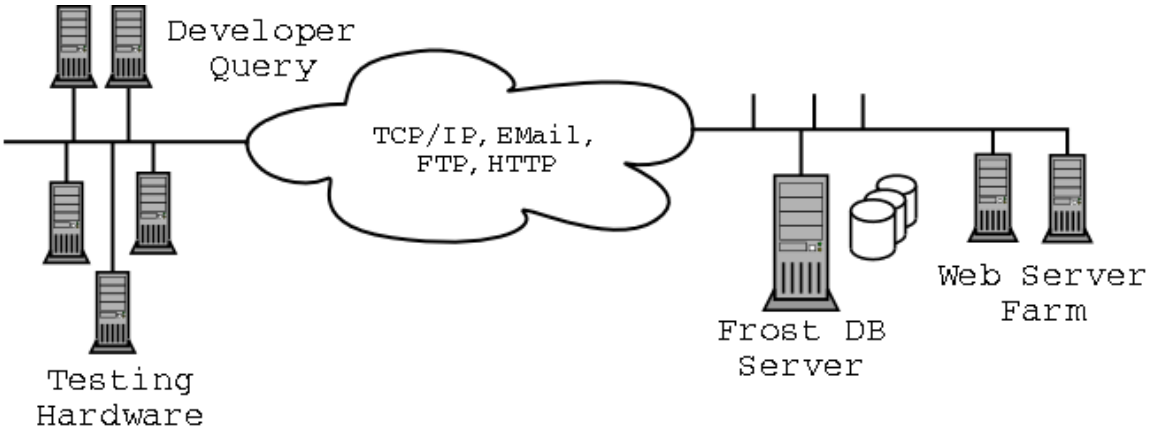
Software testing is often the road less traveled, but it does make all the difference.

This paper details the origins of Frost, the current architecture, and demonstrates some of the projects currently using Frost.

Background

Frost is a powerful component of XP. At the core of Extreme Programming is testing. By testing the software throughout the development process, XP ensures the software always performs to the project requirements. Though several testing frameworks exist, they fail to address the problems of collection, summary and trending over the development lifecycle. Frost provides for distributed collection of software testing results and a central location of trending and summarization of testing results. Frost has grown out of the desire to disseminate the VTK testing framework to other development efforts, and to distill our knowledge of software testing as presented by Bill Lorensen at Software Quality Week 2000.

Frost is our new system, built upon the lessons and experiences of VTK, providing mechanisms for collecting, summarization and trending of test observations by distributed developers. Frost is designed to give the developer a transparent view of the software at any time. Dashboards provide the top-level overview of the project's quality, with detailed information available at a click. By increasing the project's transparency, Frost allows development to proceed unhampered by the need to correct for errors and bugs introduced by other developers. Bugs that would normally be caught by the SQA department, or worse, by the end user, can be found and eliminated before leaving the developers hands. The Frost framework consists of three components: test result collection, result storage, and reporting front end. In the diagram below, the Testing Hardware performs testing and reports the results to the Frost DB Server. This process frequently happens overnight and perhaps throughout the day, to create a continuous snapshot of the project's quality. The Frost DB Server processes the test results, validates their content, and stores the results into an Oracle RDBMS. At any time, a developer may make a query of any testing results. Typically, a project will have created a dashboard page that is delivered through the web server farm. From the dashboard, the developer can drill down into any of the testing results captured by the testing hardware.



Frost

Philosophy

In XP, if a thing is good to do, take it to the extreme. In Extreme Testing, we feel that there are two central tenets:

- If it isn't tested, it's broken.
- If it isn't automated, it doesn't get done.

We have repeatedly found that portions of our software that are not tested are the primary source of user reported bugs. Our group primarily develops application libraries: as with any library, their exist functions and objects that are used with less frequency. During the course of development, little used and untested code is considered broken. It is often the case that bugs are found in "dusty" code that has not been covered in a test case. VTK has been developed over the course of the last 7 years, and through time, essentially all of the code has been scrutinized for correctness and regression tests have been written for it.

In many software projects, attempts are made to raise the quality of the development process, but they often fall to the wayside as schedules slip. Our software is no exception. We have found that by automating each project that normally would have raised the quality of the project temporarily, we are able to use the project to permanently raise the quality of the software. For instance, a project may involve instrumenting the code to test for memory leaks. If this is done once, the quality of the project is temporarily raised; if it is automated, each day the development team can look at the testing dashboard to see if any memory leaks have cropped up in the code.

Concepts

Frost has several key concepts: Parts, Gauges, Tests and TestGroups. Parts are components of the system to be tested, and can range from a single class, or file, to an entire sub-system and are organized hierarchically. Gauges are defined to take a single measurement, and may be applied to Parts. Tests are collections of Gauges that are applied while performing a TestRun. TestGroups organize Tests into a hierarchy, and instances of TestGroupRuns are created during the execution of a TestGroup. These concepts succinctly represent the informal process through which we had been doing software testing. The concepts are general in nature, allowing a wide range of system and software testing to be supported by Frost.

Gauges

Gauges are at the core of Frost. Each Gauge has an associated type, corresponding to the sort of data the Gauge records as a Measurement. The current Gauge types are: integer, double, text, html, string, URL, Gif, Jpeg, XML, and XML processed by XSL. For instance, the number of CPU seconds that a process consumes would be recorded using a Gauge of type double. For Gauges with a type of XSLT, the Gauge is associated with an XSL description of how to process the XML Measurements collected by the Gauge. As an example, consider an existing process that produces compiler warnings in XML. By recording this log with a Gauge of type XSL, the error log can be formatted on the fly into any desirable configuration.

For Gauges that collect arbitrary sized data and may not require the data to be stored indefinitely, the Gauge may be defined to have a LifeTime in days. After the LifeTime of the Gauge has expired, the Measurement that the Gauge collected is removed from Frost. Gauges such as integer and double are never removed from Frost.

Tests

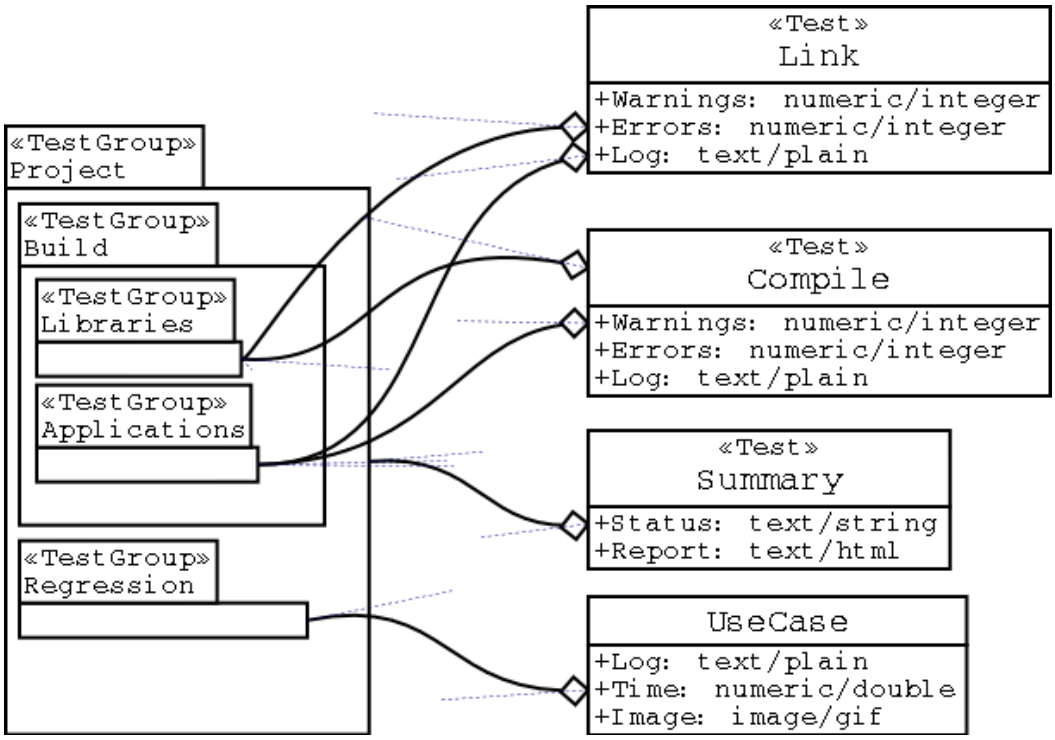
Tests are simply collections of Gauges. Generally, Tests group similar Gauges together. When a Test is executed, a TestRun of that Test is created. At the conclusion of the TestRun, the Test writer evaluates the pass/fail status of the Test. This information, along with Test start and end times is recorded into Frost.

TestGroups

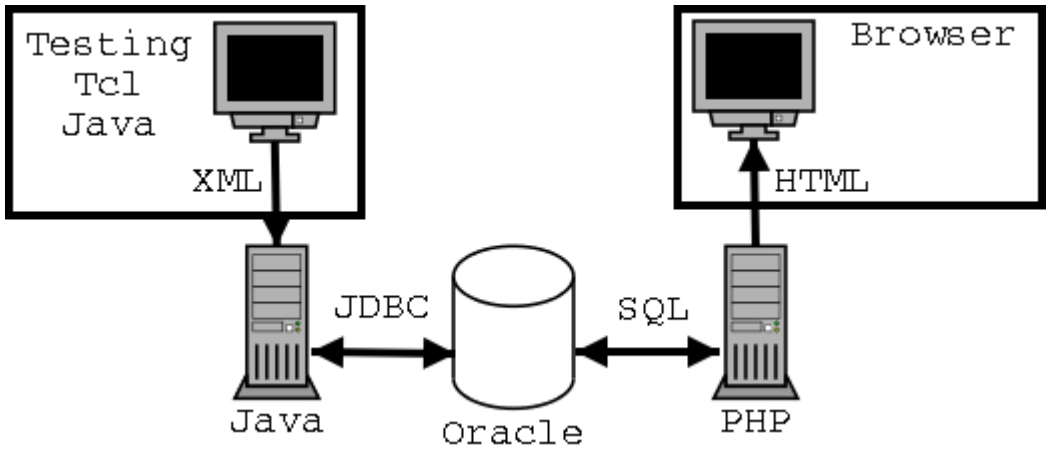
TestGroups organize and cluster Tests and sub-TestGroups. A TestGroup normally is defined to contain only one copy of each associated Test. However, a TestGroup may be defined to have a cardinality of 0..N, indicating that each contained Test may be recorded zero or many times during the execution of the TestGroup (known as a TestGroupRun).

Example

Consider a TestGroup called Project that contains a Test called Summary, and two TestGroups, Build and Regression. Summary contains two Gauges: Status, a string Gauge, and Report, an HTML Gauge. Build contains two TestGroups, Libraries and Applications. Libraries and Applications each contain two Tests, Compile and Link. In turn, Compile and Link have three Gauges associated with them, Warnings and Errors, integer Gauges, and Log, a text Gauge. The Regression TestGroup has a cardinality of 0..N, and contains only one Test, UseCase. The Test UseCase contains the previously defined Log Gauge, a double Gauge called Time, and a Gif Gauge called Image. The hierarchy is graphically represented below.



System Architecture



Testing

Testing results are collected by Frost in the Extensible Markup Language (XML). Thus, the developer need not have access to any special software libraries, and is free to do testing in any language and/or platform desired. When testing results are ready, the XML is processed by a server connected to the underlying Oracle RDBMS. The XML results may be delivered through a variety of mechanisms, direct socket connection, HTTP, FTP, or SMTP, supporting distributed development. The Frost XML file format is rigorously described in a Document Type Definition (DTD) format, allowing XML code to be validated before delivery to Frost. The server, written in Java, parses the XML file, validates the correctness against the stored definitions of Gauges, Tests

and TestGroups, and then enters the testing results into the database for later retrieval and trending. Although the server expects XML data, any programming language may be used to generate and deliver the XML to the server. Currently, Java and Tcl are the two provided API's, presenting the test developer with an easy to use mechanism for creating and delivering XML to Frost.

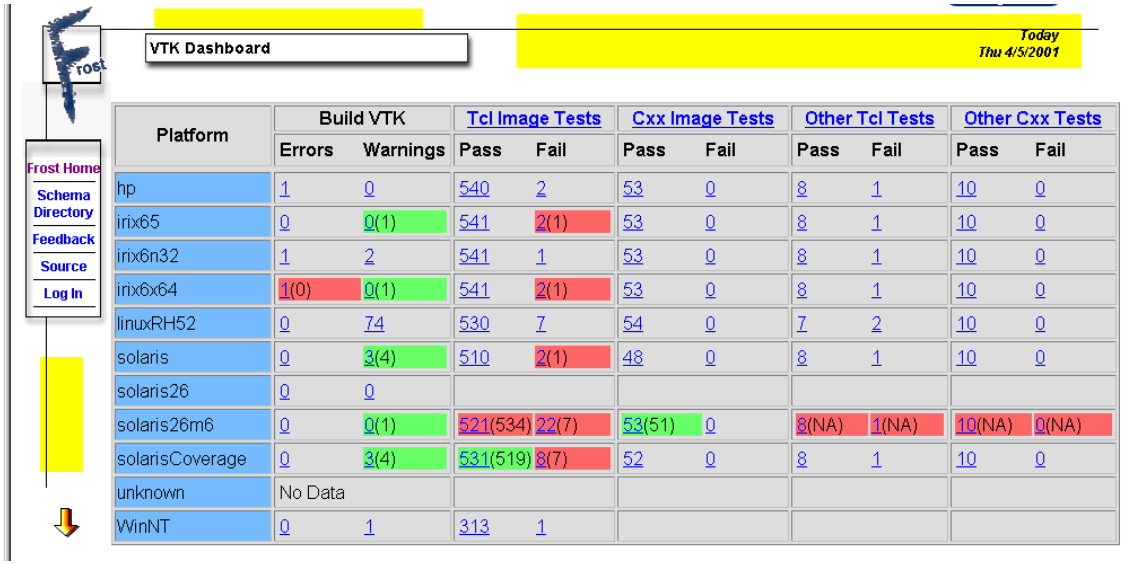
Test Reporting

From the RDBMS, testing results are presented, summarized and trended using the PHP server side scripting language. To query the stored testing results, an object of the TestGroup class is instantiated and test results are queried from the RDBMS, filling out the testing hierarchy. All reports are generated on-demand, and delivered via the browser. The Frost web site is designed to allow easy navigation to any desired level of granularity, from an individual gauge on a particular day, to the full history of a particular test for the past year. PHP includes graphing capabilities, providing an on-demand trend view of the stored test results through time.

Case Studies

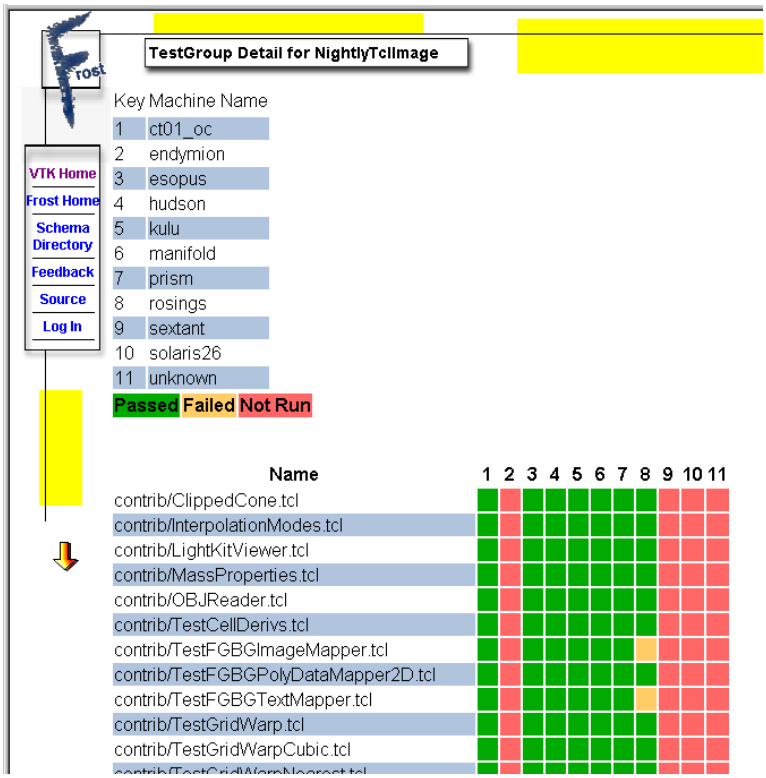
VTK Testing

The VTK testing framework has been extended to include a Frost dashboard. The VTK library is built on 11 different platforms nightly, and 500+ regression tests are performed. The figure below shows the VTK Frost dashboard. A cell is colored green if the change from yesterday was positive, red if the change was negative and gray if there was no change. The first row of the table shows 1 build error for the "hp" platform, with no change from yesterday. The second row shows that the "irix65" platform had one less build warning than yesterday, but one more test failed today in the "Tcl Image Tests" TestGroup.



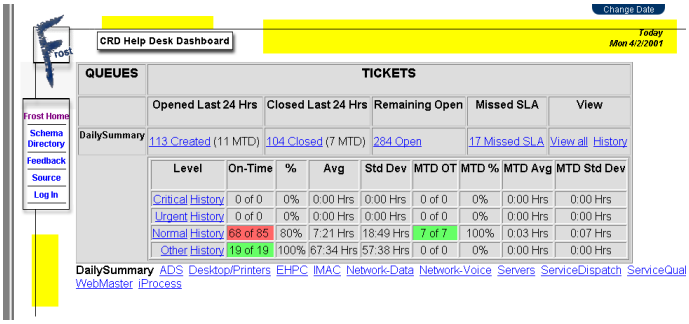
Platform	Build VTK		Tcl Image Tests		Cxx Image Tests		Other Tcl Tests		Other Cxx Tests	
	Errors	Warnings	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail
hp	1	0	540	2	53	0	8	1	10	0
irix65	0	0(1)	541	2(1)	53	0	8	1	10	0
irix6n32	1	2	541	1	53	0	8	1	10	0
irix6x64	1(0)	0(1)	541	2(1)	53	0	8	1	10	0
linuxRH52	0	74	530	7	54	0	7	2	10	0
solaris	0	3(4)	510	2(1)	48	0	8	1	10	0
solaris26	0	0								
solaris26m6	0	0(1)	521(534)	22(7)	53(51)	0	8(NA)	1(NA)	10(NA)	0(NA)
solarisCoverage	0	3(4)	531(519)	8(7)	52	0	8	1	10	0
unknown	No Data									
WinNT	0	1	313	1						

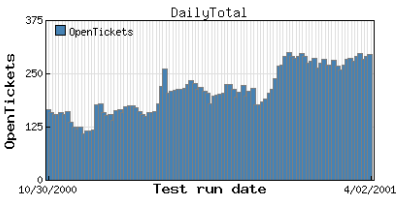
A feature of Frost is the ability to compare the performance of a test across platforms. The figure below shows a table comparing test results across platforms. The rows of the table are the tests, and the columns are the platforms. A green result indicates the test passed, yellow indicates a failure, and red indicates that the test did not run on that platform. This view of the data enables the developer to quickly scan a graphical view of the nightly regression tests searching for trends in the data.



Help Desk Status

The Help Desk project was created to provide a high level view of the IT support function of our center. Each night, statistics are collected from the help desk's Clarify database and reported in a Frost dashboard. The number of tickets created, and closed in the previous day are presented on the dashboard, including a month to date total. Tickets that were closed outside of the time limit for the urgency are also collected. The long-term trends show a gradual increase in the number of pending tickets. A log of currently open tickets is recorded every 20 minutes through the day, providing up to date information about the status of the Help Desk system. From the open ticket report, a link goes directly to the details for an individual ticket in the Clarify system.





Frost performance

Frost is also used to monitor itself in a project called Muse. The Muse dashboard presents a summary of the source code changes from the previous day, and the results of nightly testing. Several tests are available: each core function of Frost is executed and evaluated for errors and warnings. Each dashboard in the Frost system is tested to ensure that it loads within 8 seconds,

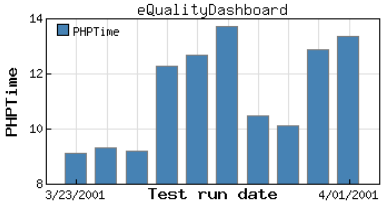
and the available disk space on the Oracle server is monitored to avoid depleting the available capacity. The dashboard below gives a quick overview of the system quality to the Frost developer, with immediate drill down capabilities.

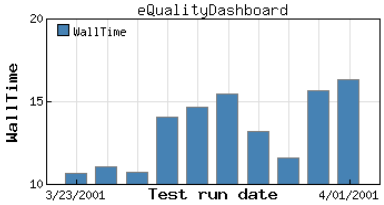
0 Updated files 0 Modified files 0 Conflicting files

Server	Test	Passed	Failed	Not Run
pragmatic.crd.ge.com	DB Creation	3	0	1
pragmatic.crd.ge.com	Tcl	2	1	0
pragmatic.crd.ge.com	DatabaseSize	6	0	0
pragmatic.crd.ge.com	XML	0	1	0
frost.crd.ge.com	PHPPerformance	7	0	0
frost.crd.ge.com	DashboardResponse	4	3	0
frost.crd.ge.com	Site	7	0	0
pragmatic.crd.ge.com	PHPPerformance	7	0	0
pragmatic.crd.ge.com	DashboardResponse	5	2	0
pragmatic.crd.ge.com	Site	7	0	0
icehouse.crd.ge.com	DiskUsage	1	0	0

eQualityDashboard: Failed

- URL: http://pragmatic.crd.ge.com/Continuous/eQuality/project/eQuality.php
- WallTime: 16.354874
- SizeInK: 292.798000
- PHPTime: 13.376000
- Test Run Detail





Conclusions

We have developed a web based testing framework designed to give the small to medium sized development team a non-invasive method of incorporating testing, trending, and analysis into their development efforts. Frost is based on open standards, XML, SQL, Java, and PHP, resulting in a flexible, scalable architecture for gathering, storing, and reporting on software and system testing.

The Frost system has proved to be extremely flexible in handling a wide range of software and system testing tasks. The core concepts of Gauges, Tests and TestGroups have proved to cover the requirements for all projects we have encountered. The simplicity of the framework, and the ease of generating testing results is beneficial to the rapid acceptance and adoption of the Frost framework.

QW2001 Paper 2A1

Mr. James Tierney
(Microsoft)

Getting Started With Model-Based Testing

Key Points

- How to improve your effectiveness through model-based testing
- How to test with models from Day One of your project
- How to keep “growing” your models so that they keep finding bugs

Presentation Abstract

Traditional testing Traditional software testing consists of the tester studying the software system and then writing and executing individual test scenarios that exercise the system. These scenarios are individually crafted and then can be executed either manually or by some form of capture/playback test tool. But hands-on testing and handcrafted test scripts are labor-intensive and inefficient ways to test modern software. These methods of creating and running tests face at least two large challenges: First, these traditional tests will suffer badly from the “pesticide paradox” (Beizer, 1990) in which tests become less and less useful at catching bugs, because the bugs they were intended to catch have been caught and fixed. Second, handcrafted test scenarios are static and difficult to change, but the software under test is dynamically evolving as functions are added and changed. When new features change the appearance and behavior of the existing software, the tests must be modified to fit. If it is difficult to update the tests, it will be hard to justify the test maintenance costs. Model-based testing alleviates these challenges by generating tests from explicit descriptions of the application. It is easier, therefore, to generate and maintain useful, flexible tests. Modeling Modeling is a way to represent the behavior of a system. Models are simpler than the system they describe, and they help us understand and predict the system’s behavior. Models are a useful method of representing software behavior. Models provide an easy way to update tests to keep pace with applications that are constantly changing and evolving. In recent years, there has been a growing movement in software testing to use the information contained in explicit models of software behavior to make it simpler and cheaper to do testing. Model-Based Testing Model-based testing is a black-box technique that offers many advantages over traditional testing:

- Constructing the behavioral models can begin early in the development cycle.

- Modeling exposes ambiguities in the specification and design of the software.
- Simple models can be used for testing from the very first day of the project.
- Models embody behavioral information that can be re-used in future testing, even when the specifications change.
- The model is easier to update than a suite of individual tests.
- The models can evolve alongside the software and will continue to find bugs throughout the development cycle.

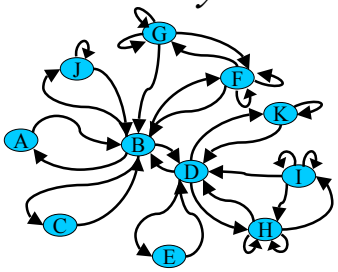
Getting Started With Model-Based Testing Using simple programmatic test tools and familiar applications, this presentation makes the case for intelligent, model-based test automation and shows how to apply it from the very first day of the product life cycle to deliver high-quality software.

About the Author

James Tierney moved into his current position of Test Architect for Microsoft Windows User Experience after being a Test Manager, Test Training Manager and Director of Test. He first got interested in Model-based testing in 1983 while testing at Fortune Systems in Silicon Valley. Finding Microsoft fertile ground for advanced testing ideas, he is helping create a company wide Model-based testing architecture. He has a couple of patents on Model-based testing, and has given presentations on MBT, Software Reliability Engineering, Poka Yoke (Mistake Proofing) Software, Testing, and Test Management.

Getting Started with Model-Based Testing

Harry Robinson
James Tierney
Jason Taylor



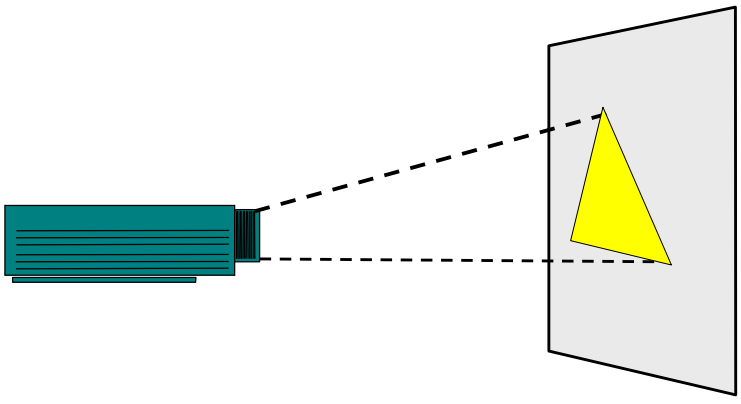
Why Model-Based Testing?

- Improve specs
- More nimble test automation
- Earlier test automation
- Better relationship with Dev/PM, working side by side, rather than against.
- Attract and retain high quality testers

What is Model-Based Test Automation?

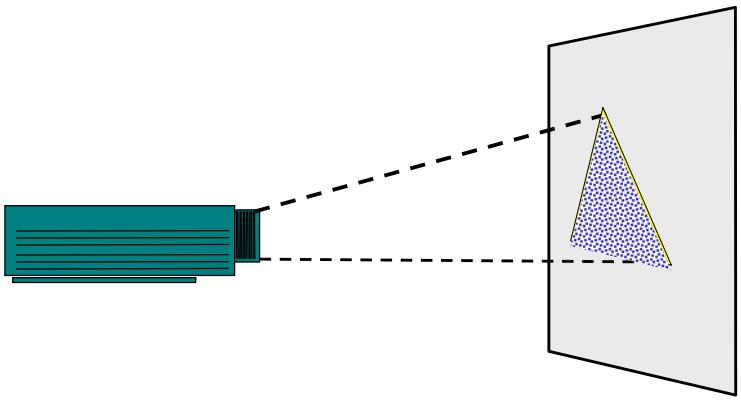
- Develop a model or map of testable parts of the application
- Determine verification points in the model
- Create test scripts by traversing the model
- Execute test scripts, fix bugs
- Note which important bugs the model missed, improve the model to catch

Traditional Testing



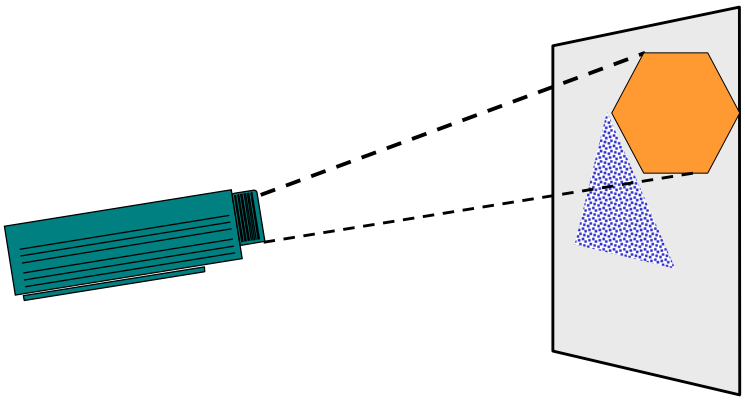
Imagine that the projector is your software under test ...

Traditional Testing



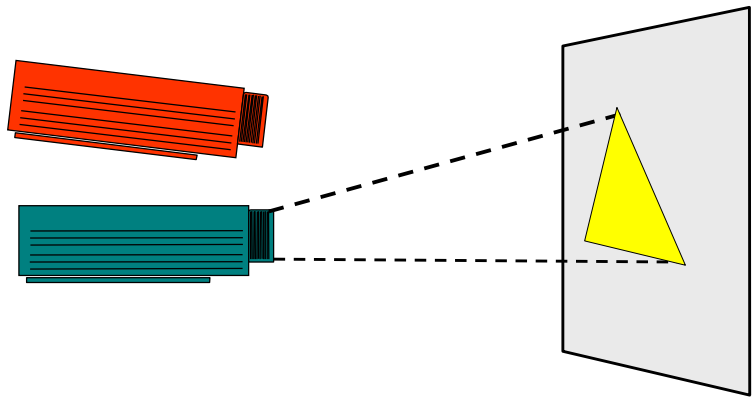
Here's traditional testing

Traditional Testing



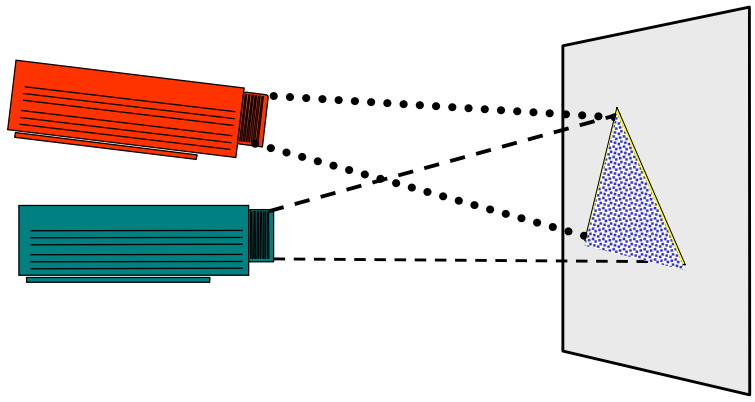
But what happens when the software changes?

Model-Based Testing



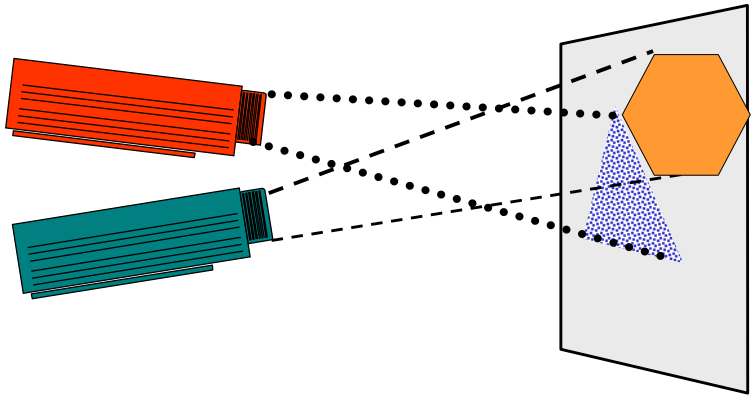
Now imagine that the top projector is your model ...

Model-Based Testing



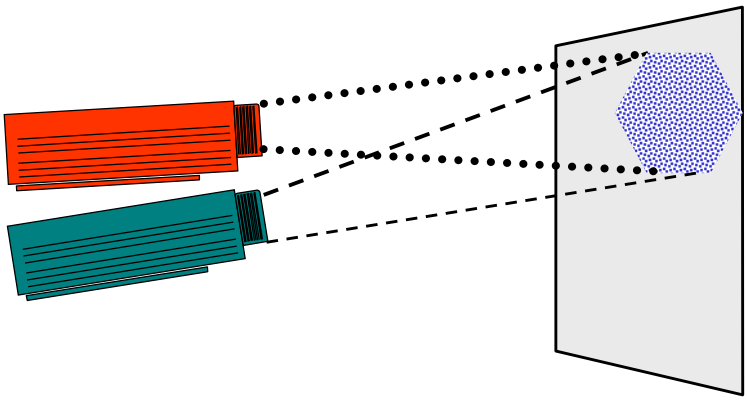
Here's model-based testing

Model-Based Testing

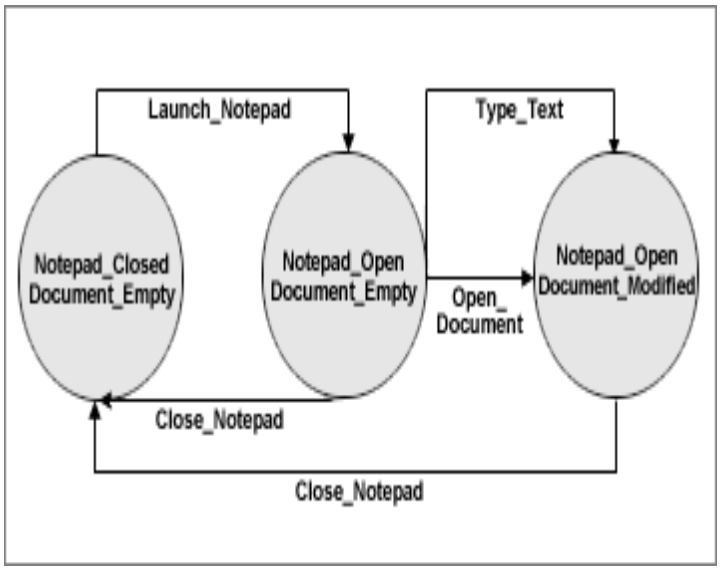


When the software changes ...

Model-Based Testing



… so do the tests.



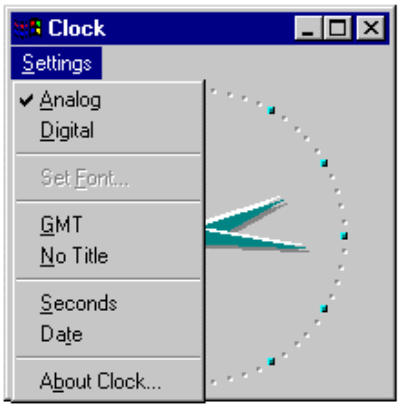
Approaches to Automated Testing

Static Tests

Monkey Tests

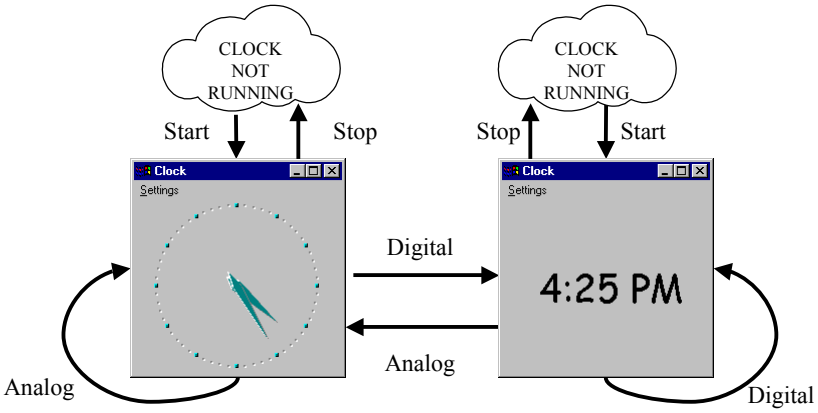
Model-Based Tests

The NT Clock



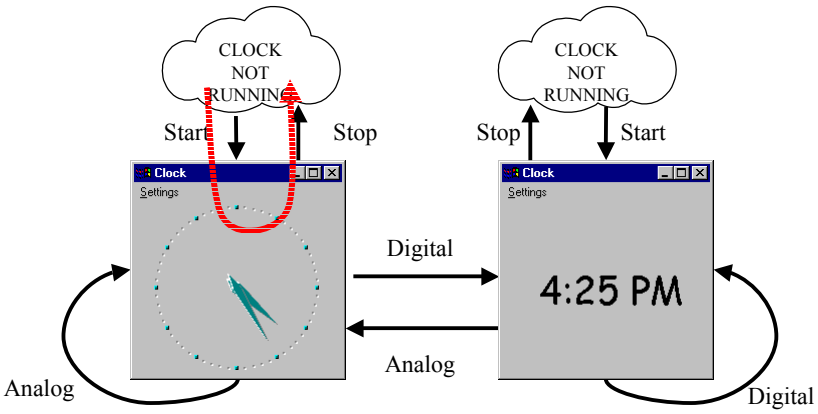
- Familiar
- Simple enough
- Complex enough
- Hard to test

NT Clock Behavior



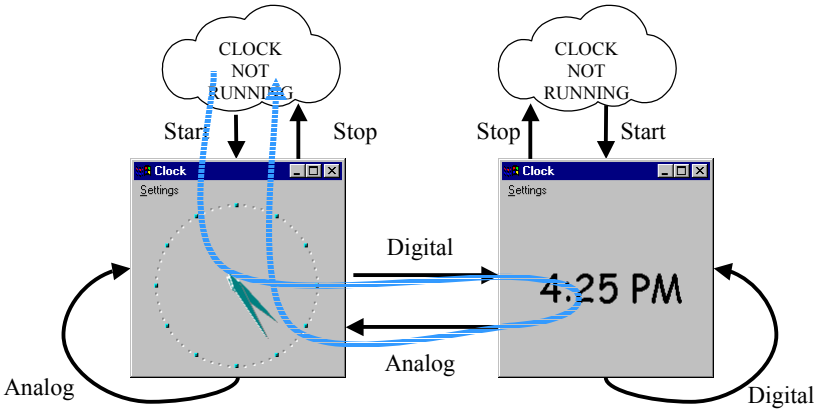
Static Tests vs. The Clock

Test Case 1: Start Stop



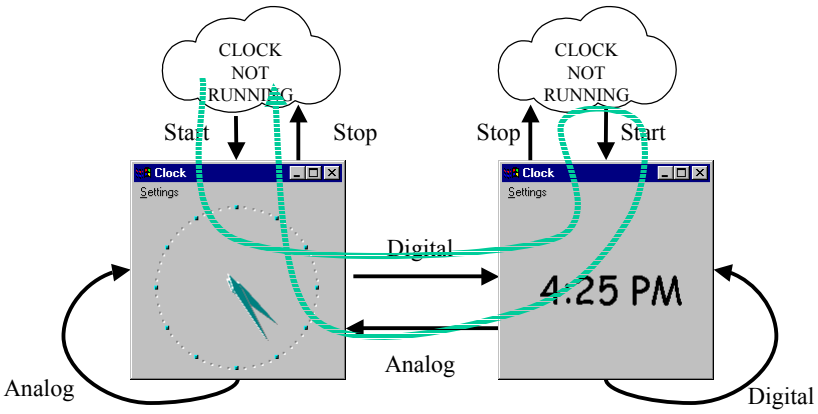
Static Tests vs. The Clock

Test Case 2: Start Digital Analog Stop



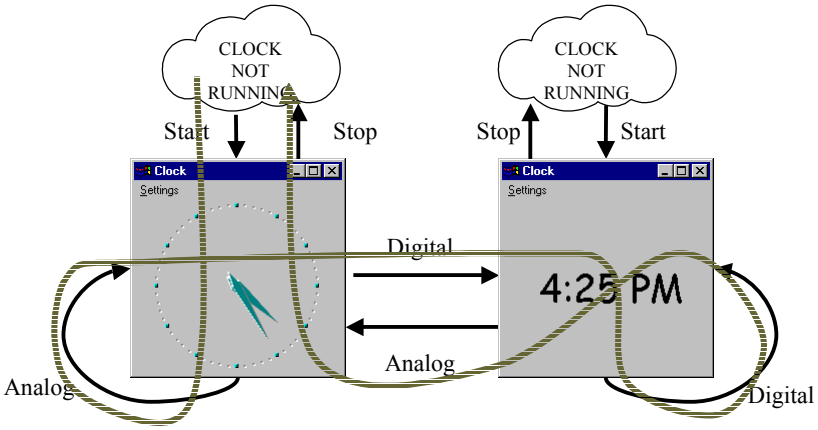
Static Tests vs. The Clock

Test Case 3: Start Digital Stop Start Analog Stop



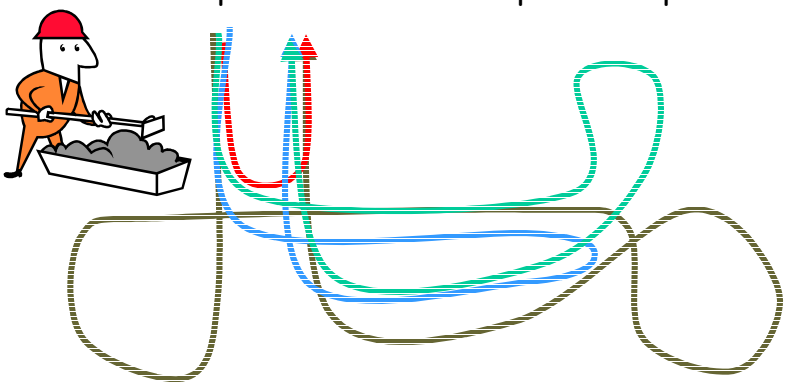
Static Tests vs. The Clock

Test Case 4: Start Analog Digital Digital Analog Stop



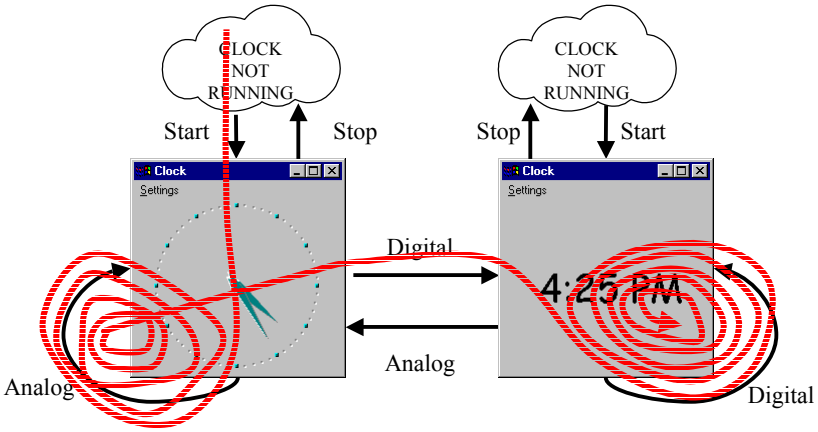
Static Tests vs. The Clock

- Hard-coded test cases – lots of 'em
- Tests do only what you tell them to
- Scripts wear out due to pesticide paradox

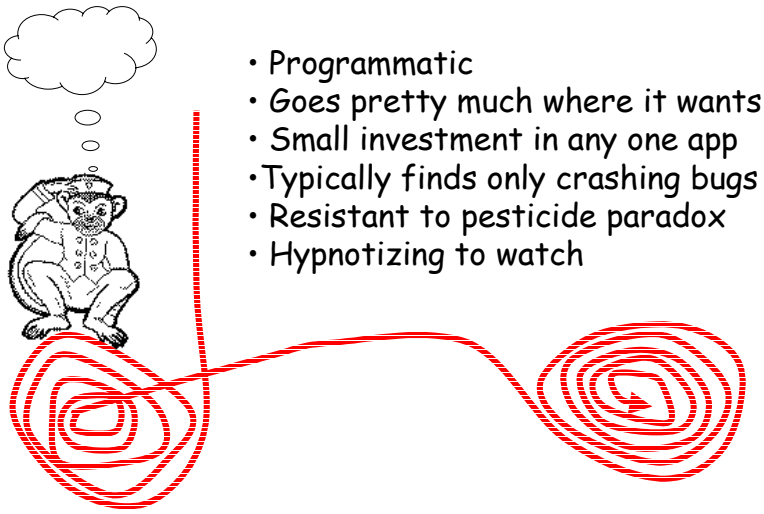


Monkey Tests vs. The Clock

Start Analog Analog Analog Analog Analog Analog …



Monkey Tests vs. The Clock

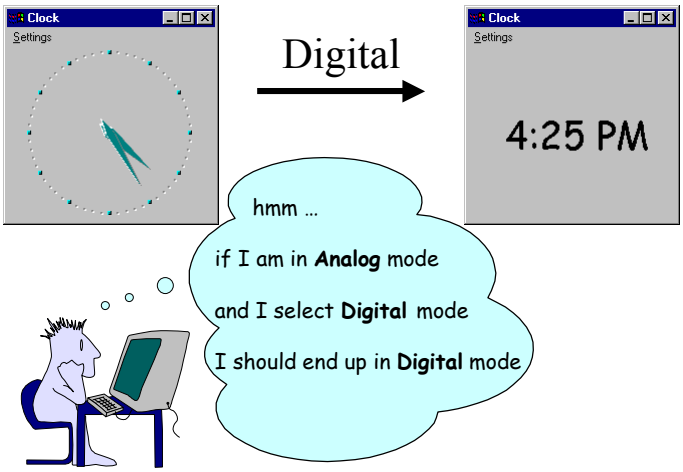


- Programmatic
- Goes pretty much where it wants
- Small investment in any one app
- Typically finds only crashing bugs
- Resistant to pesticide paradox
- Hypnotizing to watch

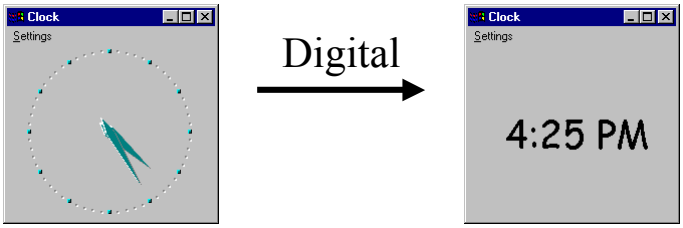
So What's a Model?

- A model is a description of a system's behavior.
- Models are simpler than the systems they describe.
- Models help us understand and predict the system's behavior.

We All Use Models Already



How to Use Models in Testing



Setup: Clock is in Analog mode

Action: Select Digital mode

Outcome: Did Clock go correctly to Digital mode?

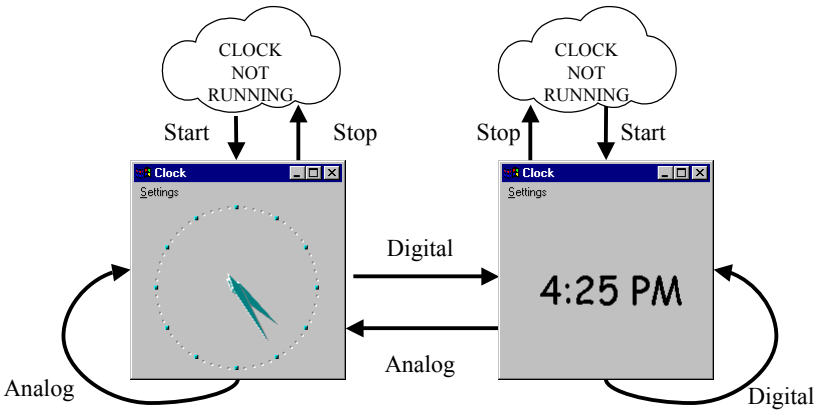
How to Create Model-Based Tests

1. Create a behavioral model of the application
2. Generate interesting test actions
3. Execute the test actions
4. Determine if the application worked
5. Find bugs

Step 1:

Create a behavioral model
of the application

Clock Behavior



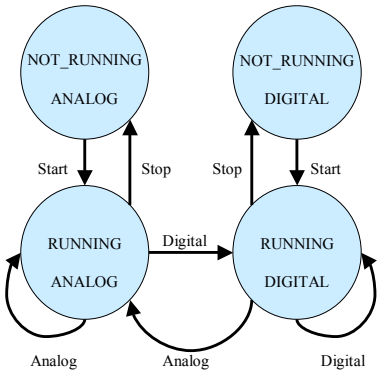
Operational Modes in The Clock

The System is either

- RUNNING or
- NOT_RUNNING.

The Setting is either

- ANALOG or
- DIGITAL.



All Actions Aren't Always Available

Clock = NOT RUNNING

Action = Stop

Rule: You can't execute the Stop action if the Clock is not running

Deriving Rules from Operational Modes

Stop

- When the System is NOT_RUNNING, the user cannot execute the **Stop** action.
- When the System is RUNNING, the user can execute the **Stop** action.
- After the **Stop** action executes, the System is NOT_RUNNING.

Using VT Code to Build the Model

```
possible = TRUE                                  ‘ assume the action is possible
if (action = “Stop” ) then                       ‘ want to do a Stop action?
    if (system_mode = RUNNING) then              ‘ if clock is in running mode
        new_system_mode = NOT_RUNNING            ‘ clock goes to not running mode
    else                                         ‘ otherwise
        possible = FALSE                         ‘ Stop action is not possible
    endif
endif
if (possible = TRUE) then                        ‘ if action is possible
    print system_mode;”.”;setting_mode,          ‘ print beginning state
    print action,                                ‘ print the test action
    print new_system_mode;”.”;new_setting_mode   ‘ print ending state
endif
```

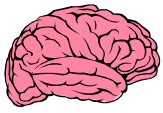
The Generated Finite State Table

Beginning State	Action	Ending State
NOT_RUNNING.ANALOG	Start	RUNNING.ANALOG
NOT_RUNNING.DIGITAL	Start	RUNNING.DIGITAL
RUNNING.ANALOG	Stop	NOT_RUNNING.ANALOG
RUNNING.DIGITAL	Stop	NOT_RUNNING.DIGITAL
RUNNING.ANALOG	Analog	RUNNING.ANALOG
RUNNING.ANALOG	Digital	RUNNING.DIGITAL
RUNNING.DIGITAL	Analog	RUNNING.ANALOG
RUNNING.DIGITAL	Digital	RUNNING.DIGITAL

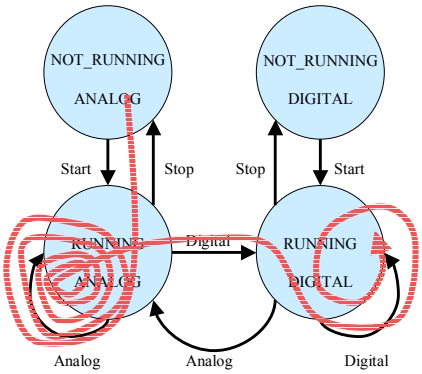
Step 2:

Generate interesting test actions

A Random Walk

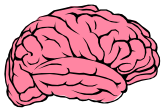


Start
Analog
Analog
Analog
Analog
Analog
Analog
Digital
Digital
…

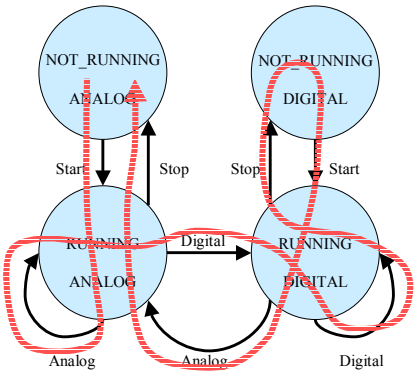


re-invent the monkey

Chinese Postman

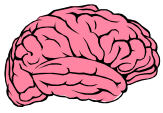


Start
Analog
Digital
Digital
Stop
Start
Analog
Stop

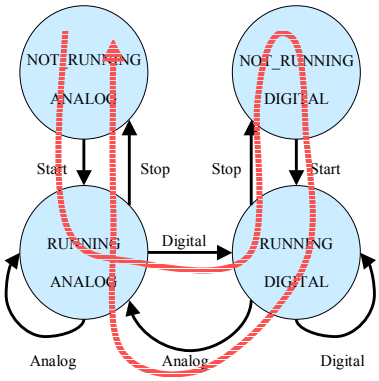


execute every action

State-Changing Postman

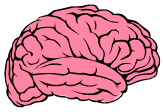


Start
Digital
Stop
Start
Analog
Stop



execute every state-changing action

Shortest Paths First

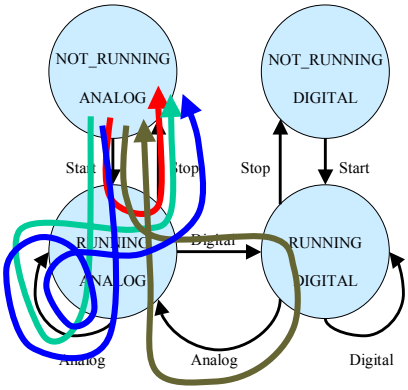


Length = 2
Start Stop

Length = 3
Start Analog Stop

Length = 4
Start Analog Analog Stop
Start Digital Analog Stop

and so on …



execute every path (eventually!)

Step 3:

Execute the test actions

Visual Test functions

`Run("C:\WINNT\System32\clock.exe")`	Starts the Clock application
`WMenuSelect( "Settings\Analog")`	Chooses the menu item "Analog" on the "Settings" menu
`WSysMenu( 0 )`	Brings up the System menu for the active window
`WFndWnd("Clock")`	Finds an application window with the caption "Clock"
`WMenuChecked("Settings\Analog")`	Returns TRUE if menu item "Analog" is check-marked
`GetText(0)`	Returns the window title of the active window

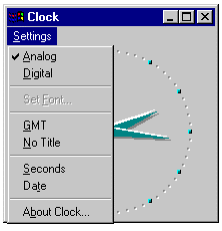
Executing the Test Actions

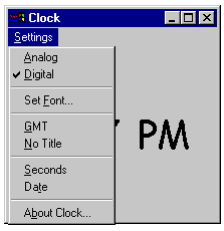
```
open "test_sequence.txt" for input as #infile     'get the list of test actions

while not (EOF(infile))

    line input #infile, action                    'read in a test action

    select case action

    case "Start"                                  ' Start the Clock
        run("C:\WINNT\System32\clock.exe")        ' VT call to start clock

    case "Analog"                                 ' choose Analog mode
        WMenuSelect("Settings\Analog")            ' VT call to select Analog

    case "Digital"                                ' choose Digital mode
        WMenuSelect("Settings\Digital")           ' VT call to select Digital

    case "Stop"                                   ' Stop the Clock
        WSysMenu (0)                              ' VT call to bring up system menu
        WMenuSelect ("Close")                     ' VT call to select Close

    end select

wend
```

Step 4:

Determine if the application worked

Use Rules as Heuristic Test Oracles





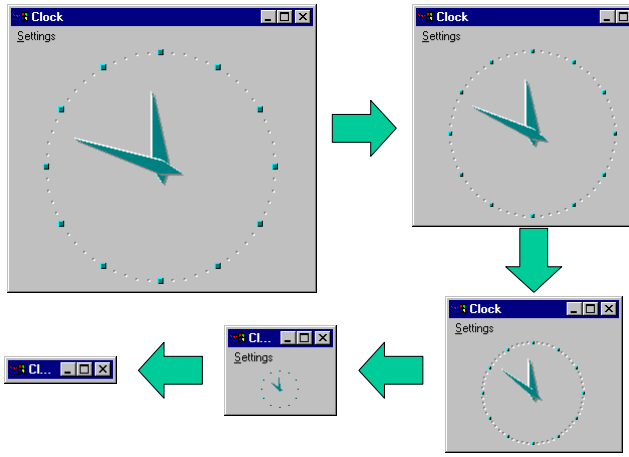

```
if ( (setting_mode = ANALOG) _                    'if we are in Analog mode
AND NOT WMenuChecked("Settings\Analog") ) then    'but Analog is not check-marked
        print "Error: Clock should be Analog mode"    'alert the tester
        stop
endif
```

Step 5:

Find bugs

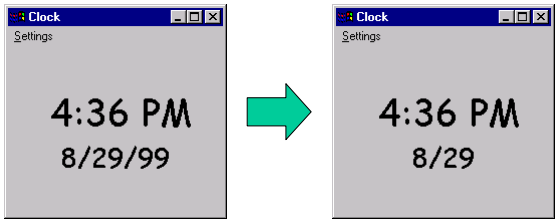
The Incredible Shrinking Clock

Start
Maximize
Stop
Start
Minimize
Stop
Start
Restore
Stop



Where Have the Years Gone?

Start
Minimize
Stop
Start
Restore
Date



Actions in the Demo Model

- Invoke
- Analog
- Digital
- Set_font
- GMT
- No_title
- Seconds
- Date
- About
- Close_clock
- Ok_about
- First_font
- Random_font
- Last_font
- Ok_font
- Cancel_font
- Double-click
- Minimize
- Restore

Chinese Postman

invoke seconds restore no_title double-click minimize restore maximize minimize restore seconds restore GMT double-click double-click digital set_font random_font last_font first_font cancel_font set_font ok_font seconds restore no_title double-click maximize set_font random_font last_font first_font cancel_font minimize restore set_font ok_font seconds restore GMT double-click double-click digital date close_clock

invoke analog date analog about ok_about close_clock

Shortest Paths First

invoke close_clock

invoke analog close_clock
invoke date close_clock
invoke GMT close_clock

... and so on ...

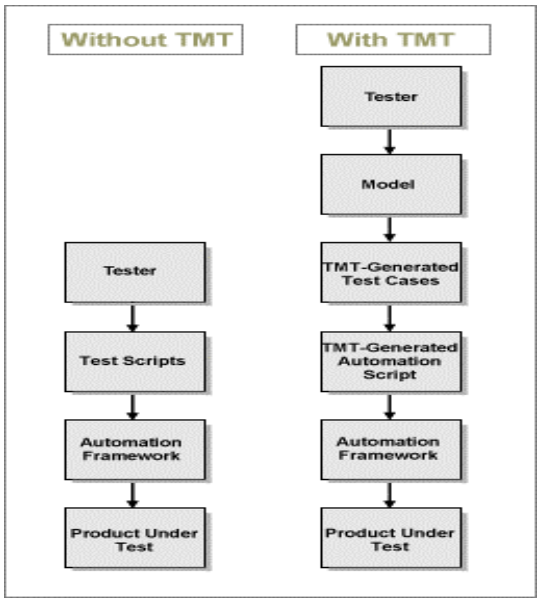
invoke seconds restore close_clock
invoke seconds seconds close_clock

Model-Based Testing

- Programmatic
- Efficient coverage
- Tests what you expect and what you don't
- Finds crashing and non-crashing bugs
- Significant investment in tested app
- Resistant to pesticide paradox
- Still fun to watch

Models Find Bugs in Specs

- Detailed Models find missing transitions, undefined states and actions.
- Scenario / high level models find missing features –what would a user want?
- Complain to PMs about issues early, when they can be fixed.



Why Model-Based Testing?

- Improve specs
- More nimble test automation
- Earlier test automation
- Better relationship with Dev/PM, working side by side, rather than against.
- Attract and retain high quality testers

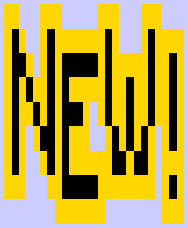
For more info …

www.model-based-testing.org

or

harryr@microsoft.com
jamesti@microsoft.com

Thanks!



QW2001 Paper 2A2

Mr. Klaus Olsen
(Softwaretest.dk)

Using The W Model To Institutionalize Inspections, And Improve Knowledge Transfers

Key Points

- We want to build quality into the system
- Defect prevention instead of defect detection
- Knowledge transfer

Presentation Abstract

This paper introduces the W-model as a development model. The W-model makes inspection and review activities visible during the development lifecycle. In the W-model test planning starts as early as possible, and through the different types of review meetings knowledge transfer between analysis, programmers and testers are build in as an integrated part of the development process. All the knowledge collected during work with business representative analysing the requirement is often hard to transfer to the group of people programming and testing the application. The W-model suggests a work method that solves this problem.

About the Author

Klaus Olsen has created his own company in April 2000 “Softwaretest.dk” in order to focus entirely on software testing, and process improvement through the testing perspective of developing software. Klaus Olsen has worked with developing software for 15 years, during 12 of these years Klaus worked as a consultant in Cap Gemini. Klaus has specialised in software testing since 1993 and he was until he created his own company responsible for introducing new employees to Cap Gemini in the Nordic area (Europe) to “Working Methods Test”, as Cap Gemini were using it. Klaus has also been involved in improving Cap Gemini best practice in Software Testing, this is documented in the company’s PERFORM Testing Guide, available to all of the companies more than 55.000 employees. Klaus is member of two Danish special interest groups in software testing, as well as he is a member of Swedish Association of Software Testers, SAST.

Softwaretest.dk

Using the W-model to Institutionalise Inspections and Improve Knowledge Transfer

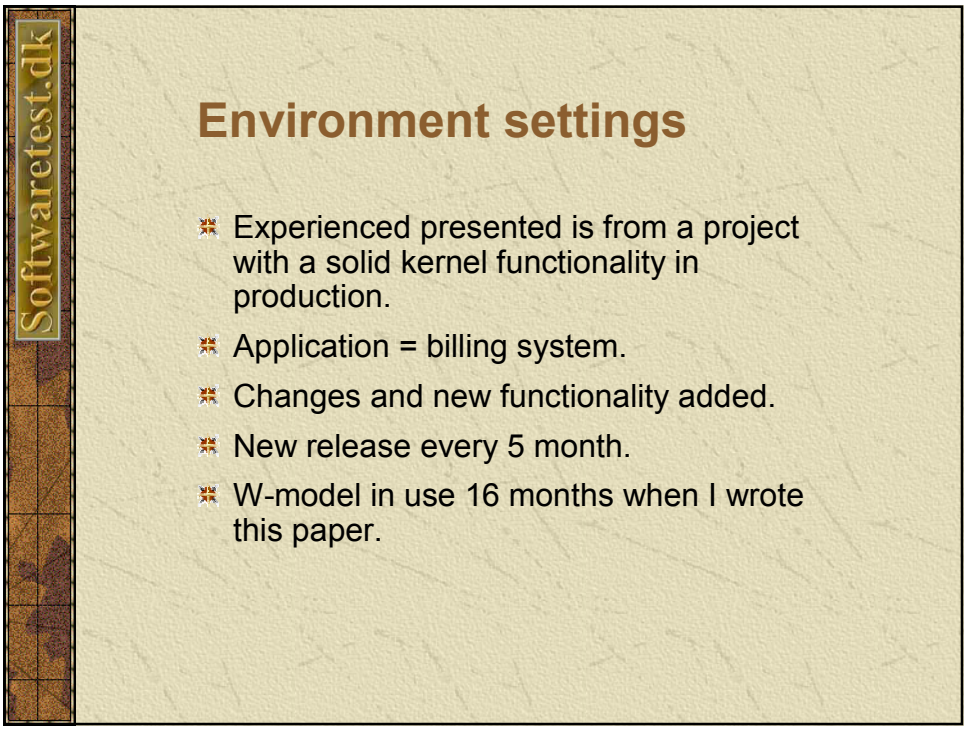
Presented at the 14th Annual
International Internet & Software Quality Week 2001
San Francisco, California, USA

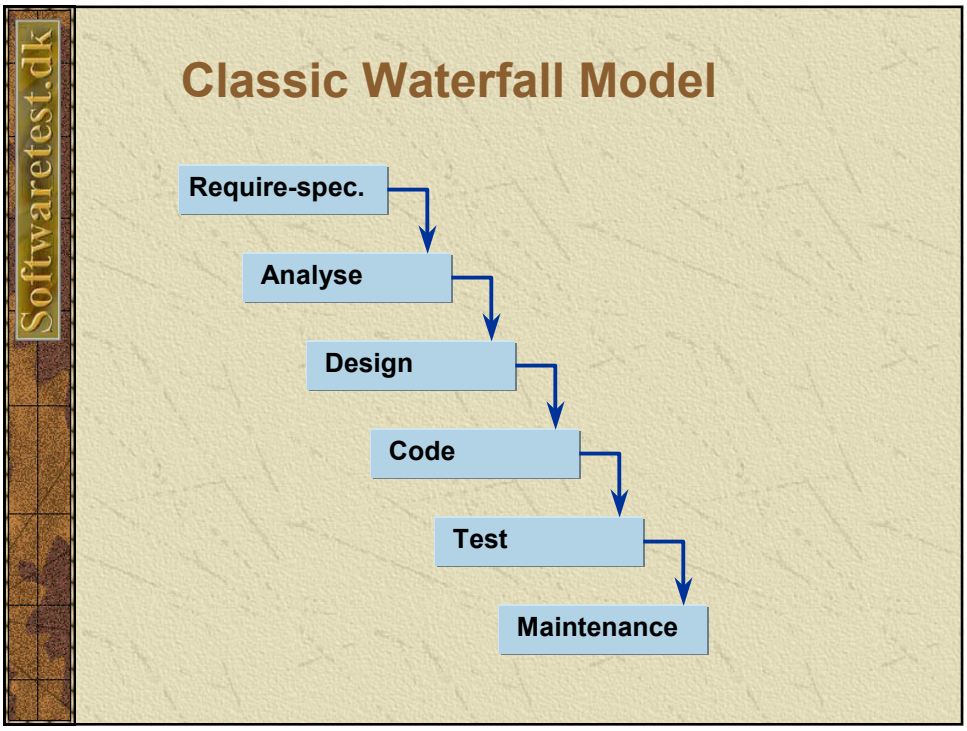
by
Klaus Olsen, Test Adviser
Softwaretest.dk - Denmark

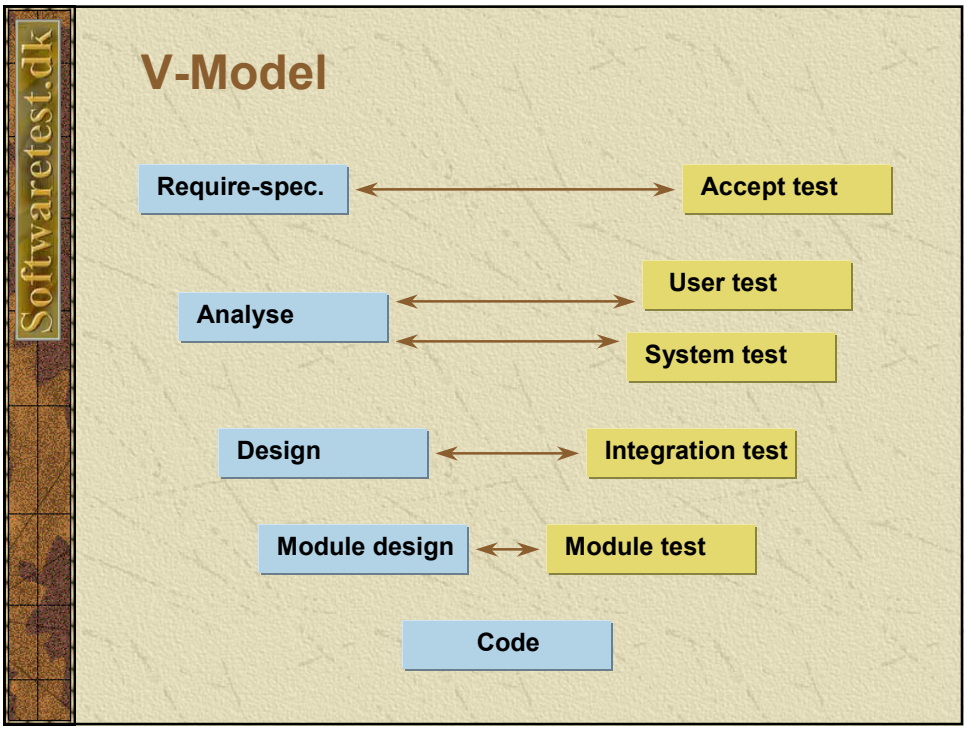
Softwaretest.dk

Who is Klaus Olsen

- I have worked with IT development projects since 1987.
- I have focused on test of software since 1993.
- I have been involved in integration of test tool both as part of regular testing and as a regression test facility.
- I have worked with test process improvement and more broad software process improvement.
- I have participated in improving test methods on a Danish, Scandinavian and Global level in Cap Gemini during a period of 3 years. (1997-2000)
- I have created my own company Softwaretest.dk focusing on all aspect of software testing as a Test Adviser.





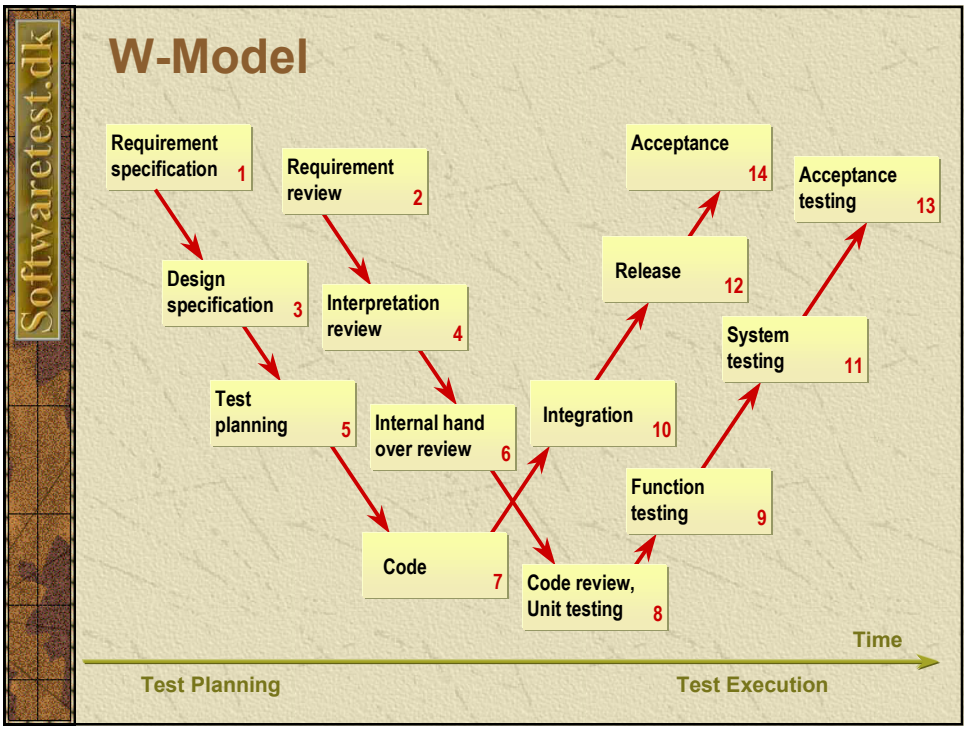


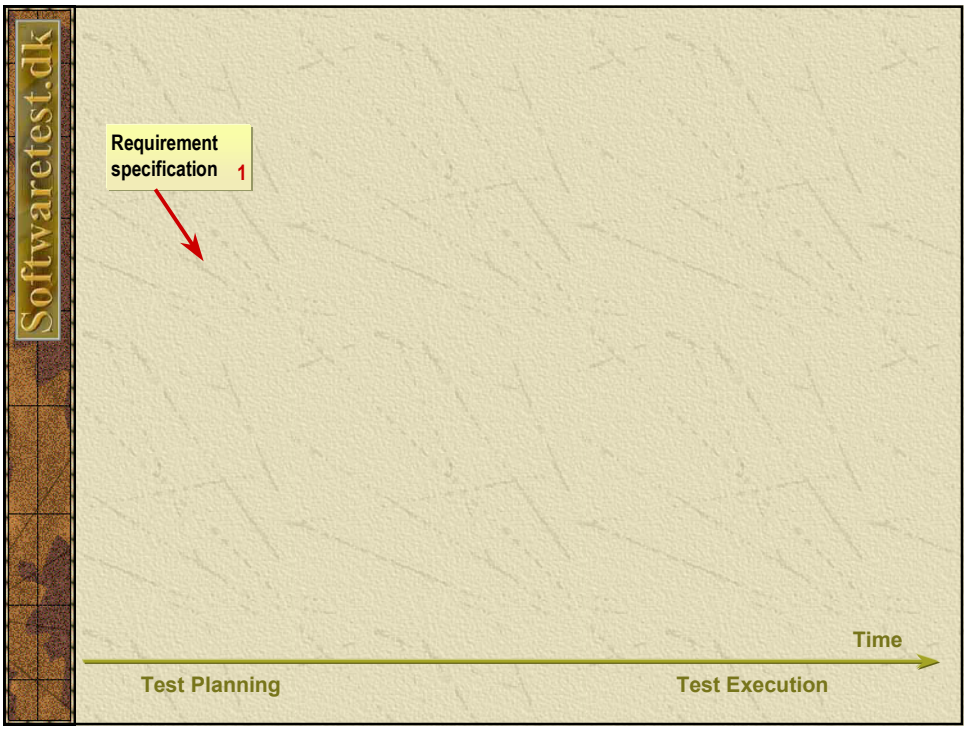
Software Development Phases	Errors introduced	Errors observed
Requirement analysis	55 %	5 %
Design	30 %	10 %
Construction and System Test	15 %	40 %
Acc. Test and operation/maintenance	0 %	45 %

Sources:
Boehm, Barry W Software Engineering Economics
Engelewood Cliffs, N.J: Prentice Hall, Hughes
DOD composite Software Error History

Main objectives

- We wanted to build quality into the system
- We were trying to accomplish **defect prevention** instead of **defect detection**
- Knowledge transfer between team members



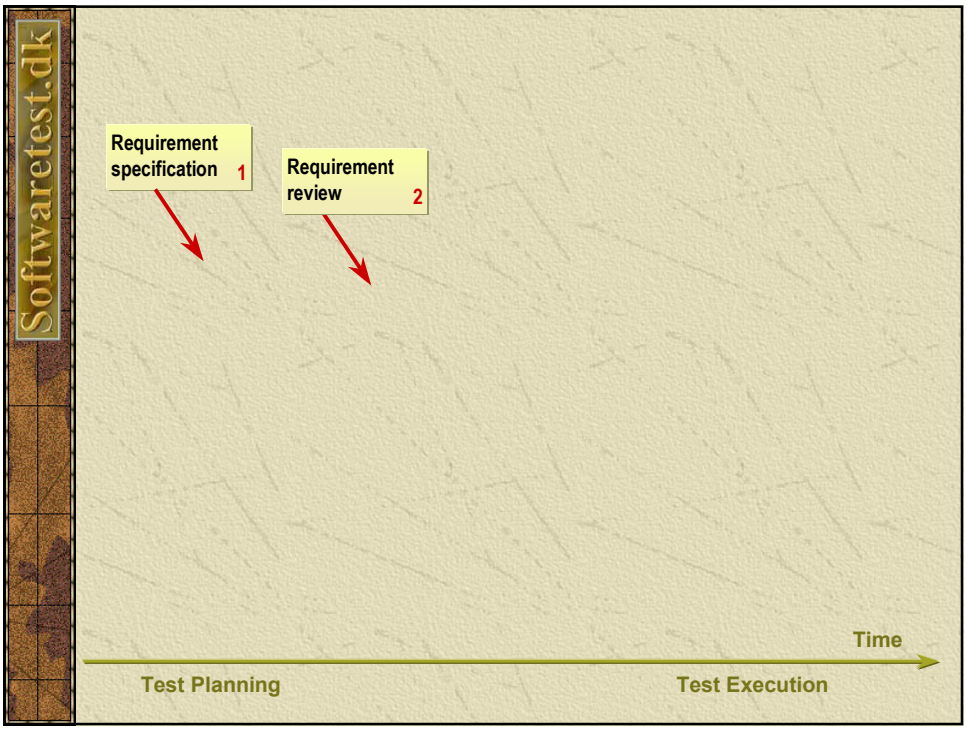


Requirement Specification

- Result of this process is a Technical Enhancement Specification (TES) document

As support to facilitate the work:

- TES Template
- TES writing guideline



Requirement Review - 1

- Early in process, a “brainstorm” type of meeting
  - The objective is to come up with all possible solutions, and to have focus on selecting the best and most feasible solution.

As support to facilitate the work:
- Checklist before calling the meeting
- Suggested agenda for the meeting

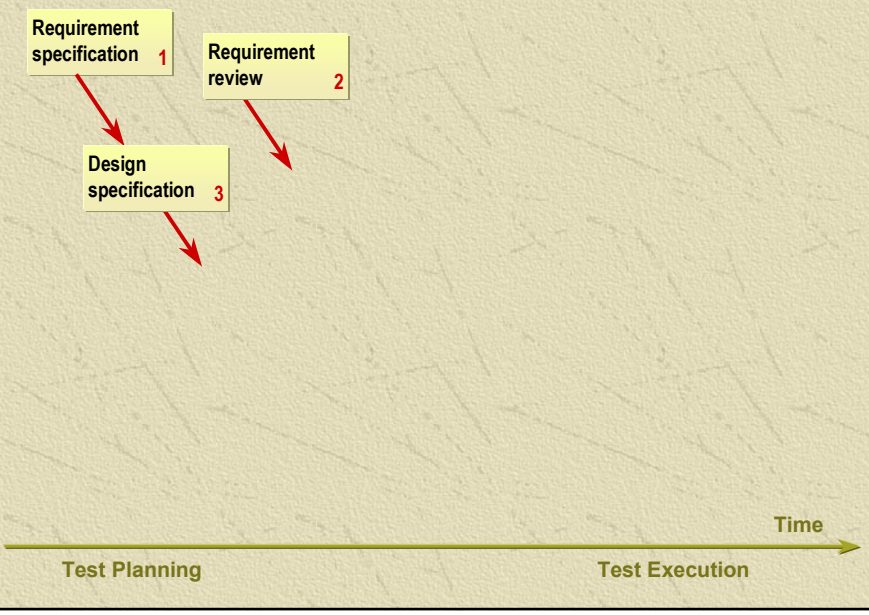
Requirement Review - 2

- Late in process, a Prebaseline review meeting
  - Focus is to ensure quality before we deliver the TES-document for Quality review with following baseline of document at the customer office

As support to facilitate the work:
- Checklist before calling the meeting
- Suggested agenda for the meeting

Requirement Review - 3

- Formal Quality review with customer
  - Focus is to ensure the right quality as viewed by Business before baseline of document.

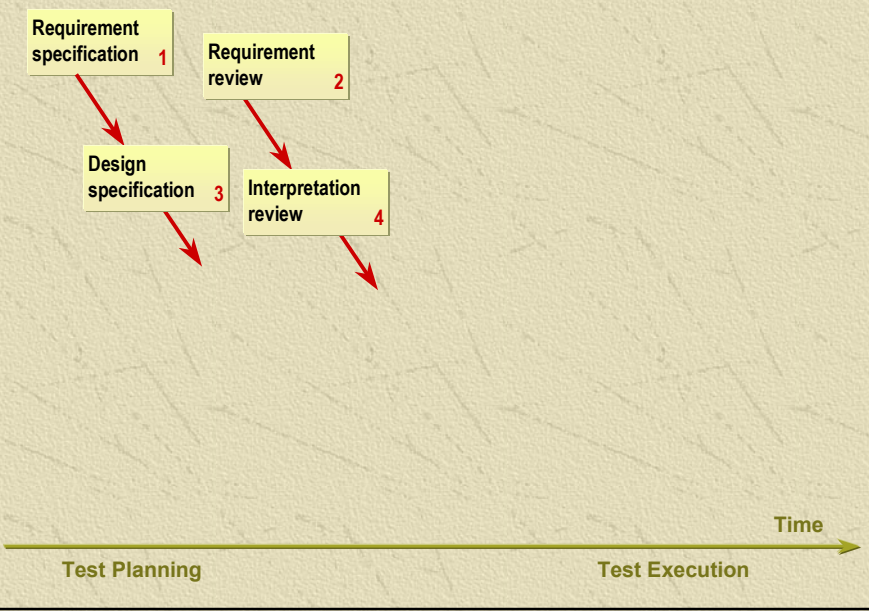


Design specification

- Site-project leader has to decide if Technical Enhancement Design (TED) is mandatory.
  - If TED is not mandatory the developer may decide to prepare a TED or a less formal work document as preparation for the Interpretation review.

As support to facilitate the work:

- TED Template
- TED writing guideline

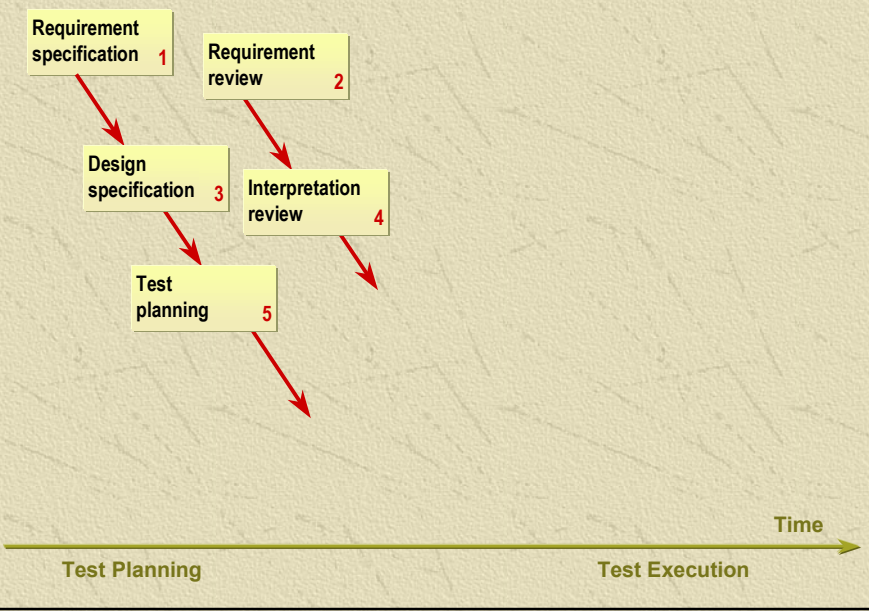


Interpretation Review

- Developer(s) assigned to this TES / TED explains how he/she understands the contents of the TES / TED.
- The TES / TED author and the Site Project Leader must confirm or correct the interpretation.
- To achieve knowledge hand over there must be a tester present at the meeting.

As support to facilitate the work:

- Checklist before calling the meeting
- Suggested agenda for the meeting

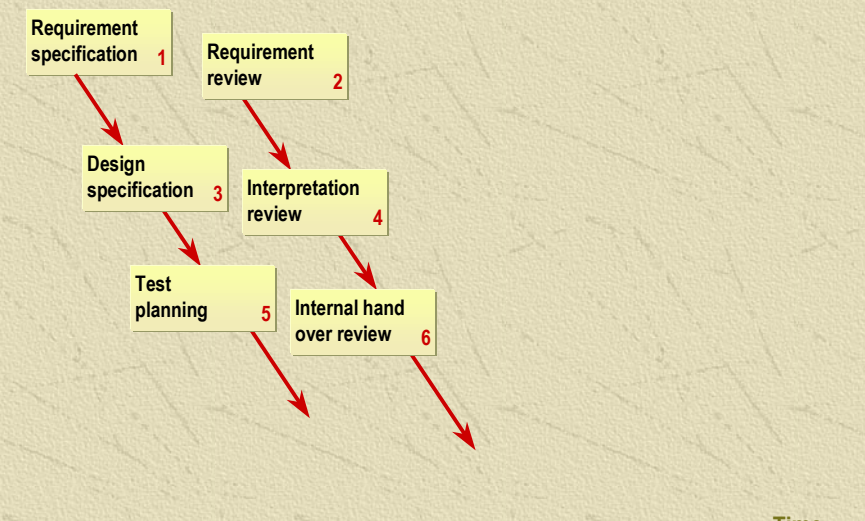


Test Planning

- Result of this process is a Test Specification Plan (TSP), which is preparation for the Internal hand over review meeting.

As support to facilitate the work:

- TSP Template
- TSP Checklist
- Checklist for test of online screens

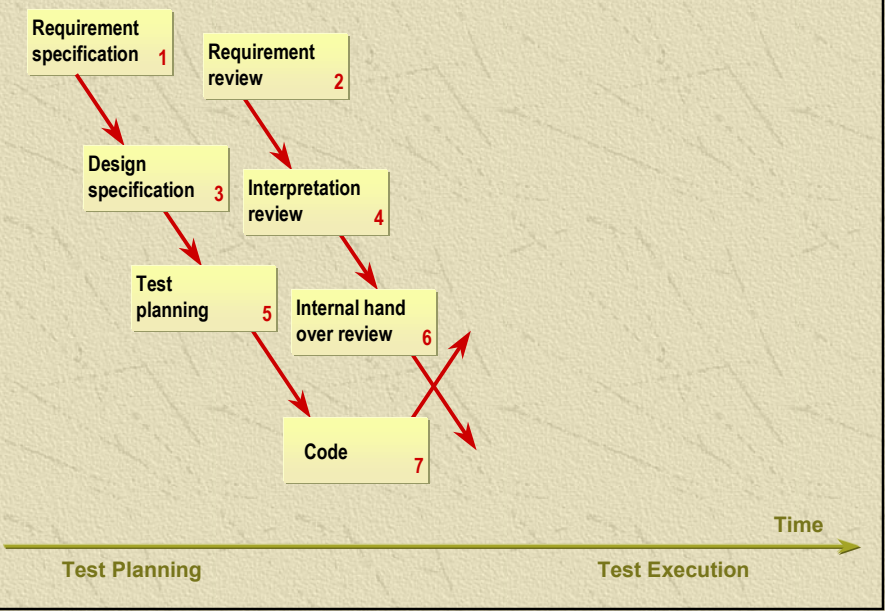


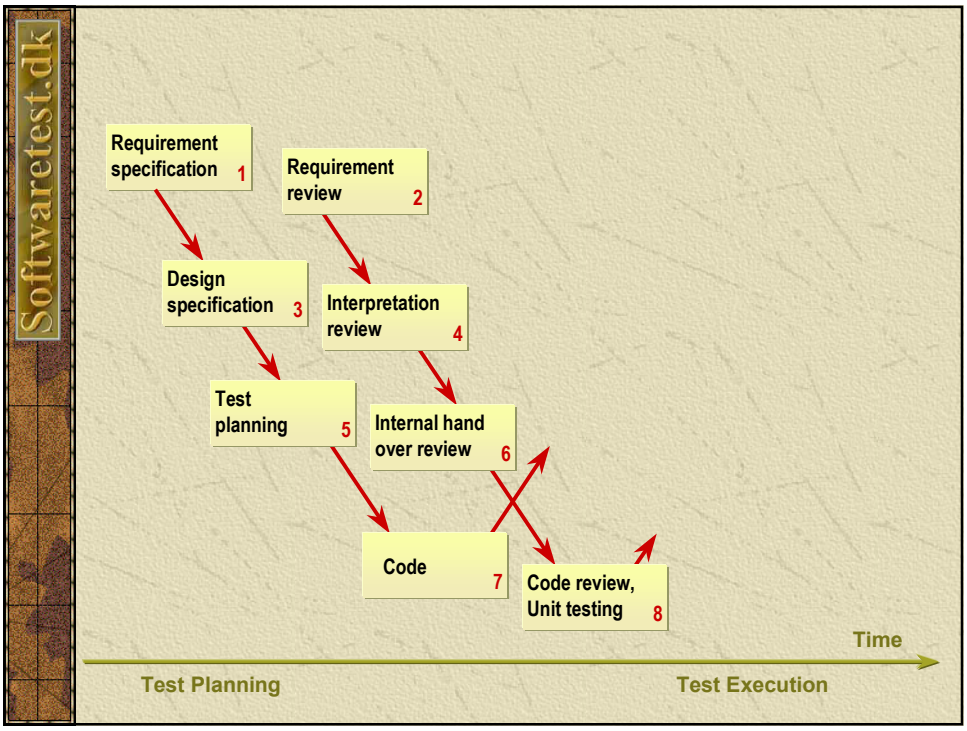
Internal handover

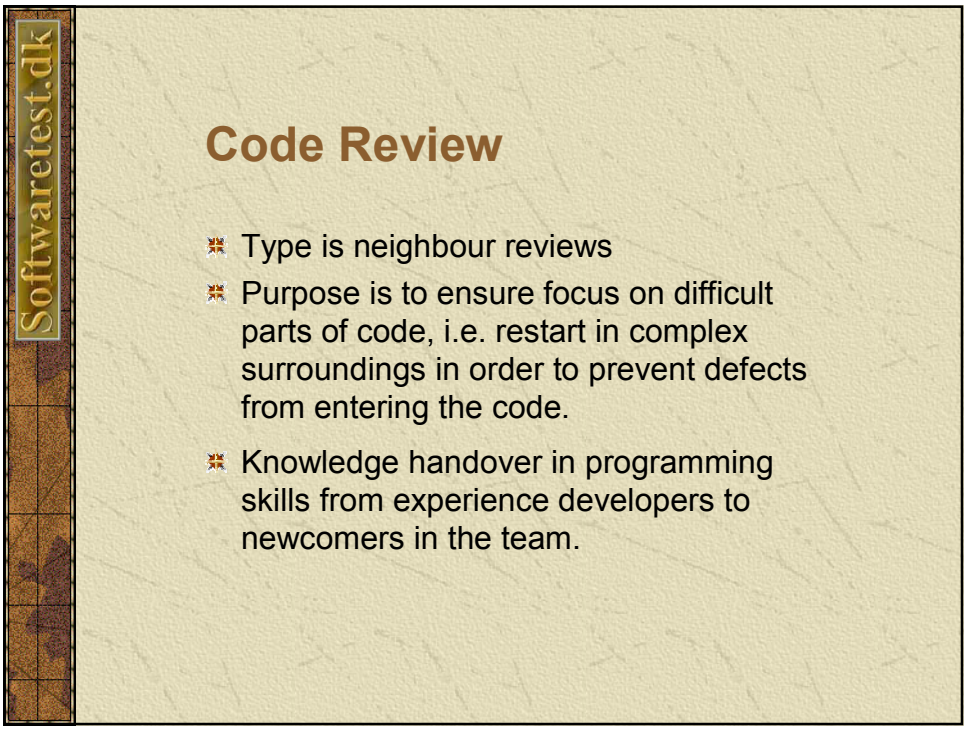
- Developer explains what and how the TES / TED has been implemented.
- Tester makes a walkthrough of the TSP and explains the testing approach.
  - Objective is to improve the test planning quality and ensure focus on right part of area under test.

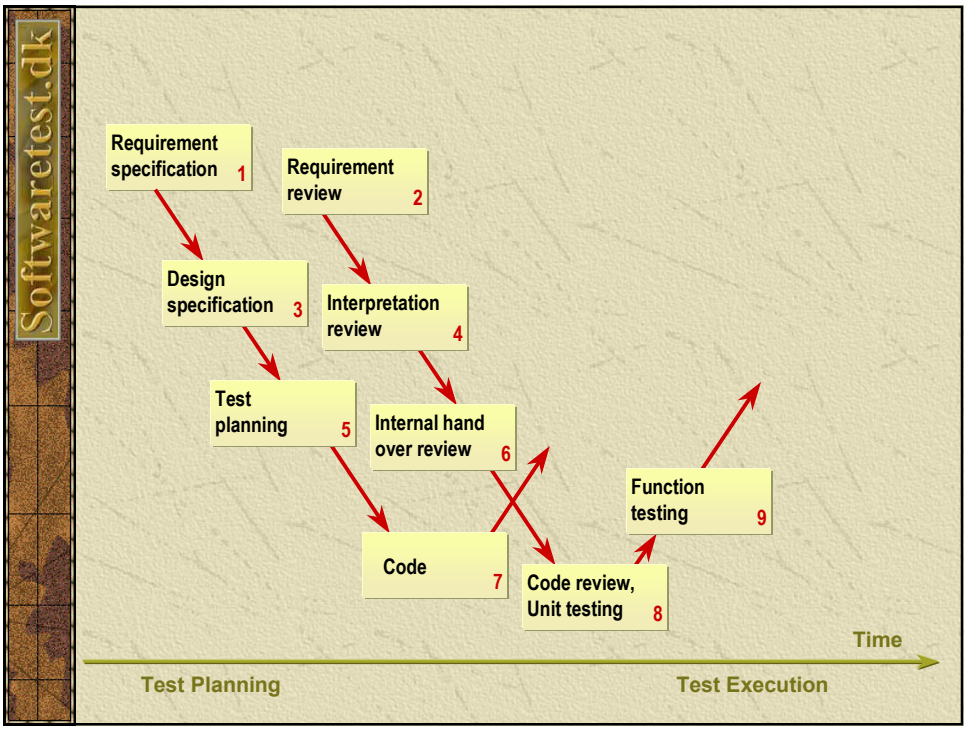
As support to facilitate the work:

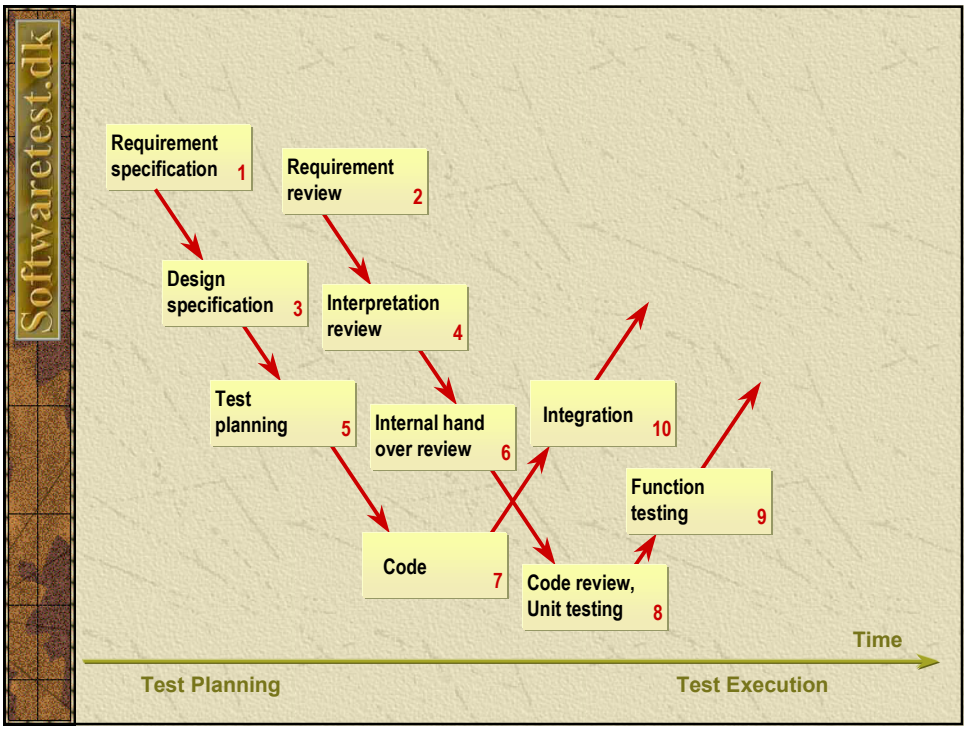
- Checklist before calling the meeting
- Suggested agenda for the meeting

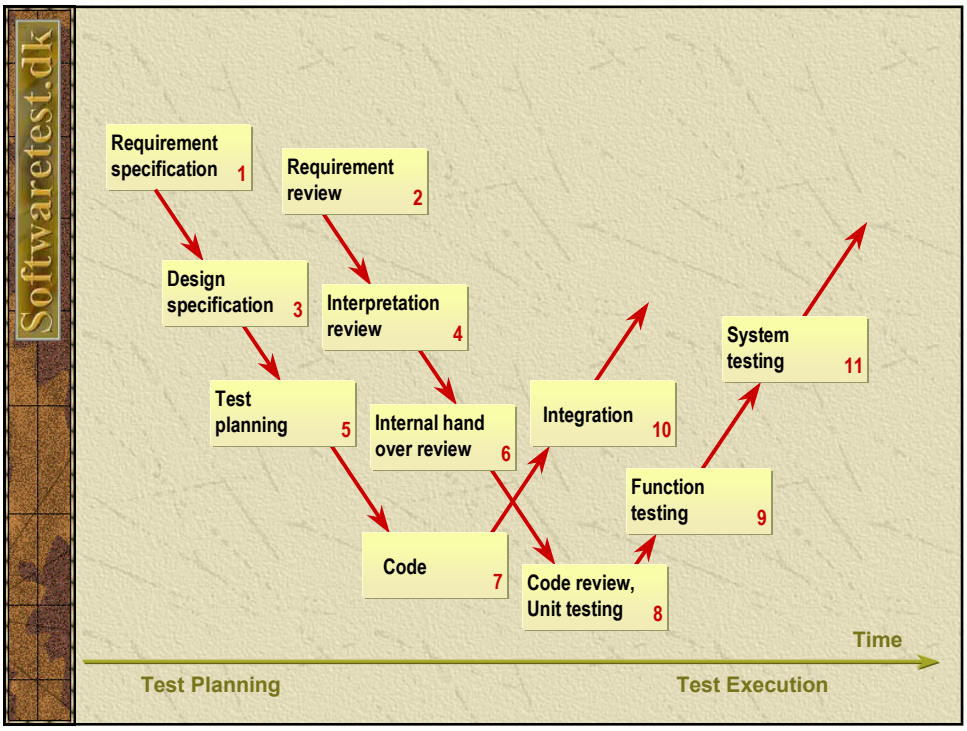


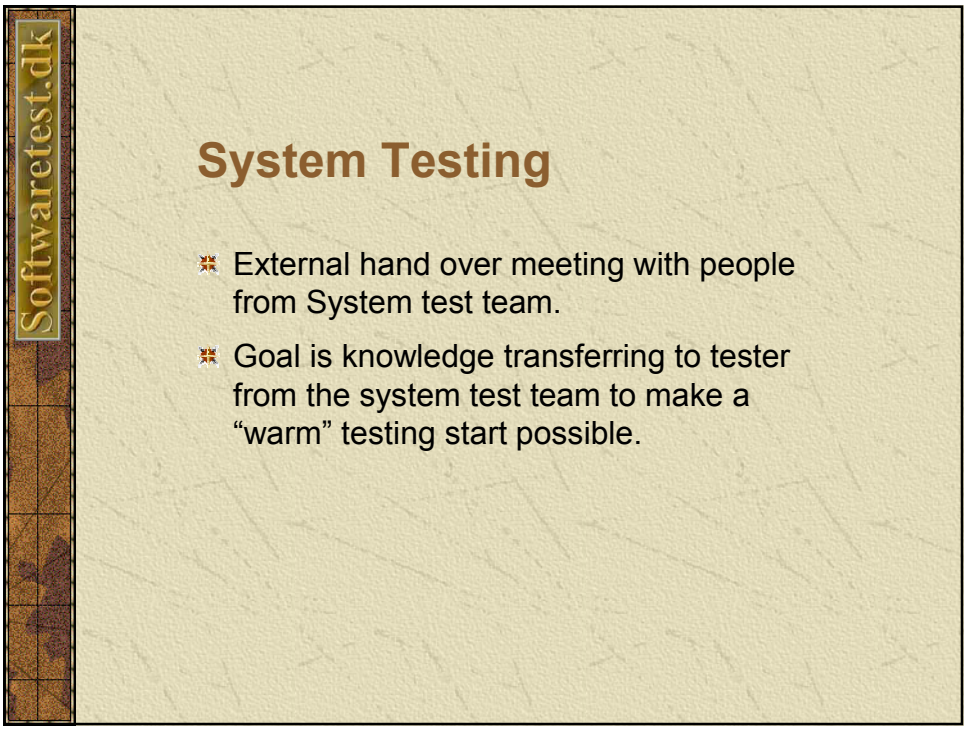


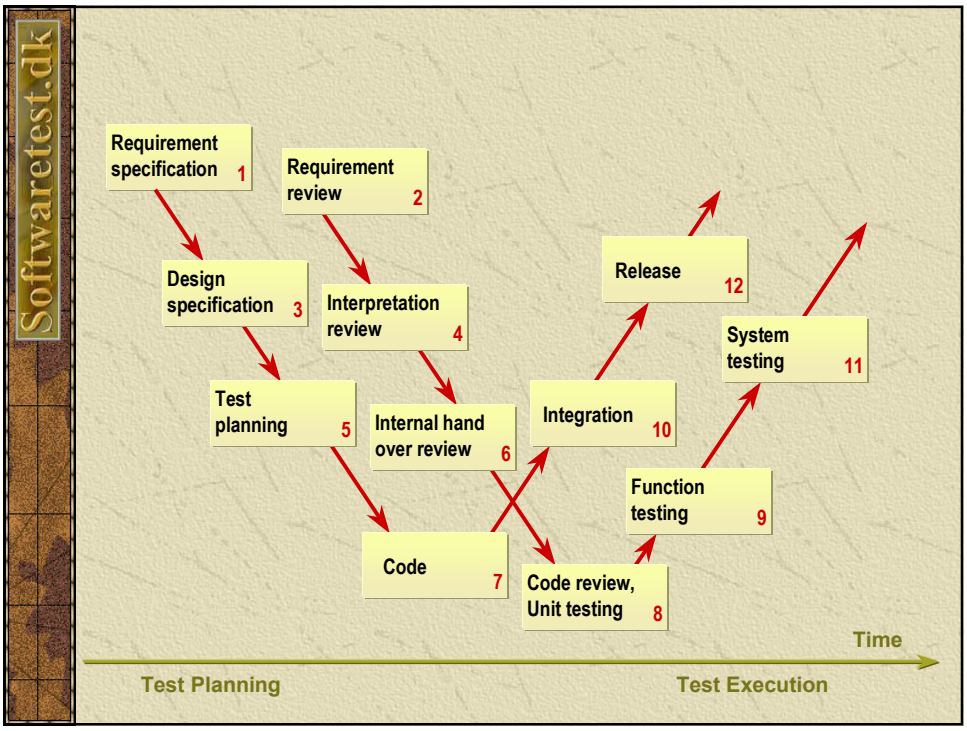


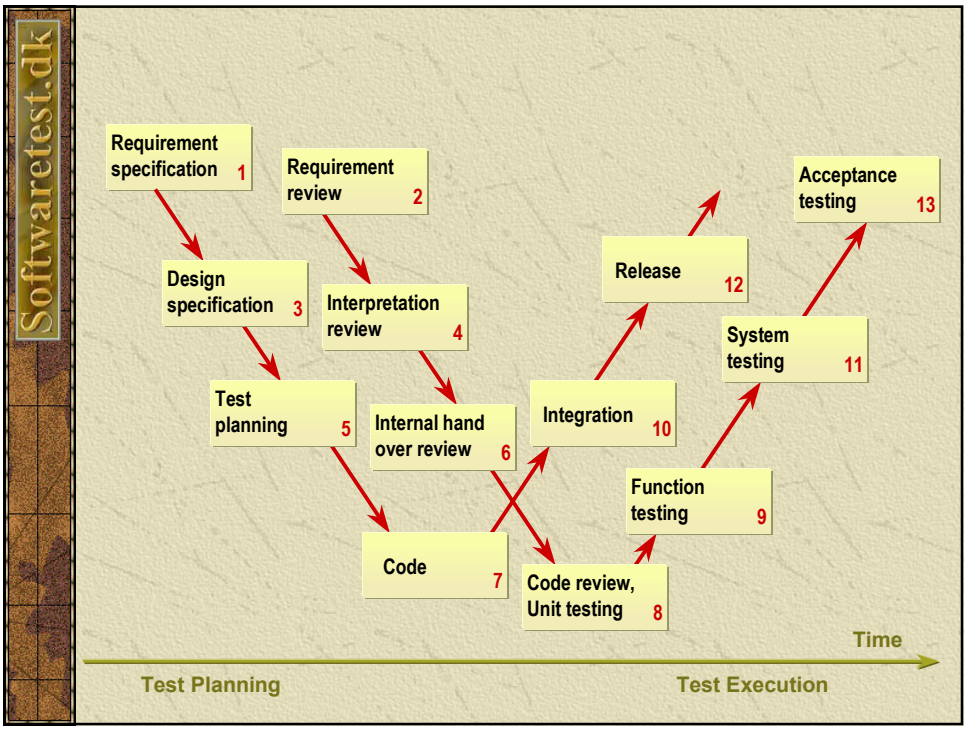


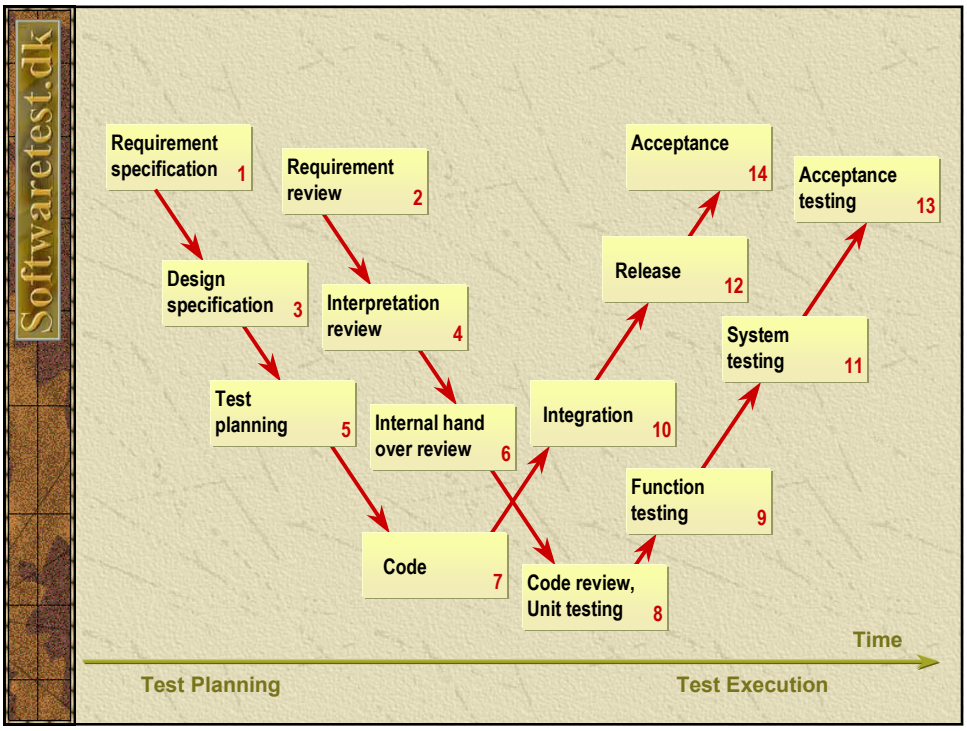


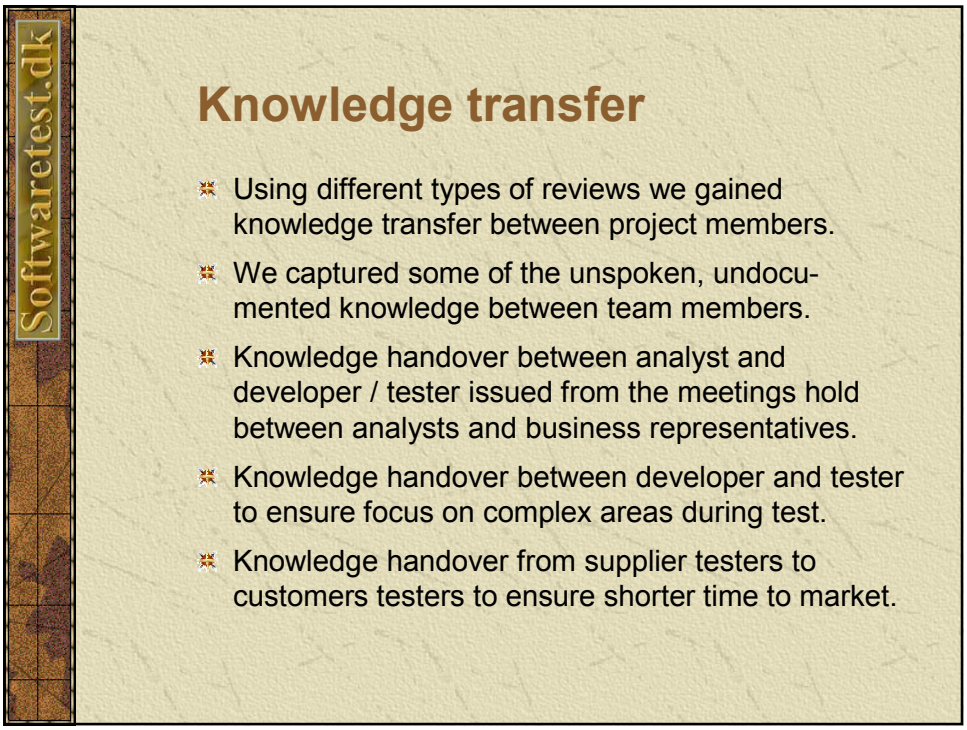












Conclusion

- *Using the W-model we have succeeded in changing the development model to more focus on review and early test planning.*
- *All project members takes an active part in knowledge transfer.*
- *Defect prevention and early defect detection is being recognized as the right way to build quality into the system.*
- *Different types and levels of Inspection have been institutionalised by the W-model.*
- *SPI works, but it requires continually follow-up on the use of processes.*

Contact information:

www.softwaretest.dk

Klaus.Olsen@softwaretest.dk

Using the W-model to Institutionalise Inspections and Improve Knowledge Transfer

Presented at the 14th Annual International Internet & Software Quality Week 2001
San Francisco, California, USA by

Klaus Olsen, Test Adviser. Softwaretest.dk - Denmark
klaus@softwaretest.dk web-site: www.softwaretest.dk

Summary:

This paper introduces the W-model as a development model. The W-model makes inspection and review activities visible during the development lifecycle. In the W-model test planning starts as early as possible, and through the different types of review meetings knowledge transfer between analysis, programmers and testers are build in as an integrated part of the development process. All the knowledge collected during work with business representative analysing the requirement is often hard to transfer to the group of people programming and testing the application. The W-model suggests a work method that solves this problem.

1. Introduction

Traditional software development often uses the waterfall model, in which each phase is finalised, before the next starts. An extension of this model is the V-model, which is traditionally passed through once to complete a development cycle. The benefit of the V-model is the parallel preparation of test specification (test plans) and development within each development stage. The W-model indicates yet another pass through: first the same steps as the V-model are processed, followed by a shadow V that provides continuous product review to support the development activities. What we have tried to accomplish is the W-model. We wanted to start test planning earlier and at the same time we wanted to institutionalise Inspection and improve knowledge transfer between team members.

2. The environment

In order to improve the knowledge transfer to readers of this paper I have included information on the set-up surrounding the project. This experience is gathered from a project, which had a solid kernel of functionality running in daily production. Changes and new functionality were added to this existing kernel. New releases were delivered to the customer every 5th month, and when this paper was written, the development process described in the W-model had been in use for 16 months, with our delivery cycle, this implies that we had been through the complete lifecycle of the W-model 3 times. Even though we didn't look at the project as an iterative project, we in fact used the W-model as an iterative development model.

3. Introducing the W-model

The decision to use the W-model [/1/ Burr, Owen] as the development process in the project, was an attempt to improve the quality of the product and the work process. In order to improve the quality of the product we wanted to formalise Inspections, which until that point had been recommended but was

optional. In this paper Inspection is used as a quality improvement process for written material as defined by [/2/ Gilb - Graham]. Project management agreed on enforcing Inspections throughout the development with different goals at different stages. The W-model is useful to make this focus on Inspection visible, because the shadow V provides continuous product reviews to support the development activities.

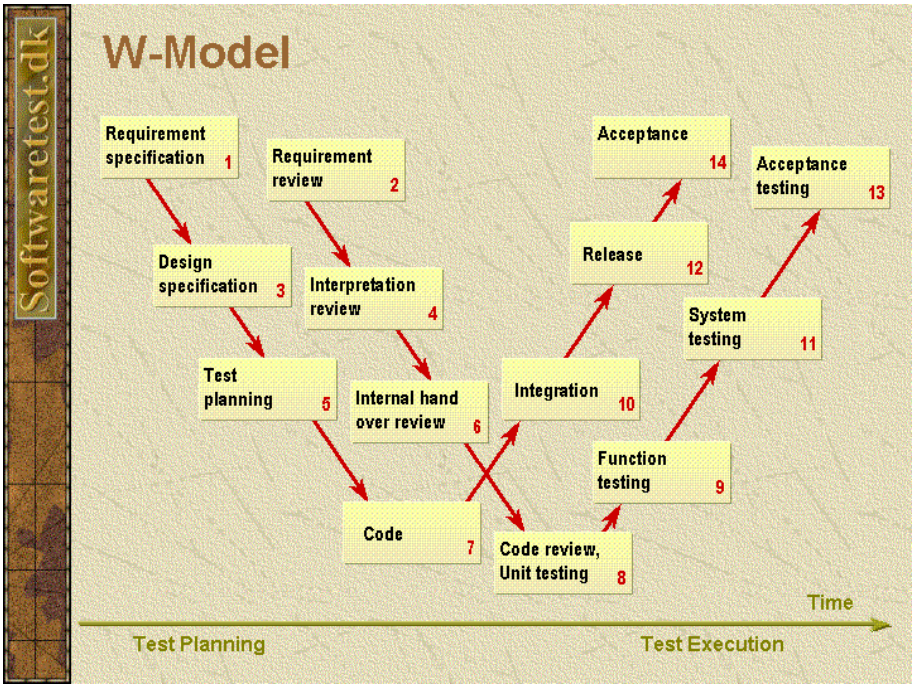


Figure 1, W-model as used in this project

By introducing different focus areas in the different stages, "quality" review, interpretation review and internal- and external-handover review meetings we have succeeded in transferring knowledge between team members working on analysis, development and testing. We have improved the process by developing templates, writing guidelines and checklists to be used in the different stages.

3.1 The W-model in details

To each phase visible in Figure 1 the process being carried out is described with reference to documents developed and available to the project in order to facilitate the work. The following section describes the W-model as it has been implemented. Number refers to different parts of the processes of the W-model above. E.g. 3.1.1 = process 1 in the W-model

3.1.1. Requirements specification

The result of the requirement process is a Technical Enhancement Specification (TES) -document. Two documents are available to facilitate the work, TES template and a TES writing guide.

3.1.2. Requirements review

The reviews actually consists of three meetings, the first two internal in the development project, the third external quality review at the customer's office with users, followed by a formal baseline of the document.

- Meeting 1 - Early in process, a "brainstorm" type of meeting.

  The objective is to suggest all possible solutions, and to focus on selecting the best and most feasible solution. The TES-author is responsible for calling the review meeting. Two documents are

available to facilitate the work, checklist before calling the meeting, and a proposed agenda for the meeting.

Participants in this first meeting should be persons who can add something to the issue under review. A suggested list of participants is included in the project, but left out of this paper. The TES-author is responsible for updating the TES after review meeting.

- Meeting 2 - Late in process, a pre-baseline review meeting.

  Focus of this review is to ensure quality before the TES-document is delivered for quality review as the last step before the document is baseline of document at the customer's office. The TES-author is responsible for calling the review meeting. Two documents are available to facilitate the work, checklist before calling the meeting, and a proposed agenda for the meeting.

  A review only using mail could be a solution to this review, but having a formal meeting creates more synergy between participants of the review and tends to result in better quality of the TES-document before baseline. Participants in this second review should be people who either participated in the first and/or people working with documentation. The TES-author is responsible for updating the TES after review meeting.

- Meeting 3 - Formal quality review with the customer.

  Participants in this review should be people who have participated in the analysis meeting with the customer and the TES author participates from development side.
  Following the quality review meeting there is a formal baseline procedure as described in the TES writing guide.

3.1.3. Design specification

Site-project leader has to decide if a Technical Enhancement Design (TED) document is mandatory. If a TED is not mandatory the developer may decide to prepare a TED or a less formal work document as preparation for the Interpretation review (described in the paragraph below). Two documents are available to facilitate this work, a TED template and a TED writing guide.

3.1.4. Interpretation review

These are internal development meetings, where the developer(s) assigned to this TES/TED explains how he/she understands the contents of the TES/TED. The TES/TED author and the project manager must confirm or correct the interpretation. To achieve knowledge handover a tester must be present at the meeting.

The project manager is responsible for calling the review meeting. Three documents are available to facilitate the work, a checklist before calling the meeting, a proposed agenda for the meeting and as a prerequisite either a TED or a work document created by the developer must be available. The responsibility for updating the TED, and if necessary the TES, must be nominated at the meeting.

3.1.5. Test planning

The result is a function test plan, which is a preparation for the internal hand over review meeting. Three documents are available to facilitate the work, first a Test Specification Plan (TSP) template, then a TSP checklist and finally a checklist to be used when executing test of online screens.

3.1.6. Internal hand over review meeting

This is an internal development meeting. Developer explains what and how the TES / TED has been implemented. This includes mentioning of special areas where another solution than the one specified in the TES has been used, and areas of complexity that the tester needs to pay special attention to. The developer must also explain what has been tested in component test.

The tester makes a walkthrough of the TSP and explains the testing approach. The objective is to improve the quality of test planning and ensure focus on the most important areas during the test. The project manager is responsible for calling the review meeting. Three documents are available to facilitate the work, a checklist before calling the meeting, a proposed agenda for the meeting and as a prerequisite: that implementation has taken place and that the tester has created a TSP document. If necessary, the responsible for updating the TES and TED will be nominated at the meeting. The TSP author is responsible for updating the TSP.

3.1.7. Coding

The development team performs this activity. One document is available to facilitate the work, a checklist to be used when executing test of online screens.

3.1.8. Code review and unit testing

The purpose is to ensure focus on difficult parts of code, e.g. restart in complex surroundings in order to prevent defects from entering the code. One document is available to facilitate the work, a proposed agenda for the meeting.

3.1.9. Function testing

This is performed in the test environment by the internal development test team. Documents created during test planning (see section 3.1.5) are used in conjunction with a document added after collecting best practice from all testers, a checklist to be used when executing test of online screens.

3.1.10. Integration released

Transfer from development to external test environments.

3.1.11. System testing

An external hand over meeting with persons from the external test team takes place. The objective is knowledge transfer to testers at the customer site in order to make a kick-start of the testing possible. The knowledge transfer is focused on new or changed functionality. The knowledge transfer includes comments on reference data needed in order to execute the test. If necessary (e.g. when external testers have limited application experience) each handover should explain the necessary parts of the surrounding system areas as well.

The external test manager is responsible for calling the meeting. Four documents have been created on to facilitate the work, a questions and answers template to facilitate external testers test preparation and testplanning process, as well as keeping track of knowledge gathering. A pre-meeting template used to communicate and clarify as much as possible before the meeting. A checklist to be used in the planning process by the development test team member. And a PowerPoint template including a proposed agenda for the external handover meeting.

3.1.12. Release

This includes a formal review to approve updates in the system description, or perhaps a new chapter in the system description, based on TES/TED documents. The objective is to keep the quality of the application system documentation as high as possible.

3.1.13. Acceptance testing

The customer's Accept Test Team performs this.

3.1.14. Acceptance

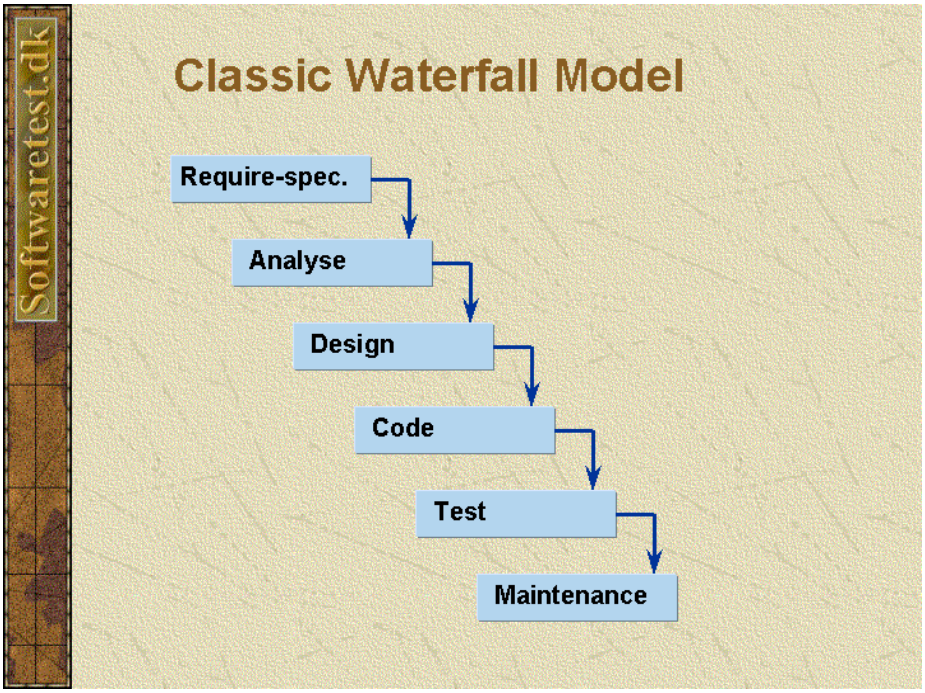
A go / no go meeting to decide if this product is ready to put into production.

4. Knowledge Transfer

A main reason for choosing the W-model as the development process in the project was improvement of knowledge transfer between project team members. Even though analysis and design authors try to document as much as possible, there will always be space for further interpretation. By using different types of reviews we gained knowledge transfer between the project members and at the same time captured some of the unspoken and undocumented knowledge between team members. This further expanded into knowledge handover between developer and testers. After developing an area, each programmer involved would explain what was implemented, and discrepancies from the original requirements or analysis if any. And as the last step we included an external knowledge handover between internal tester and external tester.

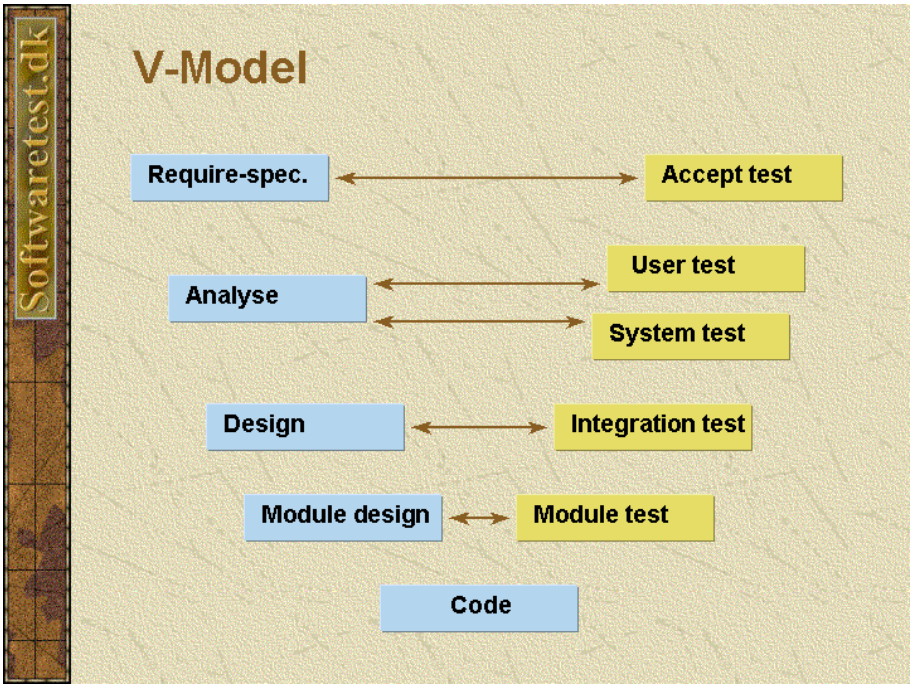
4.1 The Classic Waterfall model

When the project started we were using the Classic Waterfall model [/3/ Georgiadou], where each phase was completed before the next phase started. This model is visualised below:



4.2 The V-model

Gradually the project moved towards the V-model, as attention grew on test planning and test execution throughout the project. One version of the V-model is visualised below:



5. Conclusion

With the shift from the classic waterfall model through the V-model to the W-model we have succeed in shifting the visual view on the development model to a model with focus on review and early test planning. Equally important has been the shift for all project members to take an active part in knowledge transfer, and that defect prevention is starting to be recognized as the right way to build quality into the system. Quality is free [/4/ Crosby] compared to the cost of doing things wrong – and then having to redo and retest the corrections later. Different types and levels of Inspection have been institutionalised by the W-model, and internal education in Inspection has been executed thus changing the type and effectiveness of Inspection. Software Process Improvement works, but it takes time and demands continually follow-up on the use of processes by each team member, and still we find new areas that can be improved in our development process.

References

1. Burr, Adrian & Owen, Mal "Statistical Methods for Software Development: Using Metrics to Control Process and Product Quality", ISBN 1-85032-171-X
2. Gilb, Tom & Graham, Dorothy "Software Inspection", ISBN 0-201-63181-4
3. Georgiadou, Elli and Milankovic-Atkinson, M. "Testing and Information Systems Development Lifecycles" paper presented at the 3rd. EuroSTAR'95.
4. Crosby, Philip B. "Quality is Free – The Art of Making Quality Certain", ISBN 0-451-62585-4

QW2001 Paper 3A1

Mr. Ralph Dalebout
(IBM Printing Systems Division)

Beta Testing With Rapid Development

Key Points

- Traditional Beta Test processes don't work well in today's rapid development environment.
- There is a new Beta Test approach which embraces a "total solution" focus.
- The preliminary results of the new Beta Test approach have been very positive for IBM.

Presentation Abstract

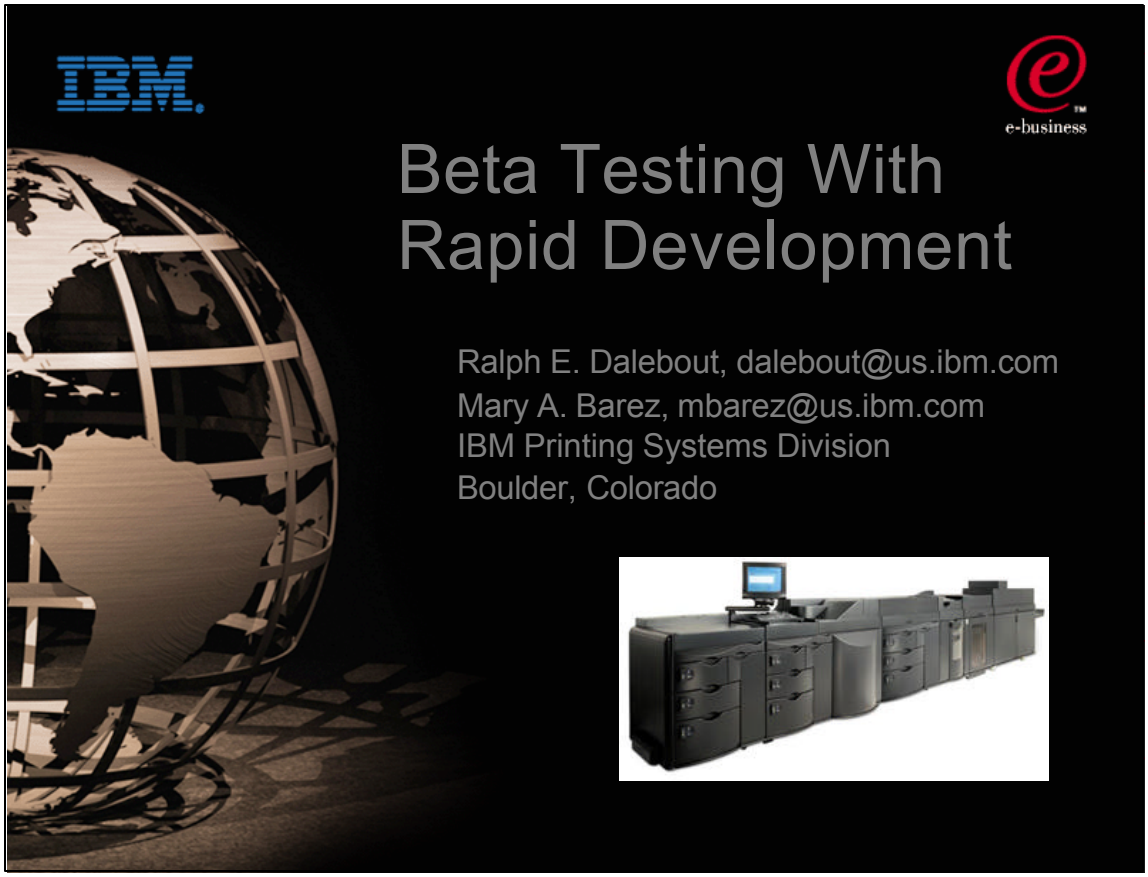
Today’s competitive world demands rapid product development and deployment. While the time-to-market window is shrinking, the sheer number and complexities of Beta Tests are increasing. Beta Test environments encompassing complex, high-availability, world wide networked environments are required to test end-to-end solutions. Traditional Beta Test approaches no longer work or are not cost-effective. IBM has implemented a new approach to Beta Testing and the results have been very positive.

About the Author

Ralph Dalebout has a degree in Mechanical Engineering from the University of Utah and a MBA degree from the University of Washington.

Ralph’s career with IBM spans 30 plus years. He has held a number of different positions in marketing (IBM Systems Engineer and Account Representative), product development (Product Planner), competitive analysis / market research and product test.

Ralph is currently a Beta Test Coordinator in the IBM Printing Systems Division’s Development Test and Support group. He manages hardware and software Beta Tests with customers inside and outside IBM. Ralph developed and implemented the Development Partnership Program, which is the focus of the technical paper and presentation.



Introduction

Presentation Scope and Audience:

- Definition of Beta Test
- Applicable principles and test process changes
- Business relationships most applicable for:
  - Medium to large customers
  - Medium to large providers of products

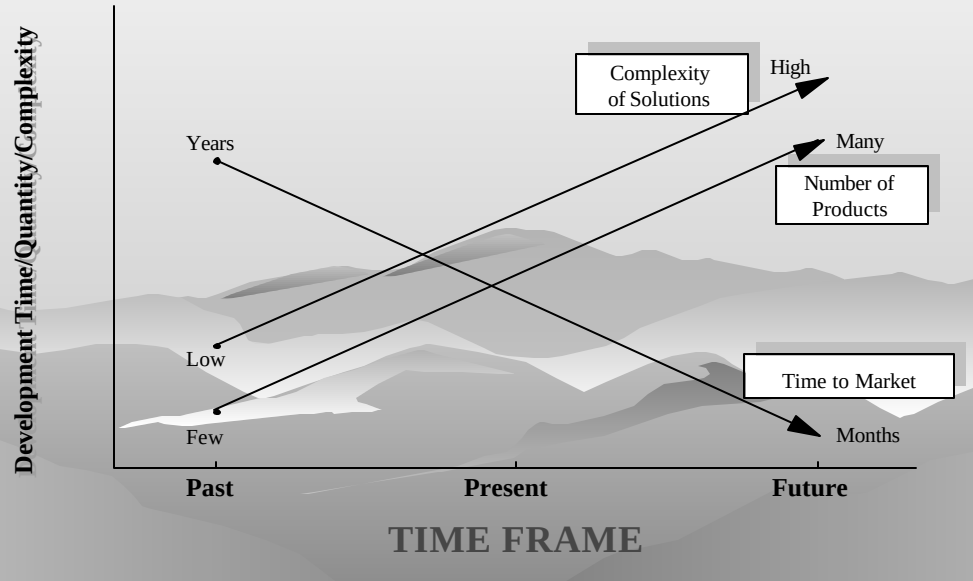
Major Points in the Presentation:

- Shifting paradigms
- Challenge to deliver better products faster and cheaper
- Traditional Beta Test process limitations
- Improving and leveraging Beta Tests
- The Development Partnership Program (DPP)

Product Development & Deployment Models

- Past: Manufacturer
  - Aggregated development
  - "Casters-up" development
  - Fairly long development cycles
  - Few new products released
- Present: Hardware (and Software) Integrator
  - Disaggregated products and customer environments
  - Using a leveraged model to face strong competition
  - Shorter development cycles
  - Many new / enhanced products released
- Future: Customer-Environment Integrator
  - Complete disaggregation
  - Even shorter development cycles
  - Customer-site integration of products
  - Focus on the customer solution

Major Development Trends



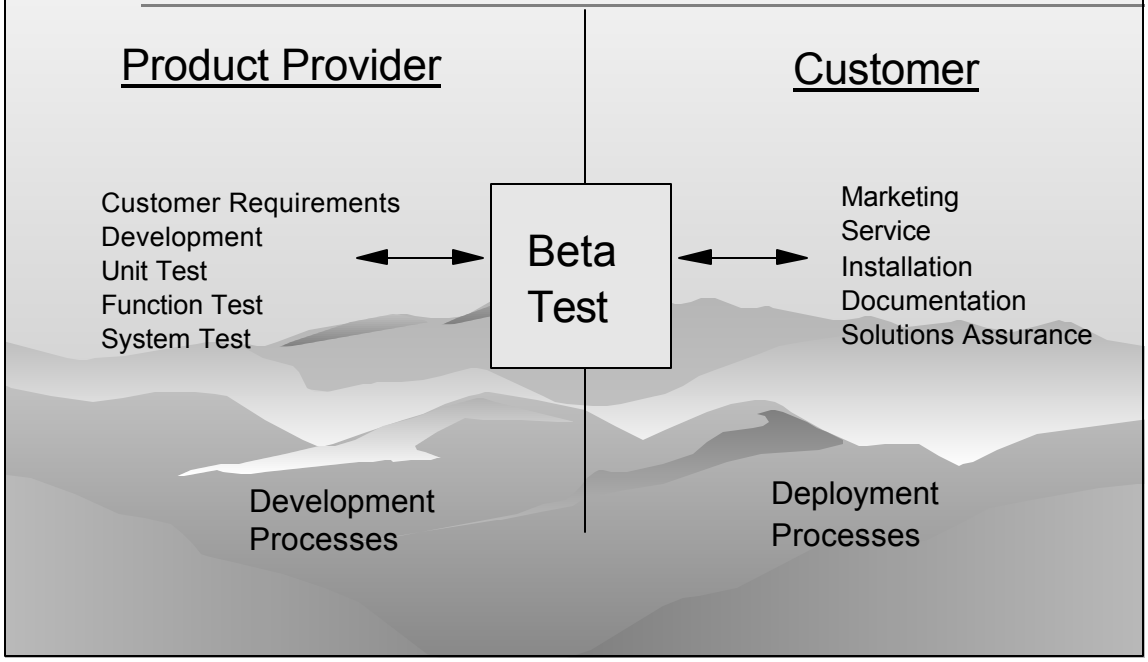
Why Focus on Beta Testing?

- Beta Testing is a natural focus for rapid deployment.
  - Beta Test is often the first customer contact with new product or new product enhancement.
  - Customer environments already exist.
  - Beta Testing often has to be done anyway.
  - Quicker product delivery dictates assessment of all test activities, including Beta Test.
- Beta Testing can be the easiest and least expensive way to mitigate test, schedule, and delivery risks.

Traditional Beta Test Limitations

- Time-consuming Process
  - As many as 18 steps or phases before even starting Beta Testing
- Complicated By External Factors
  - Marketing / competitive pressures
  - Misunderstandings and miscommunications
  - Changing IBM and customer commitments
  - Resource limitations and reassignments
  - Changing customer priorities
- Low or Limited Probability of Success

Development and Deployment



Using Beta Tests to Facilitate Rapid Deployment

- Using complex, disaggregated customer environments
- Integrating Beta Testing with other testing phases
- Early validation of installation, service, and maintenance procedures
- Augmenting test coverage
- Addressing customer feedback and problems in the product, improving final product quality
- Improving customer satisfaction and product acceptance

Development Partnership Program (DPP)

- Program was initiated in the middle of last year.
- Participants are selected based on previous Beta Test involvement and/or established relationship.
- The DPP is two years in duration and renewable.
- Participants are given product plan previews.
- Participation in any given Beta Test is optional.
  - Based on account environment / requirements
  - Available resources / timing / commitments
- A simple agreement between IBM PSD and an established customer is put in place.

Development Partnership Program (DPP)

Major program elements:

- Customer review of all products prior to announcement
- Early validation of selected products via a Beta Test
- Test coverage in new or complex environments
- Earlier installation and integration, resulting ultimately in earlier customer deployment
- Cultivation of customer references

Major Beta Test deliverables:

- Defect reporting and tracking during and after test
- Periodic status reports published to all stakeholders
- Product evaluation and feedback to development
- Lessons-learned (final report) for each product

Development Partnership Program Results

- Representative printing market segments are covered.
- "Continuous" Beta Tests within a customer relationship eliminated much of the overhead of individual Beta Tests.
- The DPP relationship with the customer helped isolate product deployment from external and internal factors.
- The DPP provided flexibility and benefits for both IBM and its customers.
- The success rate and the value of the Beta Test increased dramatically under the DPP.
- The DPP resulted in much faster and better customer acceptance of the product(s).
- The DPP can be used outside of IBM Printing Systems.

Extrapolating the DPP Beta Test Model

Future Challenges:

- Still shorter development time to market
- Difficulty keeping pace with
  - Technology
  - Changing customer environments
  - Increasing user sophistication
- Addressing cost containment while staying competitive
- Effective and timely product quality assurance

Extrapolating the DPP Beta Test Model

Using the DPP to Respond to the Challenges:

- Get the product into customers' hands as early as possible in the cycle, reducing overall time to market.
- Establish ongoing Beta customer relationships to facilitate integration of products into customer environments, without the traditional Beta Test overhead.
- Use Beta Test relationships to leverage and integrate all phases of testing.
- Provide early service process validation.

Future DPP Extensions

- Expand the number of participants
- Add new / more complex customer environments
- Modify interfacing processes to take advantage of DPP
- Analyze other test phases to reduce redundant testing
- Provide quality assurance not conducive to lab testing
- Port to non-printing products

Conclusions

- Test organizations need to aggressively change in order for companies to stay competitive.
- Beta Test can play a critical role in test changes.
- The traditional Beta Test process doesn't help much.
- The DPP gives flexibility to Beta Testing.
- The initial DPP Beta Test results are very positive.

Beta Testing With Rapid Development

Ralph E. Dalebout
and
Mary A. Barez

IBM Printing Systems Division
Boulder, Colorado

May, 2001

Introduction

Today's competitive marketplace and global economy require rapid product development and deployment. While the time-to-market window is shrinking, the sheer number and complexities of products are increasing. In order to meet the dictates of rapidly advancing technology and rapidly increasing customer sophistication and demands, the frequency and quality of hardware and software releases must find a way to keep pace.

Over time, the focus of testing is naturally migrating from specific hardware and/or software testing to customer "solution" testing. Also, many providers of services and products still appear unaware how to successfully leverage and integrate different test efforts and activities, and some even seem unaware of the value and bottom-line (revenue) criticality of test effectiveness.

Other than performance testing for networks, operating systems, and web loading, testing tools and test automation have limited applicability and value in a dynamic and fast-paced development environment. In addition to the limitations of tools and the expensive tool requirements (co-requisite software and hardware), tool licenses and the training to obtain tool skills and expertise are in themselves costly. Test tools and test automation, therefore, are often not cost-justifiable, and cannot be implemented or updated fast enough to meet aggressive development schedules. Most importantly, tools often cannot be used with a diverse product set, particularly integrated hardware and software products such found with printing products. With printing products, simulation, particularly of hardware, is often difficult; test tools are not as much value or are too costly for product development and testing to be competitive.

This development framework and the need for rapid market response are occurring at a time when development and test organizations are facing spiraling costs and cost containment pressures. This is especially true with printers and other peripherals, where the competition is fierce and profits are constrained.

Beta Test environments encompassing complex, high-availability, and worldwide networked environments are required to test end-to-end solutions. In many cases, traditional Beta Test approaches are no longer cost-effective or are simply too slow to implement. To address these challenges, IBM Printing Systems Division has implemented a new approach to Beta Testing, the Development Partnership Program (DPP), and the initial results have been very positive.

.

Product Development and Deployment Models

To understand the new challenges that need to be addressed by the Beta Testing process, an understanding of how the product development process has evolved is required. Next a basic understanding of the Beta Test Process, the relationship of Beta Test to other test efforts, and the reasons for a new Beta Test approach are important. Finally, from a print-product perspective, the use of Beta Test to support and promote rapid product development and deployment becomes a reality, addressing the shift in the product development paradigm.

The Past: Manufacturing -> A "Casters-Up" Approach

IBM's development model was to completely design and develop printers from the ground up. The entire design was IBM's, with the exception of the power supplies, and those were developed to IBM's specifications. This development model is sometimes referred to as a "casters-up" design and development approach, with IBM in the role of manufacturer.

One of the major problems with the "casters up" process was the time required to develop and deliver a new product. Tremendous technical resources (engineering, testing, and production planning) were expended to develop a complete product. During the development period, significant changes could occur in the marketplace, including new customer requirements and increased competition, as well as the introduction of new technologies that could not be incorporated into the developing product.

The result of this approach was that very few major new printer products were delivered each year. However, with such a long development cycle and only a few new products introduced each year, it was fairly easy to fit product quality assurance and Beta Tests into the overall development and production cycle.

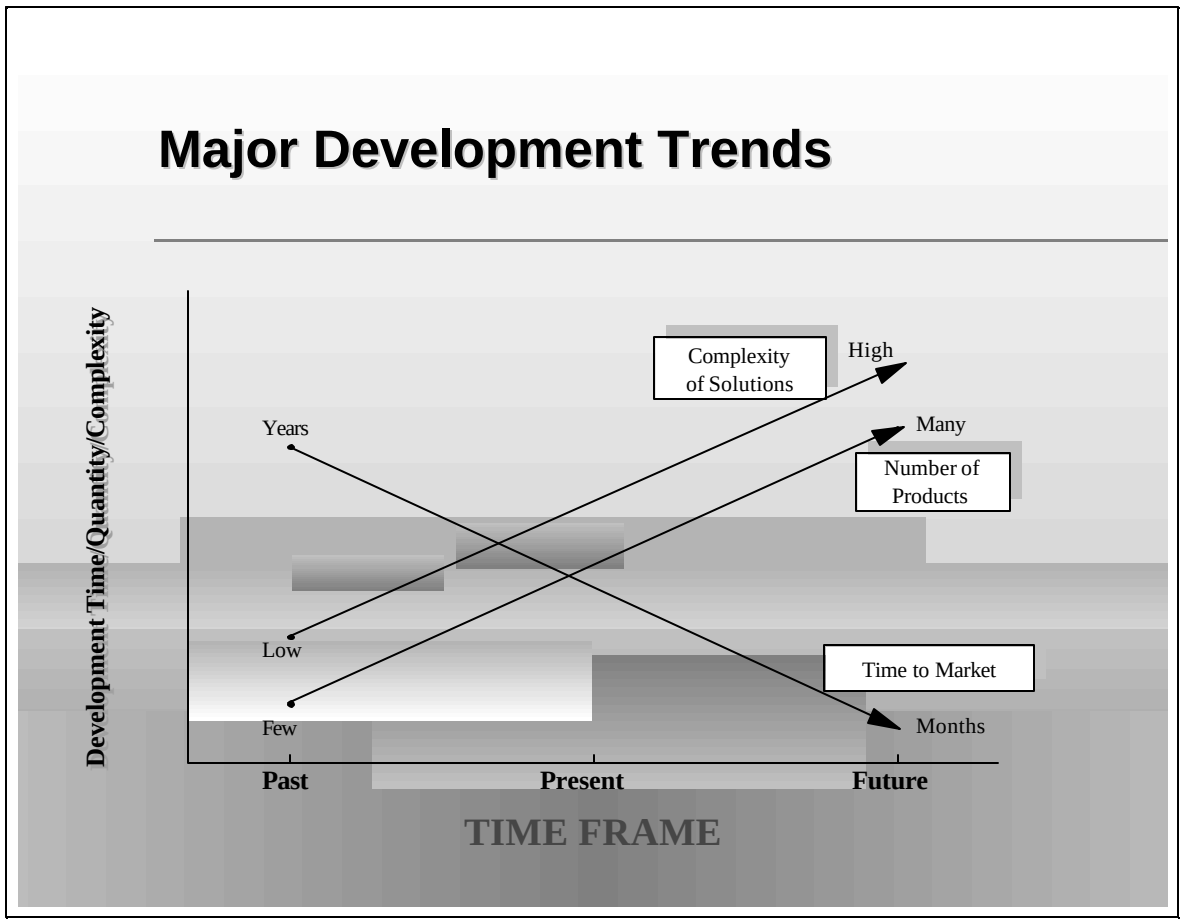
The Present: Leveraging -> Hardware Integration or Purchase Complete

Over time, IBM, and most other computer companies, made a dramatic shift in their product development model. They moved from a "casters up" model to the current "leveraged development model." Rather than develop the entire hardware and/or software product, components were purchased from other companies. These other companies / suppliers are often competitors in two ways. The competitor may use the purchased component (under the covers) in their own product. Secondly, the component is also sold to multiple companies other than IBM. This is commonly seen in the automotive industry, where auto makers will acquire transmissions and engines from other automotive companies for some of their automobiles.

Within this model, IBM provides product differentiation in their products by providing standardization and architecture, additional functionality, and component integration and testing. In addition, IBM provides the co-requisite services offerings, resulting in a complete end-to-end solution for customers. The "buy versus make" options are continually evaluated. In comparing past development to the

present development model, one of the dramatic differences has been the number of new and enhanced products delivered each year. In 2000, IBM Printing Systems Division delivered many new and enhanced hardware and software products to the marketplace, and these new products are often far more complex and sophisticated than the few products delivered using the past manufacturing development model.

With this hardware-leveraging model, the time to market has improved markedly. On a recent printer announced by IBM, the project inception to delivery of the new printer was under six months. With this leveraged model, IBM can be viewed as a hardware integrator and value-added supplier.



These industry trends are depicted in the above diagram.

Traditional Beta Test Process

To better understand the challenges facing test organizations, especially as they pertain to Beta Testing, one must first understand the traditional Beta Test process. The following sections provide an overview of the traditional Beta Test process, using the IBM Printing Systems Division as a point of reference, followed by the challenges associated with the process.

A Long and Involved Process

The following is a summary of the possible steps, not necessarily consecutive or in order, involved with a traditional Beta Test, from initial project definition to actual start of Beta Test. Given that an individual product could have several customers involved in the Beta Test, the amount of work and project management required can be staggering.

1. Define and convene Beta Test core team.
2. Define the functional attributes and/or applications that need to be tested.
3. Define reliability, availability and serviceability attributes and/or applications to be tested.
4. Establish general test procedures, Beta Test metrics and test entry / exit criteria.
5. Establish the number of Beta Tests (customers) needed and the duration of each.
6. Develop an initial Beta Test plan.
7. Solicit input from the various support organizations who will be involved in the Beta Test.
8. Publish final Beta Test Plan to the implementation team.
9. Describe customer profiles required to ensure successful Beta Tests.
10. Identify the customer's technical support profile to ensure a successful Beta Test.
11. Rank priorities of all Beta Test customer variables for selecting final Beta Test customers.
12. Review the Beta Test Plan and selection criteria with Marketing and Service account teams.
13. Request Marketing to nominate Beta Test candidates and provide account profiles.
14. Review Beta Test candidates and select accounts which best fit the selection criteria.
15. Obtain approvals and legal clearances to disclose unannounced product to customer.
16. Disclose unannounced product(s) to customer.
17. If customer is interested in participating in the Beta Test, review the draft Statement of Work describing the responsibilities of both parties and the guidelines of the Beta Test.
18. Finalize the Beta Test Statement of Work and have both parties sign.
19. Initiate the Beta Test.

When you review the number of steps leading up to the initiation of a Beta Test, it is fairly clear this can be a time-consuming process. While you can do a number of things early in the cycle, there are many things that can't be done until fairly close to the actual start of the Beta Testing.

Beta Test Limitations and Restrictions

There are a number of limitations and restrictions inherent in the traditional Beta Test process.

Influenced by External Factors and Dependencies

The successful execution of the Beta Test process can be impacted by a number of factors. In many cases, Marketing input and customer relationships (and, therefore, marketing commitment and time) are necessary to provide qualified candidates for the Beta Test. In some cases, the marketing and sales staff attempt to use the Beta Test to resolve a competitive situation and/or marketing issue. Also, there are many trials and tribulations on the road to the final list of Beta Test candidates.

Once you get into the actual Beta Test, a number of problems can surface.

- There is a misunderstanding between IBM and the customer relative to the scope of the Beta Test, the deliverables, and/or the commitments agreed to by each party.
- There isn't sufficient support from all the parties (IBM and customer) who need to be involved in the Beta Test.
- The resources necessary to assure a successful Beta Test aren't available. Work priorities change or assignments and roles change. Support personnel aren't available when required due to personal problems such as medical leave, attrition, or a death in the family.
- Customer mergers, acquisitions, and consolidations can significantly impact Beta Tests.
- If schedules for the Beta Test shift, the window of opportunity or interest level for some customers can change.

Low Probability of Success

At the end of the Beta Tests for a new product, the level of success achieved in the Beta Test is evaluated by mapping the test results back to the original Beta Test objectives and assessing the degree of success. Also, the results, or average, of multiple Beta Tests can be combined and/or summarized to make a more general product assessment. Experience has proven that a majority of the Beta Tests don't meet their planned objectives. This conclusion does not eliminate the need for Beta Tests, but it does indicate that is there is much room for improvement in the Beta Test process.

Not an Integrated Test Approach

Beta Testing is often viewed as an early predictor of product success and not often viewed as part of the overall product Quality Assurance strategy and efforts. That is, Alpha and Beta Tests are rarely a fully integrated phase in the testing of the product. In many instances, the Beta Test may be coordinated by a completely different group or department then the group that is providing product quality assurance services.

High Cost and Overhead

The overhead required to plan and implement a Beta Test is costly and time-consuming, and some overhead must be expended for each Beta Test customer/site. There are almost always multiple Beta sites for a major hardware or software product.

The Role and Importance of Beta Tests

There are a number of reasons why a focus on Beta Testing is valuable to understanding how to improve time to market, and for addressing cost containment while staying competitive, and while concurrently providing effective and timely product quality assurance.

Beta Testing can be the easiest and least expensive way to mitigate test, schedule, and delivery risks. and to are becoming a more critical factor in the total testing process.

Natural Uses of Beta Testing

Alpha and Beta Tests are the first place where the developed product and the customer are introduced. In order to assure delivery of a solid product / solution to the customer, Beta Tests are a natural way to test in a complete customer environment with a focus on the end-to-end customer solution. For printing products, the customer is often willing to use the product in complex environments and with production (non-printing) applications and systems. This customer capability inevitably finds product defects and problems that cannot be found in the test lab that has a strictly printing focus.

The beauty of Beta Testing is product transparency to the user and to the provider of the product being tested.

New Testing Requirements

There are always new or changing requirements for the testing of a product. Many of these cannot easily or quickly be tested within the test lab or within the planned testing phases. Some examples of new requirements that are best tested in the customer's environment are National Language Support (NLS) and enablement, cultural basis and use of products (this is particularly relevant for printing), accessibility, interoperability, network configurations, access and security.

Strong Impetus for Improving Beta Testing

There are a number of reasons Beta Tests are becoming a more critical factor in the complete test process to assure delivery of a solid product / solution to the customer. Some of the facets of Beta Testing that make it a place for improvement are:

- Uses complex, disaggregated customer environments
- Potential for integration of Beta Testing with other testing phases
- Early validation of installation, service, and maintenance procedures
- Potential for augmenting test coverage
- Addresses customer feedback and problems to improve final product quality
- Potential to improve customer satisfaction and product acceptance

Augment Product and System Testing

The integration of Beta Test into overall Quality Assurance strategy and phases can improve development time and lower development costs.

- The importance of getting the solution or product to the customer sooner dictates shorter test cycles, and promotes either earlier or concurrent Beta testing in all cases.
- Testing is not only a viable career path, but testing discipline and services are costing companies more and requiring more skills and training. Therefore, integrating and capitalizing on all the test activities is critical to quick and effective quality assurance. There is little room for testing overlap and redundant work.
- It is easier and faster to leverage test efforts to improve quality than to try to achieve the same results through development economies or re-engineering of product development processes.
- Beta Test, unlike other test phases, assures product viability for the customer in the customer's environment. This is an extremely powerful concept.

Complex, Real World Customer Environments

A primary objective of test is to simulate customer environments (usage and workload) to ensure the new products / solutions will operate correctly in the customers environments.

Actual customer environments have become more complex over time. With continuing globalization, operations are worldwide and products must be available 24 hours a day, seven days a week. Networking and connectivity are paramount and many of the communicating devices are remote. Many customers manage thousands of networked printers located around the world from a single center.

To recreate typical customer environments (hardware, software and networking) in a test lab today is nearly impossible. Beta Testing can not only be an attractive extension to address this testing requirement, but a way to meet the specific needs of valued customers.

Increasing Interdependency of Hardware and Software

In the past, the interdependency between hardware and software was fairly limited. The interfaces / protocols were well-defined and could be fairly easily simulated on either side of the hardware / software equation.

In contrast, consider a printer and the components required to make the printer operational. The printer itself can have over thirty microprocessors managing different components of the printer. For printer front-end processing, there may be a control unit communicating via different communications networks and protocols to printing services software (IBM and non-IBM). The printing services software can be running on five (or more) different operating system platforms that are, in turn, running on a multitude of different processor architectures. The printing services software can be communicating with an Enterprise Resource Planning (ERP) system that is creating print spool files and handing them off to the printing services output manager. Finally, all this can be operating in a high availability environment with automatic backup and fail-over capabilities!

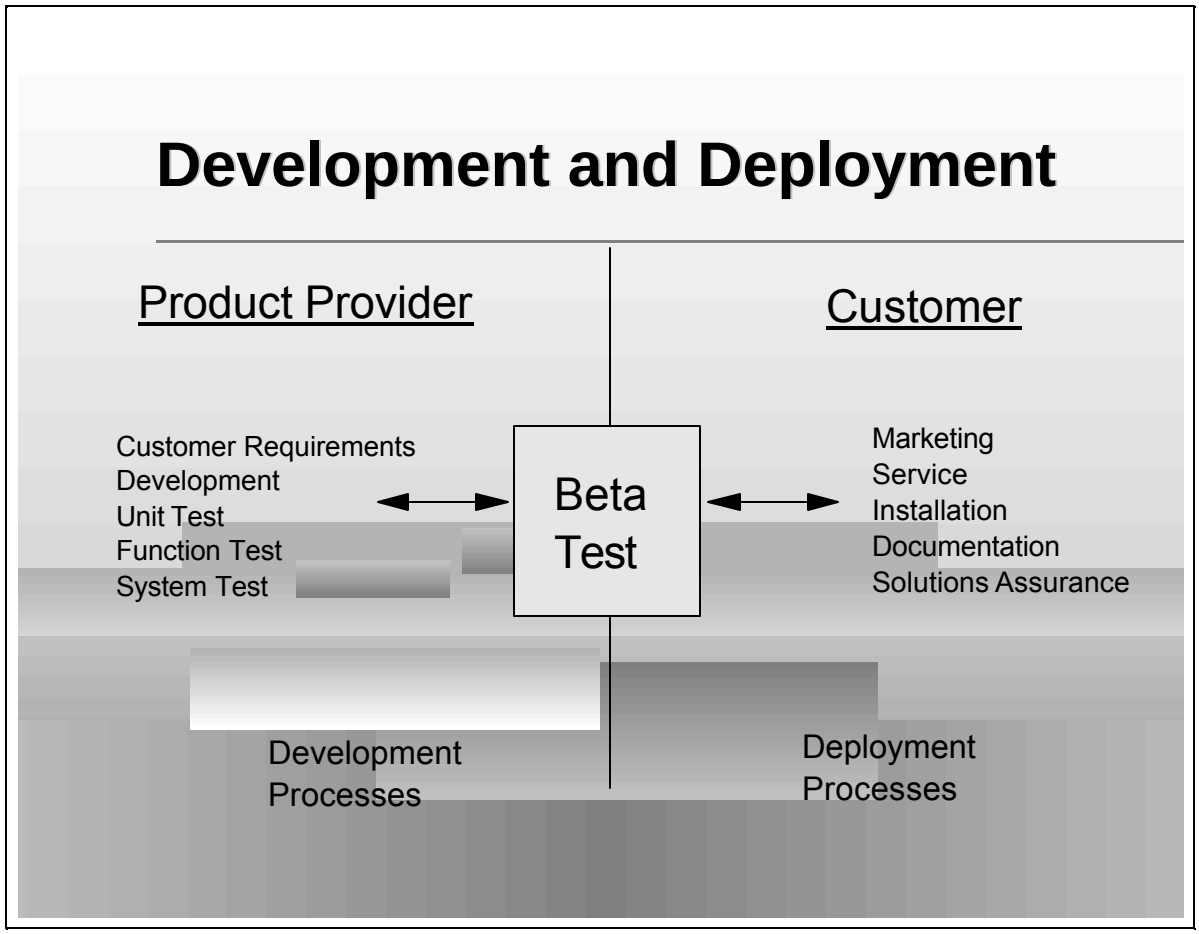
It is apparent that there are a huge number of combinations and permutations. The testing challenge is enormous, with inherent problems with test coverage, variations, risk mitigation, and scope. Setting up a test lab to reflect even the most common customer environments is difficult.

Providing Customer Solutions

Customers want assurances that new and enhanced products have been tested in an end-to-end environment similar to theirs. It isn't enough to have tested a few of the components / elements with one another. The emphasis is on testing complete environments or solutions. This is where Beta Testing can complement traditional development testing, and provide an early method of product integration into not only the customer environment but the customer's solution.

Development and Deployment Impacts

The following diagram depicts the role of Beta Test within the development and deployment processes, and hopefully shows why a focus on Beta Test can expedite and improve both development and deployment cycles.



Rapid Product Development and Deployment

There is a serious mismatch between the new rapid development paradigm and the traditional Beta Test model. One of the major problems is that the shortened development cycles don't allow enough time to implement a traditional Beta Test. The traditional approach requires too much time leading up to the actual start of the Beta Test.

Another major problem is that the probability of success of the traditional Beta Test is too low and needs to be improved. With a traditional Beta Test approach, the customer sometimes cannot afford or tolerate the overhead or time required to support the effort.

The Development Partnership Program (DPP)

IBM Printing Systems Division has developed a new Beta Test program, the Development Partnership Program (DPP), that addresses a number of these problems and requirements. The DPP isn't an complete answer and it isn't a one-for-one replacement for traditional Beta Testing. However, it has a number of advantages and the initial results have been quite positive. The Development Partnership Program provides a vehicle to streamline the overall Beta Test process, while significantly improving the Beta Test's probability of success. It shrinks the end-to-end time for the Beta Test selection and implementation and does this using less test resources.

DPP Program Overview

An overview of the program is as follows:

- Participants are selected based on previous involvement in one or more hardware and/or software printing product Beta Tests. Experience shows many of the same Beta Test customers show up again and again with Beta Tests for new or enhanced products.

- The program is two years in duration and is renewable. The guidelines and framework are very similar to what is included in a normal Beta Test Statement of Work.

- Two major elements of the program are pre-announce and/or pre-development product evaluation and early product validation.

    1. The program participants are given previews of product plans, to assist them in determining what new and enhanced products they would be interested in Beta Testing.
    2. The DPP participant has the choice of what Beta Tests they want to participate in based on applicability to their environment and available resources (people and hardware). For example, a company who has decentralized operations and doesn't utilize high-end printers (1,000 plus pages a minute) wouldn't be interested in Beta Testing a new high-end printer.

- The major Beta Test deliverable is product feedback and evaluation to development groups, but other deliverables include:
    1. Defect reporting and tracking during and after test
    2. Periodic status reports published to all stakeholders
    3. Lessons-learned (final report) for each printing product.

- A **simple** agreement between IBM Printing Systems Division and an established customer is put in place.

Advantages of the Development Partnership Program

- Over time, the customer environment and requirements are well-known and are updated, as well as the customer's profile.

- The customer has practical experience and realistic expectations on how the Beta Test works.

- The relationship between the two parties is well-established. There is a low probability of any surprises. DPP capitalizes on established customer relationships and established Beta Test agreements to provide ongoing Beta Tests for many/several software and hardware products.

- The DPP is a two-way street in several ways, providing faster and better quality delivery and support for the customer and providing IBM with difficult-to-do-in-house testing, early product integration, and faster product delivery. Some examples of this symbiotic relationship are:
  - ⇔ Defect tracking and the resulting fast(er) delivery of fixes
  - ⇔ Using the customer environment and/or test regions to validate a product with less investment and hardware / software infrastructure on both sides
  - ⇔ Customizing or configuring the product to assure integration and use in the customer environment and with the customer's applications
  - ⇔ Marked improvement in the contents, timeliness, and quality of installation instructions, tips and techniques, and user guides for both the Beta customer and the final product
  - ⇔ Customer jobs are run by both sides, improving and refining other product test phases for the benefit of the specific customer.

- DPP allows leveraging of regular test resources and test lab environments, saving not only on testing time, but on expenses for test equipment, lab space, equipment setup and support.

- The time-consuming process of qualifying a new Beta Test candidate is avoided.

- The probability of success is significantly improved.

- Less staff and critical resources are required to conduct a Beta Test under the DPP, including fewer builds and fewer requirements on delivery mechanisms and formal documentation.

- Joint and fairly seamless integration of the product into the customer's environment.

- Dependencies on external factors and groups are markedly mitigated, including mitigation of marketing, sales, and customer changes and problems.

Results Of The Development Partnership Program

Current Status

The accounts participating in the Development Partnership Program currently provide Beta Test coverage for the major printing marketing segments. The program was initiated in the middle of last year, focusing initially on IBM internal accounts that provide printing services to IBM and to non-IBM customers.

The program has allowed IBM Printing Systems Division to support a greater number of Beta Tests in shorter timeframes with less personnel. "Continuous" Beta Tests within a customer relationship eliminated much of the overhead of individual Beta Tests.

The DPP relationship with the customer helped isolate product deployment from external and internal factors. The DPP has provided flexibility and benefits for both IBM and its customers.

The DPP has resulted in greater customer satisfaction, providing reference accounts with positive experiences and product recommendations. This is due to the fact that the DPP has resulted in much faster and better customer acceptance of the product(s).

The success rate and the value of Beta Testing has increased dramatically under the DPP, and the DPP is applicable outside of IBM Printing Systems Division.

Next Steps

The intent is to expand the number of participants, focusing on external accounts. Specifically, there is a focus on the selection of Beta Test customers who improve product test coverage and who best represent the product marketplace. This focus includes addressing global, cultural, accessibility, or special customer product requirements that are difficult to test outside of the customer's environment. IBM is also actively using Beta Test as a means to verify that customer printing jobs and applications run successfully prior to product announce. This strategy is crucial to successful and earlier product deployment to all customers, not just Beta Test accounts.

The Future: Extrapolating the DPP Beta Test Model

Shorter Development Cycles

The future brings a continued challenge, as refinements to the existing leveraged-development and the purchase-complete development models occur. The ongoing time-to-market pressures and additional customer requirements will dictate further shortening of the development and test product cycles. Additionally, the need for rapid deployment will escalate in order for products to take advantage of new technologies and hardware improvements.

Complexity of Customer Environments

The use of Beta Testing provides products that not only exactly meet the customer's specifications and requirements, but provides for a smoother integration of the product in the customer's environment. This is mandatory for printing products, but is also valuable for other software and hardware products.

Strong Customer Relationships

The idea of having an ongoing Beta relationship with a whole set of customers promotes a plug-and-play mentality for new products. This is particularly useful with new printers, but can be applied with almost any software or hardware product. Facilitating early product deployment in an almost continuous Beta Test environment also provides a validation of product marketing requirements. Customer Beta Test feedback can be used to adjust marketing plans and sales objectives.

Early Product Integration

Having products already integrated in the customer's environment promotes and expedites integration of replacement products and the addition of new products. The customer is experienced with the product set and installation procedures, and already has contacts and staff in place.

Summary

Therefore, the DPP promotes and expedites migration to, and integration of, new products. With an established Beta relationship, the customer is more likely to use (and hopefully purchase) products that would not be included in a separate Beta Test or considered critical for outright purchase. The DPP better enables the paradigm shift from product manufacturer to product integrator than traditional Beta Tests. This paradigm shift must be made as quickly as possible to provide products that function and integrate easily in today's disaggregated environments.

QW2001 Paper 3A2

Mr. Roger M. Records
(Boeing Commercial Airplane Group)

Assuring Quality In Outsourced Software

Key Points

- Knowledge, experience and skills required to achieve successful subcontract management
- Definition of seven essential subcontract management functions to ensure quality products
- Maximize quality assurance with high leverage QA reviews for both products and processes

Presentation Abstract

This paper shows how to mange the quality of subcontracted (out-sourced) software through the use of high-leverage quality reviews. The reviews address key process and product issues across the entire subcontracting life cycle (from RFP preparation to product acceptance). The context for the review discussion is our Subcontract Process Model which is compliant with SEI/CMM Level 2. The supporting SQA process and product reviews can be tailored for project/contract size to assure quality built-in for the subcontracted deliverables. The model provides the foundation to describe a proven strategy for validating the achievement of project objectives, while avoiding cost and schedule overruns.

About the Author

Roger M. Records is an Associate Technical Fellow with Boeing Commercial Airplanes. He is currently working with a Software Quality Assurance group with responsibilities in SQA courseware design and SQA technology transfer. He also works as a project manager in Airplane Safety Engineering with responsibility for offshore subcontracting. From his experience with five subcontracted projects, he has established and validated a QA solution for the subcontracting lifecycle and is currently training other project managers in its use. Prior to his work in Software Quality Assurance, Mr. Records supported contract work in the human factors domain for NASA and the FAA with Boeing's Flight Deck Research group. He is a co-inventor for a U.S. patent held by Boeing for electronic checklist software. He has authored or co-authored 11 technical papers for national publication and conference presentations.

Assuring Quality in Outsourced Software

Roger M. Records
Associate Technical Fellow
roger.records@pss.boeing.com

BOEING

Problem: How to get . . .

New groups doing subcontracting

Facts & Data to attain

New suppliers providing service

GOAL:

Successful Contract

- Build the right stuff
- Deliver on time
- Deliver it for contract price and make a profit

RESULT:

- *Successful contractors*
- *Quality products — useful to the Project*
- *Delivery: On time, within budget*

BOEING

The Keys to Success in SSM

Right Process

- Based on SEI/CMM Experience
- Documented Process and Project Roles

Experience

- Tailored Procedures and Reviews
- Verify Quality of SSM Deliverables

SQA Training\Coaching

- Utilize Proven QA Process
- Training and Coaching Using Templates

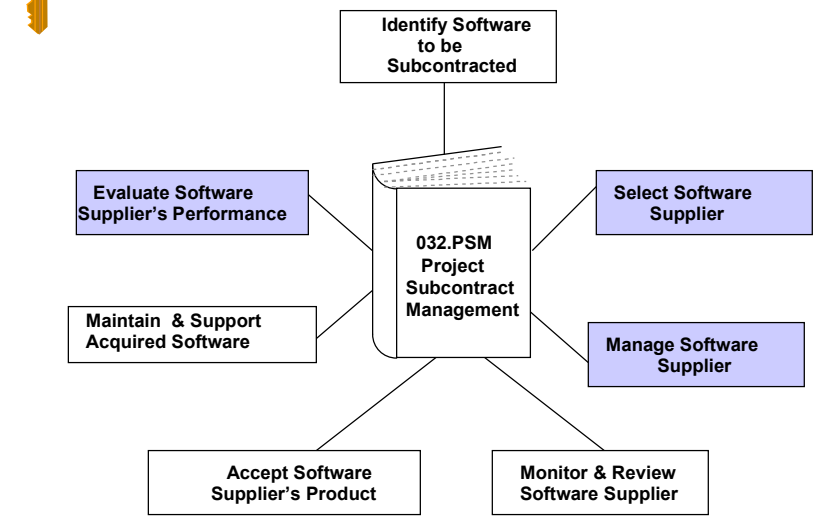
BOEING

Project Role in SSM

	Contract Admin	Project
Identify Software Project for Subcontracting	• Provide Guidelines	• Evaluate Application Candidates
Prepare RFP	• Draft RFP • Legal Requirements • Procurement • Technical	• Provide Technical Requirements • Answer Supplier Queries
Select Supplier	• Review Proposals • Qualify Legal • Qualify Procurement	• Review Proposal • Technical • Supplier Competent
Manage and Monitor Supplier	• Monitor Performance • Coordinate Payment	• Review Deliverables • Quality Assurance
Accept Software Product	• Coordinate w/Project	• Quality Assurance • Acceptance Testing
Evaluate Supplier	• Performance Tracking • Coordinate w/Project	• Quality Assessment • Performance Assessment

BOEING

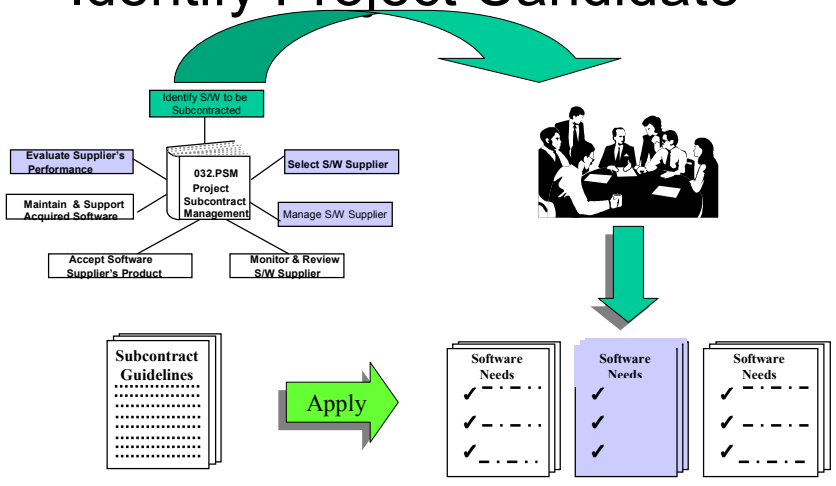
Subcontract Process Model



Joint Project/Contract Admin

BOEING

Identify Project Candidate



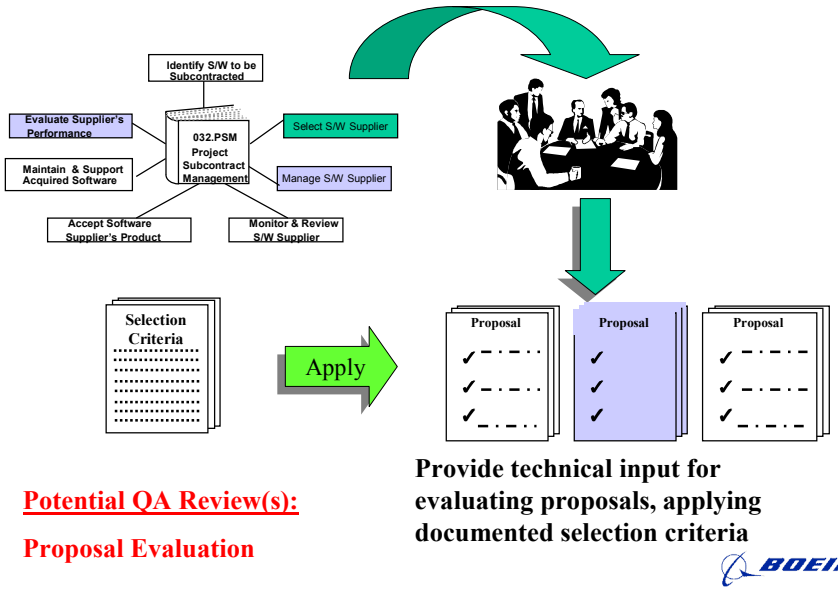
Potential QA Review(s):

RFP Requirements

Evaluate candidates for subcontracting using selection guidelines

BOEING

Select Supplier



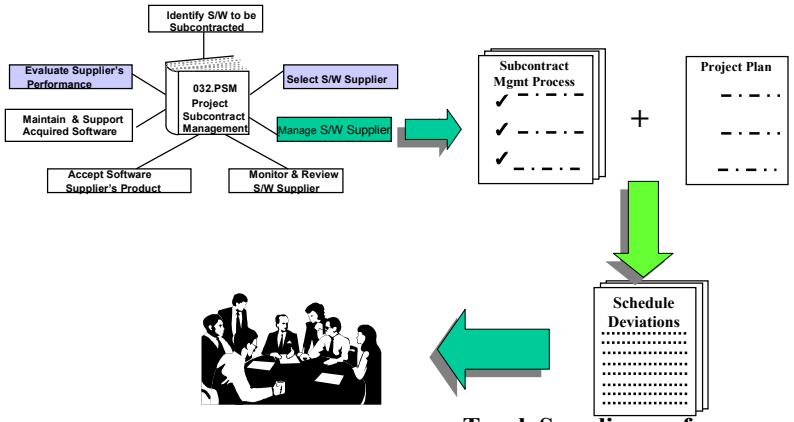
Potential QA Review(s):

Proposal Evaluation

Provide technical input for evaluating proposals, applying documented selection criteria

BOEING

Manage S/W Supplier



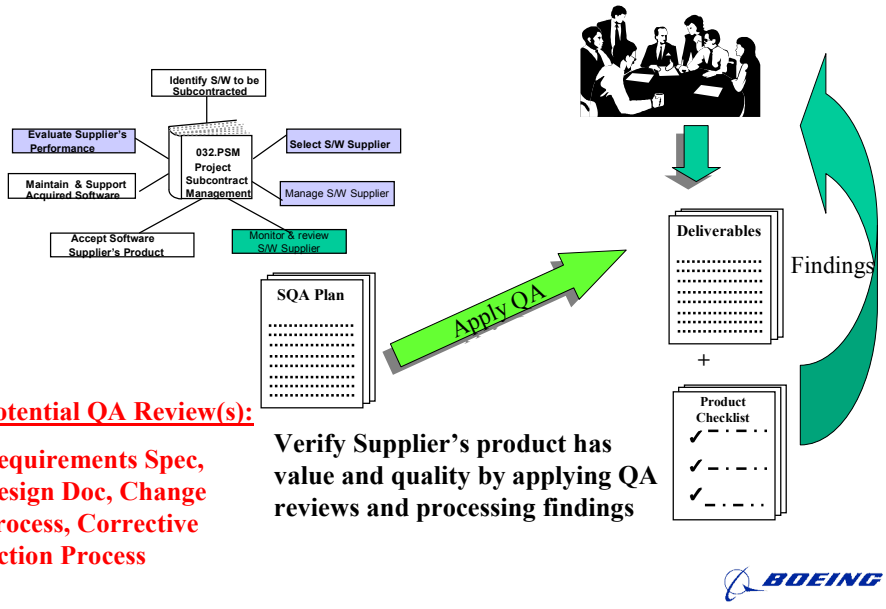
Potential QA Review(s):

Project Plan, SSM Process

Track Supplier performance and report deviations for corrective action

BOEING

Review Supplier's Deliverables



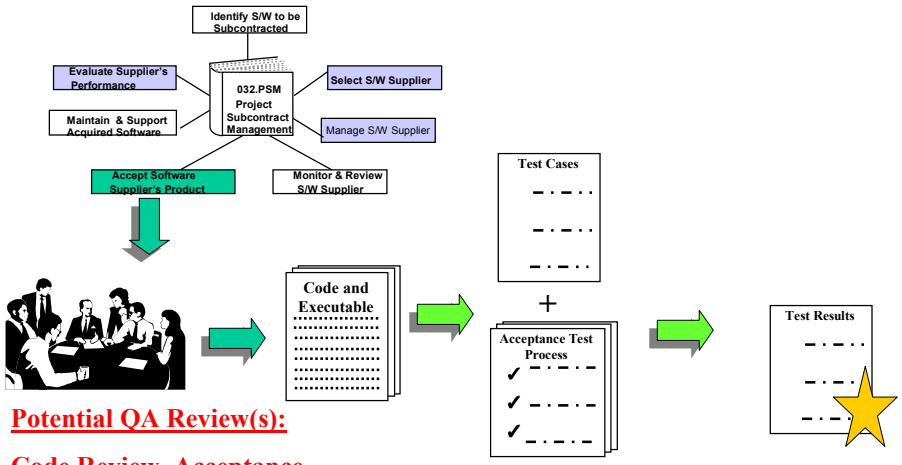
Potential QA Review(s):

Requirements Spec, Design Doc, Change Process, Corrective Action Process

Verify Supplier's product has value and quality by applying QA reviews and processing findings

BOEING

Accept Supplier's Products



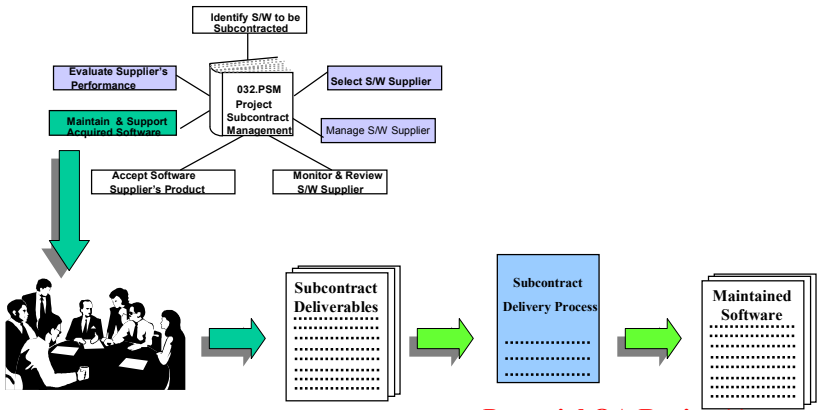
Potential QA Review(s):

Code Review, Acceptance Test, User's Manual

Perform acceptance tests on executable delivered by supplier and generate acceptance report.

BOEING

Maintain Delivered Software



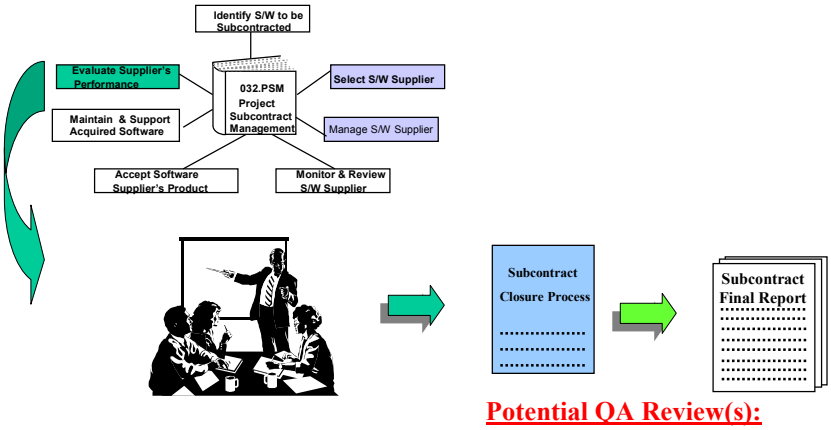
Perform Subcontract Delivery Procedure using contracted deliverables.

Potential QA Review(s):

Delivery Process, Status Reporting

BOEING

Evaluate Supplier's Performance



Project Manager assists in evaluation of supplier and recording lessons learned.

Potential QA Review(s):

Completion Process, Final Report

BOEING

The Project's Real Work in SSM

	Contract Admin	Project
Identify Software Project for Subcontracting	• Provide Guidelines	• Evaluate Application Candidates

Prepare RFP	• Draft RFP
	  • Legal Requirements
	  • Procurement
	  • Technical
		• Provide Technical Requirements
		• Answer Supplier Queries

Select Supplier	• Review Proposals
	  • Qualify Legal
	  • Qualify Procurement
		• Review Proposal
		  • Technical
		  • Supplier Competent

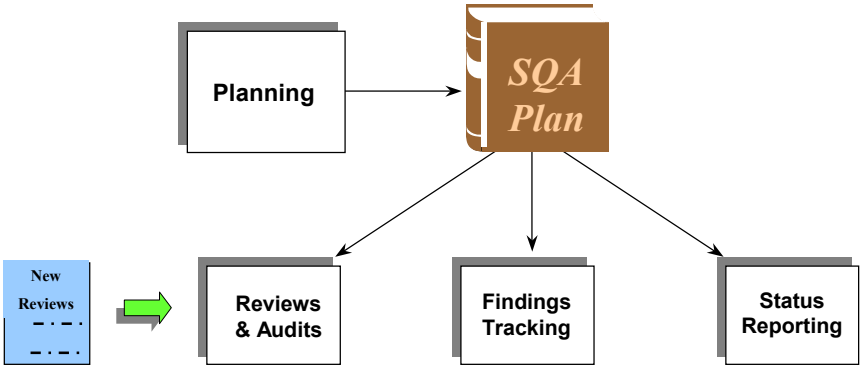
Manage and Monitor Supplier	• Monitor Performance • Coordinate Payment	• Review Deliverables • Quality Assurance
Accept Software Product	• Coordinate w/Project	• Quality Assurance • Acceptance Testing
Evaluate Supplier	• Performance Tracking • Coordinate w/Project	• Quality Assessment • Performance Assessment

After the Contract is Issued ...

60-80% of Subcontract Management Activity is <u>Quality Assurance</u> for Contract Deliverables

BOEING

SQA Process Model

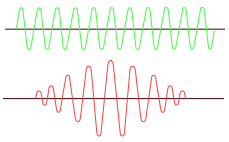


BOEING

Project Manager

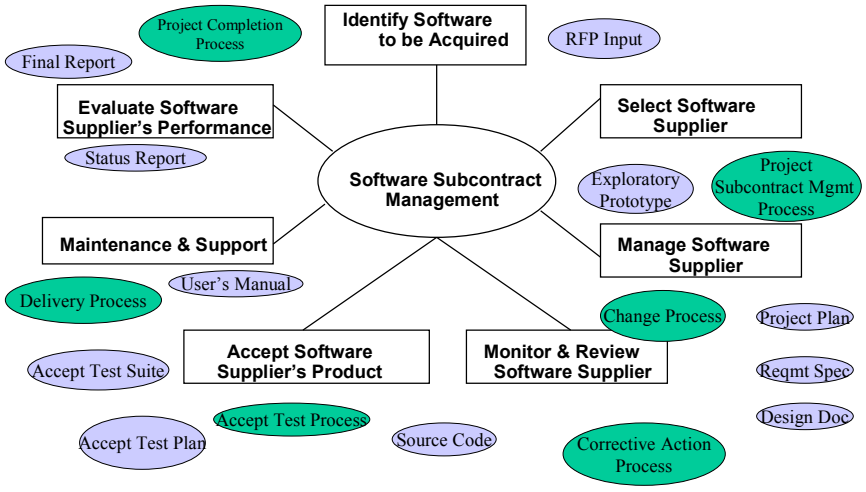
Needs Experience with SQA

Communication Offshore is a Challenge



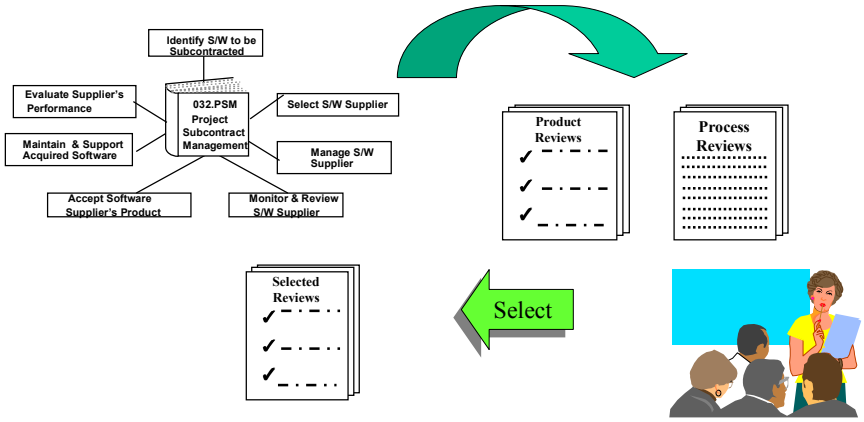
BOEING

SQA Coaching for SSM
Step 1: Awareness Training



BOEING

SQA Coaching for SSM
Step 2: Select Reviews



BOEING

Project Manager, SQA Focal and domain specialists choose high leverage QA reviews

SQA Coaching for SSM
Step 3: Tailor Reviews

BOEING

Project Manager , SQA Focal and domain expert use group standards to tailor checklists.

SQA Coaching for SSM
Step 4: Modify SQA Plan

Project Manager and SQA Focal add SSM Reviews to SQA Plan

BOEING

SQA Coaching for SSM
Step 5: Apply Modified Plan & Perform Reviews

Product Delivery

Project Manager, SQA Focal, and domain specialists perform selected reviews on contract deliverables.

BOEING

Assuring Quality in Outsourced Software

Roger M. Records, Associate Technical Fellow
Boeing Commercial Airplanes Group

1.0 Introduction

1.1 Why Subcontract?

It's one of the newest buzzwords – subcontracting or outsourcing. Today there is considerable interest in subcontracting software development or maintenance generated from the Information Systems (IS) community. There are predictions in IS journals that the amount of offshore work will triple in the next three years. So, what is the attraction? Probably some of the allure is just the fact that everyone is doing it. If my competitor is going offshore for software, maybe I should too. The other attraction is the perception that there is money to be saved in offshore subcontracting. India and Russia are holding a software sale! You can get 50 percent off if you buy over there! Are these claims valid? It probably depends on whose data you trust.

But there is one certainty most software shoppers will agree on – there is a definite risk associated with the decision to go to the outside. This risk can take several forms, including (but not limited to):

1. Product risk – will it work?
2. Schedule risk – will it be completed when it is needed?
3. Cost risk – will subcontracting really save the money we think we're saving?
4. Legal risk – this is a contract. Might this activity result in litigation?
5. Personnel risk – will morale suffer? Could we lose key staff by subcontracting?

The topic of this paper is, how to reduce this risk?

1.2 What About Reducing the Risk?

There are several components to a risk reduction strategy. We need to ensure we have the right project and are subcontracting for the right reasons. We need to get the right supplier – one that is competent and reliable. We also need to do a good job of monitoring the quality of both the product and the subcontracting process. This implies that we have explicitly defined what a successful project looks like.

1.2.1 First Risk Reduction Activity - <u>Select the Right Project</u>

The issues in project selection begin with current project personnel. The risks are, you might lose key project personnel or see a huge decline in morale. How do your current employees view the decision to subcontract a portion of your work? Is there a morale issue ready to surface? Most current software personnel look at offshore subcontracting

as exporting jobs. And a key question should be, what kind of jobs will be exported? If it is the work assignment that no one wants to get stuck with, the resistance will not be so great. Conversely, if the subcontracted work is challenging and a very desirable assignment, you can expect pushback. So, look for a package of work that will not be missed by your current staff. You want to make your team support subcontracting!

The kind of work that will be missed by current employees is work that leads to career development. Don't outsource your leading edge assignments. Your employees will be pleased when they can see the opportunity to acquire new skills. So, a parallel effort in your IS group should be opportunity for new training. If some of the money saved by subcontracting is used to enhance the careers of the hometown folks, the response will be much more positive. Recently our organization had an example of a legacy FORTRAN application that was subject to a certain amount of change each year. The decision to outsource the maintenance of this software, leaving the in-house developers opportunity to enhance their C++ skills was well received by folks who saw no future in analyzing FORTRAN code.

There is a second issue in project selection – defining a well bounded and well understood problem is essential for offshore development or maintenance. The risk is one of escalating cost and extending scheduled delivery. This issue will be expanded in the discussion of good requirements specifications later in this paper. Sufficient to warn at this point, if you can't find a robust problem definition and boundary, you've opened up a giant hole in which to pour subcontracting money.

The third issue for selecting the right project is to consider what happens when you export your code. If your code contains proprietary information, it is at risk offshore. Your risk is the loss of competitive advantage. In spite of non-disclosure agreements or other legal ploys, you have lost a considerable amount of control of your intellectual property. It is difficult to monitor foreign journals to ensure your algorithms are not being published. In addition, there are legal restrains on what kind of data can be exported. You need to verify that you are in compliance with U.S. law when you send data overseas. There will be a cost associated with verifying your compliance (one of many which might impact that 50 percent savings you think is on the horizon).

Finally, there is the issue of how to maintain the code that is returned to you at the contract's end. The risk is the increase in cost of continued operations. This is, in fact, a quality issue which will be addressed in more detail in the code review discussion following.

1.2.2 Second Activity for Risk Reduction - <u>Selecting the Right Supplier</u>

The attributes for the right supplier are straightforward to define, but not so easy to find. First on the list would be experience in the application domain. You don't want to spend your training dollars on your subcontractor. In fact, you really want a proven track record in the specific domain. The way to assess this experience is to request a

description in the Request for Proposal (RFP). Then you must define the experience acceptance criteria for your proposal evaluation.

Another factor in selecting a supplier is the evidence that the supplier understands the requirements and has sufficient knowledge to do the job. The bid can be evaluated for this expertise. Was sufficient thought given to create a reasonable bid? One way to make this judgement is to have your staff create an estimate for the proposal, then compare with the supplier's estimate. A large discrepancy might suggest that the supplier doesn't understand the problem, or that insufficient thought was given for an accurate bid. Either condition should raise some concern about that supplier's qualifications. An example is a project that our business group put out for bid. One of the responses gave a quote for the project total which was about three times our estimate. Our RFP consisted of five tasks, and the bidder simply divided the total into five equal parts and estimated each task with equal cost. Since we had estimated one of the tasks as about ten times the cost of the second, we concluded that this potential supplier had not thought about the individual tasks at all! Is that the kind of company you want developing your software?

A caveat that should be stated is to document the reasons for your evaluation of each proposal. You may have to supply a justification for your choice of suppliers, and documented criteria for your selection (along with the evaluation of the bidder) will ensure you are not open to charges of bias.

The final issue to consider in the selection of the right supplier is the advantage of a long-term working relationship. Your new supplier needs to learn how to work with your organization. This is an investment in training that you can recoup if there is ongoing collaboration. There is also the need for you to learn you supplier's strengths and weaknesses. You can leverage the strengths, possibly focusing your next contract to maximize their contribution in this area. But, if there is a commitment to a long-term relationship, you can create an opportunity for growth in the areas of weakness. Our organization worked with a supplier who had an obvious lack of understanding of usability testing. We tutored them about selecting test personnel with characteristics similar to our end users. We demonstrated how additional use cases could be generated from observing target audience usage. These are non-trivial problems when the supplier is overseas and has to give considerable thought to locate candidates for usability testing. Our second contract with this supplier was much more successful in front-end testing. They had benefited from our investment in their testing process. And so did we!

1.2.3 Third Activity for Risk Reduction - Monitor Quality Throughout the Project's Life Cycle

This is one of those truisms that is obvious to Quality Assurance (QA) people (so this paper is probably preaching to the choir). For many IS practitioners and for some project managers, it is not a self-evident truth! So a prerequisite to software subcontracting is to have a Software Quality Assurance (SQA) function installed and implemented. While the scope of this paper is not "how to establish your SQA function," we can offer a few suggestions if you have no documented SQA function in place. For the project about to

initiate subcontract management, our recommendations will make SQA installation more feasible.

To be successful, the SQA function must be cost effective. From experience, this means SQA must be perceived by the project manager and project personnel as being value added. It must also address the concern many project managers and developers hold, "you are piling more on my already full plate!" We will provide specific suggestions for establishing an operational SQA function in the "Third Key" section below.

With a useable SQA function in place, the next question is, "How can Quality Assurance be used to mitigate risk in outsourced work?" The remainder of this paper will address that issue.

2.0 How is Quality Assured When Software is Outsourced?

There are three key components for assuring quality in outsourced software. The first key is having the right process, the second is having the right project manager, and the third is having an effective SQA function tailored for outsourcing. For success in subcontracting, all three must be in place.

2.1 First Key – Have the Right Process.

The road to successful subcontracting has many potential potholes to fall into. We have already discussed several. You could choose the wrong project to outsource. You could select the wrong supplier. You could be subcontracting for the wrong reason. A well-defined process to follow will minimize these and a host of additional risks. Fortunately there is a voice of experience to listen to, which will not only identify additional potential problems, but also recommend procedures to safeguard the project. The Software SEI/CMM Level 2 addresses Software Subcontract Management (SSM) as one of its Key Process Areas (KPA). Even if there is no interest in SEI/CMM assessment, the SSM KPA offers insight into the main issues to consider when outsourcing software. Our business group has developed a process model that addresses the issues found in the SSM KPA.

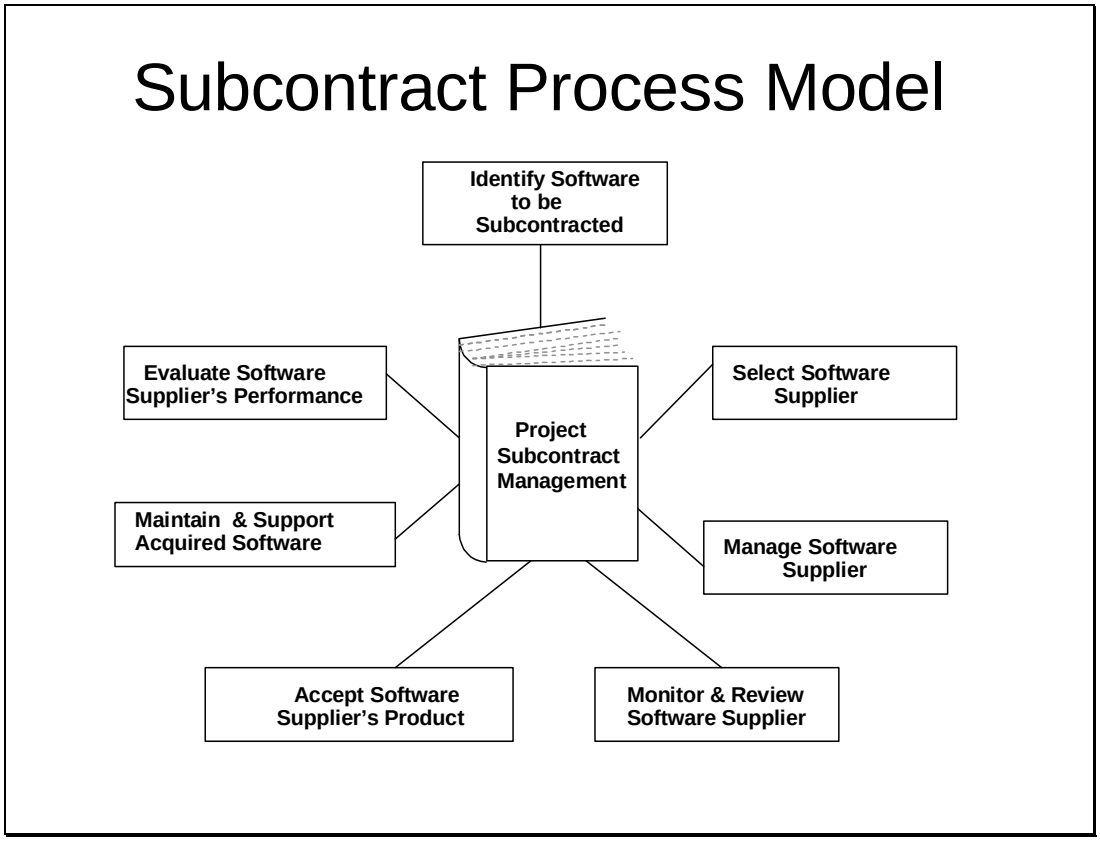


Figure 1. SSM Process Model

As shown in Figure 1, this process has seven components, which come into scope at various stages of the SSM life cycle. We have already discussed, at a high level, the component for the pre-contract phase, identifying the software to be subcontracted and the component for the contract phase, selecting the software supplier. We will re-visit each of these topics, but in the more detailed context of assuring quality.

2.1.1 Identify Software to be Subcontracted.

In this phase the project manager's prime responsibility is to prepare the statement of work input for the RFP. The goal is to give the potential supplier all the information required to create an accurate bid. The project manager must define the requirements with enough detail to produce a good estimate, yet not drift into the design (since you're paying the supplier to furnish design specs). To enhance the supplier's understanding of the desired functionality, it is very useful to supply use cases and scenarios along with requirements definition for the RFP. Any supplement that you as a developer would find useful, is probably useful to the bidder as well. For very small projects, the project manager may have sufficient background to supply all the RFP input. But most projects will require the expertise of a domain expert and a system user as well. The concept of using a team of experts will be the preferred approach to most of the tasks defined in our SSM Process. The quality of the RFP requirements is one of the SSM QA reviews that is recommended. This QA review will reduce the risk of providing poorly defined requirements that result in costly changes later in the contract cycle. Usually the project

manager and an SQA focal will be responsible for performing the reviews, but additional evaluation may involve other specialists for specific reviews.

Preparing and issuing an RFP is a task for someone other than the project manager. Most projects will find that the skills of a purchasing agent and a contracts specialist will be required for issuing and processing responses to the RFP. The project manager does have a significant role. Most RFPs are issued with a provision for the bidder to submit queries if clarification is needed. Some queries might be about payment schedules or contract language (such as nondisclosure agreements). The appropriate specialist would answer those queries. When the query involves system-related topics, the project manager or the team application specialist must respond. Usually these queries are collected and answers are distributed to all prospective bidders.

RFP queries are a source of quality improvement in the next subcontracting iteration. These queries will suggest additional questions to include on the RFP Input QA Review. At project completion, a part of the project evaluation should include analysis of the queries to see if they reveal the need for additional review questions.

2.1.2 Select Software Supplier

Our organization has found there is a savings of both time and money expended in supplier selection by eliminating suppliers not in compliance with the RFP guidelines. Rarely do we appreciate the potential supplier's innovative approach of ignoring the required response format. We want all the proposals to be evaluated on an equal basis, and that demands comparing each response section by section. There are additional considerations beyond format. It is well to document these criteria prior to proposal evaluation. Some of the selection criteria are evaluated by the contract specialist and some by the purchasing agent. The project team does play a significant role in evaluating the technical approach and domain expertise outlined in the proposal. In addition, they may be able to evaluate the bidder's interest in the project by "reading between the lines." The desired supplier not only has the qualifications to do the job, but also has an interest in working with our company, preferably over a long term.

There may be cases in which the selection is already determined, and the RFP is in reality an RFQ (request for quote). These cases of "single source" relationships may reflect a previous work relationship or a case of unique skills that only that supplier offers. The proposal or the bid quotation still needs to be evaluated using many of the same criteria used in competitive bidding. It is a good practice to prepare your own work estimates for single source bids – to be used as a reality check.

The recommended candidates for QA reviews during supplier selection include a review for your documented selection criteria, and another for your supplier selection process. These reviews will reduce the cost risk of charges of bias from losing bidders when you can demonstrate consistent application of selection criteria for all proposals according to the application of a documented procedure.

2.1.3 Managing the Software Supplier

Managing the supplier begins with a well-defined working relationship. The foundation for this working relationship is the Project Plan. The plan, prepared by the subcontractor, contains a minimum of a list of project deliverables along with the delivery schedule. One consideration for the first-time subcontract project manager is the time required to complete the inspection of each deliverable. This time is significantly greater than a corresponding QA review for a software developer in next cubical. Communication by e-mail or telephone across the ocean (with a person speaking English as a second language) is time consuming. There is a time difference of nearly 12 hours. Your workday is the supplier's nighttime. Communication usually has a one day cycle time. Clarification of issues takes time! It is rare to complete the review of a document in less than a week.

There should also be a description of the roles and responsibilities for both the supplier and the customer. Giving precise definitions of both roles and responsibilities will reduce the risk of costly negotiation (where all that finger pointing takes place) or even litigation later in the contract. The Project Plan QA Review is considered essential. This review addresses issues such as the completeness of the list of deliverables, the feasibility of the schedule and the completeness of role and responsibility assignment. Performing this QA review will reduce the risk of schedule overruns as well as the cost associated with negotiating corrective actions.

The Project Plan should also include references to key processes that affect the working relationship between the project group and the supplier. The three processes that are referenced in the plan are all owned by the project group, but contain information relevant to the supplier. The first is the Software Subcontract Management Process, which is a documented procedure for the SSM Process Model in Figure 1. The other vital processes are the Change Management Process and the Corrective Action Process. Each of these processes comes into scope during quality assessment of subcontracted deliverables. While it is outside the scope of this paper to describe each of these processes, a diagram for a candidate process is included in the Appendix. Including the reference to these procedures in the Project Plan ensures that the supplier is aware of the procedures that will be followed during the duration of the contract.

2.1.4 Monitor and Review the Software Supplier

Following agreement on the Project Plan, the contract is initiated with production and delivery of contracted deliverables. There are two primary activities in the production phase: Project Tracking and QA Reviewing for products delivered. These activities require monitoring by the project manager.

The Project Plan defines a set of contracted deliverables and their delivery schedule. Monitoring delivery of these products is an ordinary project management function, and is conducted using a procedure nearly identical to any other schedule monitoring. Deviations from the established schedule are documented and reported in normal project status format. Corrective actions for deviations are negotiated between the project

manager and the offshore subcontract project manager, using the Corrective Action Process cited in the Project Plan. When the delivery of contracted products is back on schedule, this new status is reported in the next status report. When the Corrective Action Process is performed, we recommend a QA review of this process to insure compliance. Following the documented procedure agreed to in the Project Plan reduces the risk of additional contract charges being levied by the supplier.

The second activity, QA review of delivered all contracted products, is the foundation for successful subcontracting. The principle is simple: **Verify quality as early and as often as possible**. Beginning with the Project Plan, perform a Quality Review on every product received from the supplier. This is a significant investment of project resources. Our organization's experience with six completed subcontracts in software development shows the time expended is about 5 percent on Pre-contract activity, 20 percent on ordinary project management, 15 percent on Process QA, 55 percent on Product QA and another 5 percent on Project closure activity. While these data varied from subcontract to subcontract (and all the subcontracts were small), we find nearly 60-80 percent of the time a project manager spends on subcontract activity is quality related. This includes performing quality reviews, documenting QA findings, communicating the nature of defects found to the supplier and verifying the fixes supplied have repaired the defect. Our organization performs quality reviews on every intermediate deliverable including the Requirements Specification, the Exploratory Prototype (if one is created), the Design Specification, any sample output screens or documents, and the test plan with test cases. With every quality review performed we reduce the risk of having a final product delivered with missing functionality.

There are two lessons learned in the monitoring and review activity that saved significant project resources:

> Reuse current checklists for each of these products by tailoring them for the subcontract environment. If you don't have a current checklist for one of these products, borrow one from a peer group within your company or find a sample in the literature or on the web. Tailoring simply involves examining each question on the checklist for relevance to you current subcontract and the application domain. Omit or revise irrelevant questions and add new questions appropriate for this specific subcontract.
>
> The second lesson is to use a team for both tailoring the checklists and performing the reviews. Our team consists of the project manager, the SQA focal and additional personnel with specific skills and experience for the review being performed. For example, one test of a requirement that is well defined and robust is its testability. If you can define a test for the requirement, you can probably define the completion criteria and probably a testing scenario. So, use a testing specialist for the Requirements Specification review. Document the completion criteria and the scenario for use in the Test Plan and test cases. This is not time wasted, since the test plan will need to be defined at some point. If the Requirements Specification is well defined, a test plan can be generated. In a

similar case, a well-written Design Specification will produce a set of test cases with boundary conditions and other variables documented. When it comes to code reviews, enlist a good programmer as a team member to assist the project manager and SQA focal.

As the QA reviews are defined, they should be added to the current project SQA function. We define this function with more detail in section 2.3.

2.1.5 Accept the Software Supplier's Product

Product acceptance is probably one of the most familiar activities. Most of the project's current acceptance testing practices will be reusable for the subcontracting domain. The supplier is ready to deliver the source code, the executables, the user documentation and the training required to install, initialize, run and maintain the contracted software. The project manager's job is to ensure everything functions per the contract. The Acceptance Test Procedure will be in scope and will call out various QA reviews. Onsite delivery is essential. The offshore subcontract project manager needs to be present for processing all the test findings. It is also essential that both the Acceptance Test Plan and the Suite of Test Cases be prepared and previously passed the QA review prior to the arrival of the offshore personnel. The focus is then on identifying defects, with corrective action taken in real time. Testing defects are logged, transmitted offshore by the (now resident) subcontract manager for correction, and the new (corrected) version(s) is installed for re-testing. Depending on project size, one or two weeks of acceptance testing will be sufficient for most system testing. Our organization's policy is to contract for a 90-day (no cost) post contract warranty for all software including user and systems documentation. As in each of the previous phases, QA findings for product acceptance are logged in the SQA Findings Log, and are tracked to resolution by the project manager. Potential QA reviews concurrent with acceptance include source code reviews, user manual review, as well as process compliance reviews on both the testing and quality assurance procedures. These reviews address the risk of accepting software with undiscovered defects.

2.1.6 Maintain and Support Acquired Software.

Preparation for software maintenance begins with the subcontractor's training delivery. Beyond functional testing, the installation procedure, the configuration management of the new products, and the provision for product dissemination need to be addressed. If these functions are part of the normal project operation, some tailoring for the subcontracting activity will probably be required. The QA reviews that are usually performed will be tailored as well. The recommended reviews include a QA review on the delivery process and any product reviews remaining from acceptance. It is helpful to document your delivery process for subcontracted work. A sample process is included in the Appendix.

2.1.7 Evaluation of the Supplier's Performance

This brings us to the final activity in the subcontracting process, project completion. There are several tasks that will ensure that quality is maximized in future subcontracting activity. Final payment to the subcontractor is one important task. Care must be taken to verify all the contracted deliverables are completed. Performing all the QA reviews discussed previously in this paper makes this verification a certainty! A notification of satisfactory completion is generated for the purchasing agent or Accounts Payable Department. When the contract is completed, project closure should include an evaluation of the supplier by the project manager. A portion of this evaluation is a list of lessons learned. Both the lessons learned and the supplier evaluation should be made available to other groups within the company who are considering subcontracting activity. It is suggested that a project closure procedure be defined prior to initiating a contract. This will ensure important issues will all be addressed while the supplier is still available to project personnel. A sample procedure is included in the Appendix. The QA review for this final report will mitigate the risk of having to re-learn lessons and use resources for additional corrective action.

2.2 The Second Key – Have the Right Project Manager

The choice of a project manager is vital for successful subcontracting. As described the in the last section, most of the activities in the subcontracting process were focused on the project manager. And the principle for this choice is this: Subcontract Management is not identical to project management. Project management is centered on the triple constraint – cost, schedule and quality. Many subcontracts are built on fixed price contracts, so cost is a constant (ignoring changes), and the schedule is also fixed in most contracts. This leaves one significant issue – QUALITY. The risk is that changes will occur and that schedules will slide, in spite of contract stipulations. The challenge for the project manager is to deliver the contracted deliverables, on time, within budget, and still obtain quality. As we have previously stated, once the contract is in place, about 60-80 percent of the effort expended by the project manager is in some way connected to verifying quality for each deliverable at its scheduled time to mitigate this risk. So, the project manager who would manage a subcontract better come to the job with extensive SQA experience, or get ready to climb a steep learning curve for SQA.

A second difference between subcontract management and normal project management is that the project manager is a people manager, whereas the subcontract manager is not. Experts suggest people skills are primary in importance for successful project management. But when the work is performed offshore, the subcontract project manager never sees the people doing the work. Instead, the primary skill becomes the ability to communicate with the offshore subcontract manager (who, by definition, resides on another continent.) To be more specific, even verbal communication skills are mostly utilized in the project definition activity and again at product delivery (when the project manager is face-to-face with the supplier). For offshore collaboration, written communication skills are the success factor.

The project manager who will become a manager of a subcontract needs to be familiar with the application domain, or be closely teamed with application domain specialists.

2.3 The Third Key – SQA: Review Early; Review Often; But Review the Right Stuff!

Since we've discussed the importance of quality assurance in both of the first two keys to successful subcontracting, it should come as no surprise that the third key to success is Software Quality Assurance. The best advice is to make sure you are performing SQA reviews on the right stuff! There are both products and processes to consider, and selecting the right review will produce the best return on investment (ROI). Even as the contract is being defined and responses are being collected, the quality issues will determine the success or failure of the entire subcontracting activity. So, what is the most

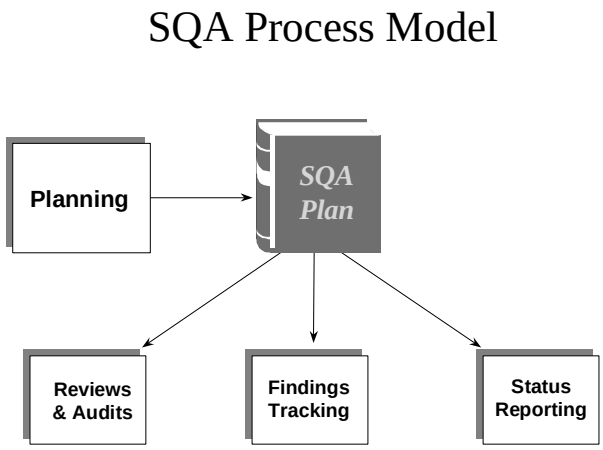


Figure 2. SQA Process Model

important SQA review? Is it a product review or a process compliance review? The answer is, "Yes!"

Before the project group can implement a successful subcontract management process, they need to have an operational software quality assurance process in place. While it is beyond the scope of this paper to define the SQA procedure, a brief overview is provided. As shown in Figure 2, the foundation of the SQA procedure is the SQA Plan. It is created by a planning activity that defines the SQA reviews to be completed for the lifecycle of the project. The SQA Plan also includes a description of the procedure for recording and tracking review findings. Finally, the plan defines the reporting of SQA status to management. These four components of the SQA function are a prerequisite for the SQA activity performed during each phase of the subcontract.

2.3.1 The Pre-Contract Issue – Highest Leverage Product Review

With an SQA function in place, the project can initiate the subcontract process. The first task is to select the software to be outsourced. The primary product in this task is the RFP, and the most important component of the RFP is the requirements for the project. Producing a good set of requirements is a non-trivial task. The goal is to produce a set of

requirements that are so well defined that the bidders for the RFP can produce an accurate bid. The consequence of failing to do a good job on requirements definition is either a large number of queries from potential bidders, or (much worse) many conflicts later in the contracting life cycle. With this in mind, many project managers experienced in subcontracting would state that a QA review for the RFP Requirements is the highest leverage review to perform. Well-defined requirements are the foundation for an accurate proposal, which is the basis for a contract with minimal change potential.

The topics of requirements engineering and requirements acceptance have been covered in many publications. Our goal in this section will be to provide a methodology for creating a quality assurance review for the RFP requirements, given their importance. It begins with defining the quality attributes for software requirements. These attributes might include (but are not limited to):

1. Lack of Ambiguity
2. Feasibility
3. Consistency
4. Testability
5. Completeness

Ideally, the QA review for RFP requirements will include questions to verify each of these attributes has been evaluated. Fortunately, there are published examples of checklists for requirements that address these attributes, and more. Obtaining one of these checklists and tailoring it for your application domain is a good first step toward initiating your subcontracting process with a quality contract.

There are enhancements for the RFP Requirements that will improve the potential bidder's understanding. One of these is specific use cases or scenarios. By including a use case that would describe the input and output, along with a description of the data transformation, the bidder's understanding of the requirement becomes more concrete.

2.3.2 The Post-Contract Activity – Perform High Leverage Product and Process Review

An operational SQA Function will provide both product and process reviews. The same can be said for performing QA in the subcontracting domain. It is not the intent of this paper to define and describe all the QA reviews that have potential within the subcontract domain. Rather, several candidate reviews are recommended for both contract deliverables and associated subcontracting processes. The approach is to determine which of the candidate reviews carry the highest potential value. It is better to perform a few high-leverage reviews than many QA reviews with low return on resource invested.

Throughout this paper we have recommended QA reviews for both products and procedures. A summary of our recommended list of contract product reviews include:

1. RFP Requirements
2. Project Plan

3. Software Requirements Specification
4. Design Document
5. Source Code
6. Acceptance Test Plan
7. Acceptance Test Suite
8. User's Manual
9. Subcontractor's Status Report
10. Project Final Report

The RFP Requirements, Acceptance Test Plan, Acceptance Test Suite and Project Final Report are products created by the project. The subcontractor creates the remaining products as contract deliverables. There may be other products for a specific contract, such as an exploratory prototype, which would be candidates for additional QA reviews.

The summary of our recommended list of process reviews include:

1. Change Procedure
2. Corrective Action Procedure
3. Acceptance Test Procedure
4. Delivery Procedure
5. Project Completion Procedure
6. The Subcontract Management Process

The purpose of a process review is to verify project compliance with "the way we do things." Each of these procedures and processes is owned by the project. There may be additional processes or procedures, such as a vendor selection procedure, which would be candidates for additional QA reviews.

Attempting to perform all 16 QA reviews is not recommend for a small contract or for your first subcontracting activity. You can choose those reviews that have the highest potential ROI. The RFP Requirements review is recommended as a high leverage QA review. Since incomplete requirements are frequently cited as the reason for project overrun or project failure, perhaps a second review might be on the supplier's Software Requirements Specification, or on the following deliverable – the Design Document. The goal is to collect facts and data that can be used to verify there is quality built into this project at each step, not just in the final delivery. Waiting until acceptance testing to discover the absence of quality is unacceptable (and costly!)

3.0 Finally – A Few Words About Cost and Schedule Overruns

If you've read this far, you'll be happy to learn the best has been saved for last! We live in a culture where contract cost and schedule overruns are not only common, but also expected! How can this be avoided, or at least minimized, in software subcontracting?

Quality Assurance is the mandatory component! It is the essential activity to protect not only quality, but scheduling and cost as well. It begins with your preparation. The best advice is to reuse wherever possible. Consider the quality reviews required. If you have

an SQA function in place, examine each of your current checklists to see if any might apply to the list of product or process reviews listed above. Most checklists that might apply to subcontract management will need to be tailored to fit this new environment. You are no longer the developer, now you are the customer. Your checklist must have a customer perspective. To save additional resources, minimize the amount of tailoring. Your team of project manager, SQA focal and selected domain specialists should accomplish the tailoring. This team can be efficient in identifying modifications or generating new review questions.

If you do not have a candidate checklists in your current SQA function, there are at least two additional sources. Check with a peer group to see if they have additional candidate checklist. If your software group has a process assets library (PAL), check this source for additional candidates. Finally, the literature, including the web, is a source of initial checklist candidates. One Web site that offers a variety of examples for use by the public is http://www.processimpact.com/. So, cost is minimized by reuse and tailoring. As a last resort, you can invent a new checklist.

In addition to QA checklists, you'll need additional processes. These processes are available from the same sources as the checklists, and will require similar tailoring. Looking at the sample procedures in the Appendix, the Subcontract Change Procedure has been tailored to show an amended contract, which would not be part of an ordinary change procedure. In a similar manner, the Subcontract Corrective Action process might contain a reference to the Legal Department, which is not a part of the usual corrective action process.

4.0 Lessons Learned in Subcontracting

There are several valuable lessons learned that have been stated in the preceding text. These will be summarized at the end of this section. A few additional insights may serve to help project managers new to subcontracting.

Invest in a long-term working relationship with your supplier. If outsourcing is new to you, working in your business culture is new to your supplier. Begin this working relationship with a goal of continued success. This will require good communication of expectations, with timely and candid feedback. Most offshore contractors put a high value on customer satisfaction. Your investment of informal training in "the way we do things" for you supplier will pay large dividends in future work.

Promptly completing your QA reviews on delivered products and sending the QA findings to the supplier immediately will establish your quality standards. Obtain an estimated completion date for all bug fixes, and monitor these commitments.

Testing early, even on partially completed products, will make acceptance testing much more successful. If your contract calls for the supplier to perform unit testing, provide test cases and request output for specific modules to ensure the design is being coded functionally correct. For complex or leading edge work, exploratory (throw-away)

prototype evaluation is used an very early in the contract to validate the understanding of requirements. In one contract, our organization created the prototype as a kind of "electronic specification;" while for other contracts the supplier creates the prototype. In one case we used evolutionary prototyping with evaluation at specific feature points as the contracted developmental methodology.

This paper began with a discussion about the risk associated with subcontracting software. We suggested three specific strategies for reducing risk. Our last "lesson learned" is to recognize the value in performing the normal risk assessment for the proposed project. Our usual approach for small project risk assessment is to provide a list of common risks and ask the project manager to add domain specific risks to the list. For the project manager about to take on a subcontract, we have added some common risks for subcontracting. The tailored list looks like this:

Common Software Risks

- Feature creep
- Personnel shortfalls
- Unrealistic schedule and budgets
- Inadequate design (Developing the wrong user interface, wrong functions)
- Gold-plating requirements
- Continuing stream of requirements changes
- Contractor failure
- Silver Bullet Syndrome
- Real-time performance shortfalls
- Friction between developers and customers

Software Risks Associated with Subcontracting

- Misunderstanding due to communication in English
- Potential for loss of proprietary information
- Resentment from in-house staff
- Dependence on supplier and supplier's tools
- Problems in knowledge transfer for domain knowledge
- Loss of project control due to coordination with support groups such as Legal, Purchasing and Subcontract Support
- Contractor's software culture may produce variation from the project's standard methodology
- Coordination of Configuration Control (two CM Systems)
- Current SQA function inadequate for increases in SQA activity

Candidate risks are selected, prioritized, and evaluated to develop plans for risk mitigation. A QA process review should be performed to verify compliance with the risk procedure.

There are additional lessons cited in the previous sections of this paper. Their summary follows:

1. Invest in pre-contract requirements understanding to avoid mid-contract conflict
2. Communication across the ocean requires time

3. Care must be taken in selecting the project for subcontracting
    a. Morale of in-house employees may suffer
    b. "Software sale" mentality – consider extra costs of outsourcing
    c. Export restrictions may come into scope
4. Select a project manager with extensive SQA experience
5. Begin the subcontracting with an SQA function already installed up front

Getting Started – Where to Go from Here?

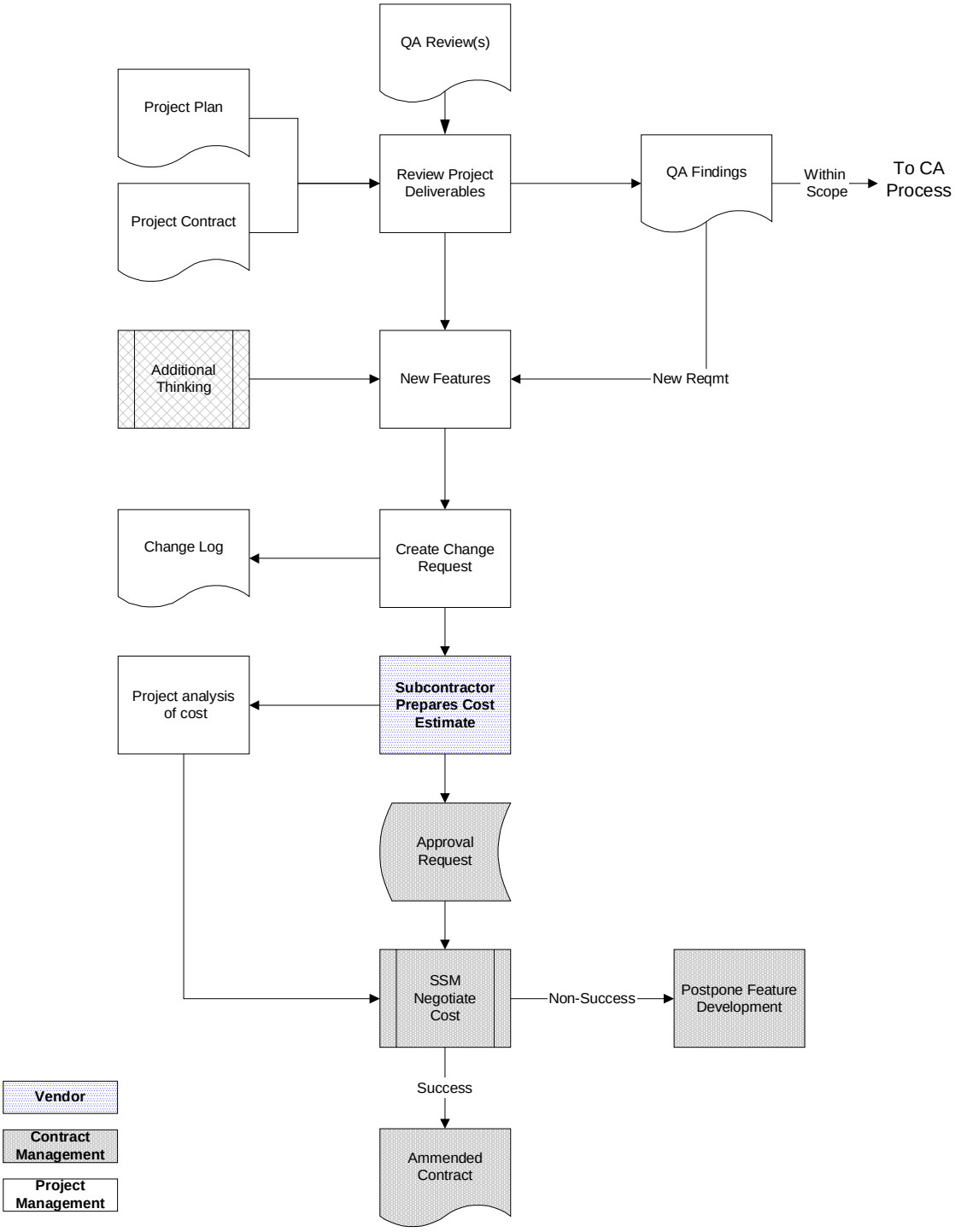
If outsourcing software is in your near future, what is the sequence of events or activities that will get you started? Here's our recommendation:

Step 1. Identify the support organizations, including Legal, Purchasing, and (possibly) the group with contacts for potential suppliers.
Step 2. Document your Subcontracting Process.
Step 3. Identify the QA reviews required for this subcontracting process and add these to your SQA Plan.
Step 4. Implement your Subcontracting Process and perform the QA reviews required.
Step 5. Add or tailor additional processes as required, with the associated QA reviews.
Step 6. Obtain training, coaching or mentoring from a peer with SSM experience, from an SQA subject matter expert or from outside suppliers, if available.
Step 7. Apply the knowledge you have acquired from this paper as you traverse your subcontracting journey.
Step 8. Document any opportunities for improvement as you proceed, and implement those improvements before your next subcontracting project.
Step 9. Prepare to mentor or coach one of your peers as they initiate SSM.
Step 10. After a few iterations of this procedure, publish your insights for others.

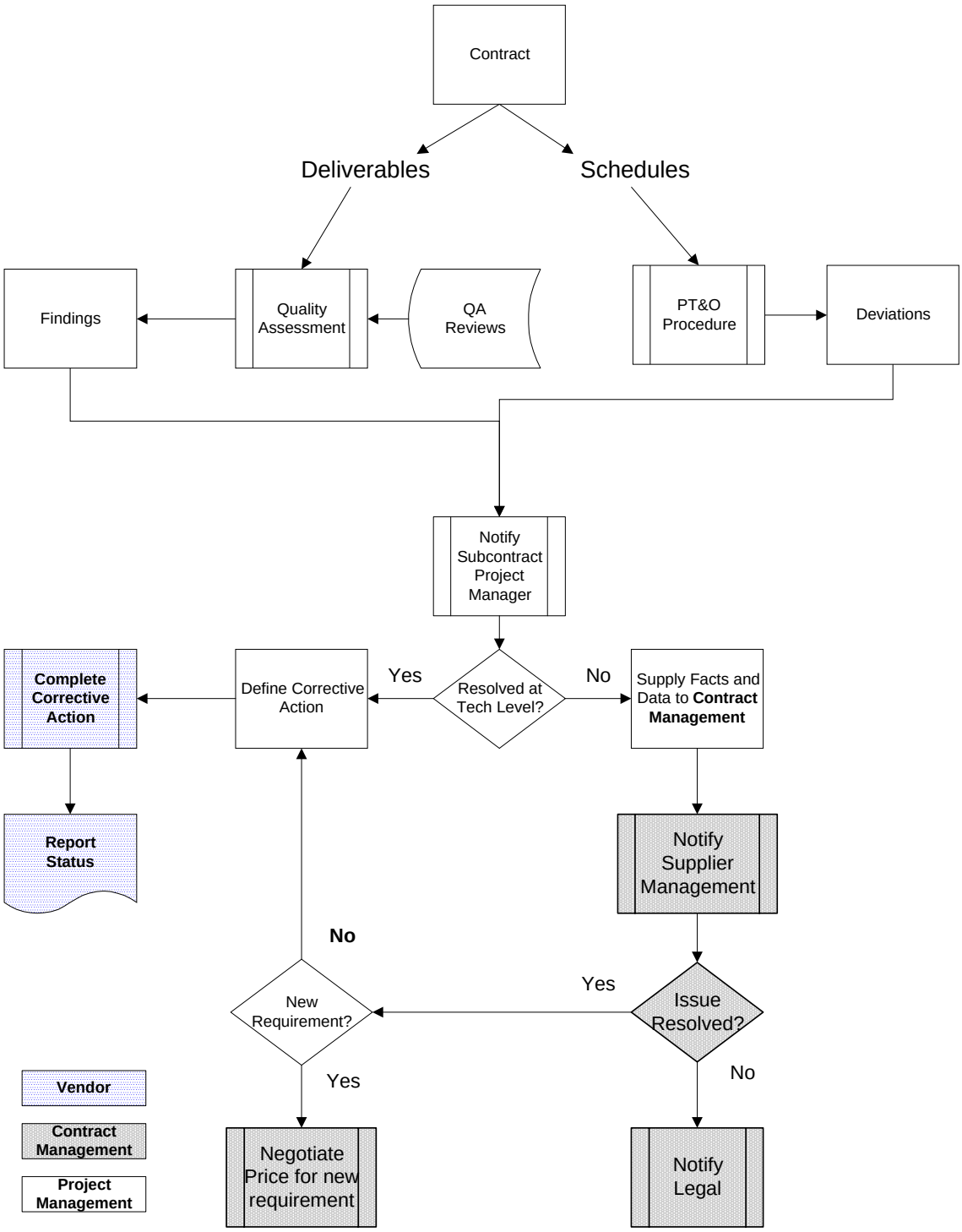
Following Steps 1-8 through a series of subcontracts allows a project to start with a small number of QA reviews and grow incrementally with each new contract. Using this approach, a mature SSM process can be developed with maximum usability and minimum investment of resources. Incrementally adding to list of QA reviews until you have an adequate number is the best assurance of receiving quality in your subcontracted project deliverables.

Appendix - Procedures

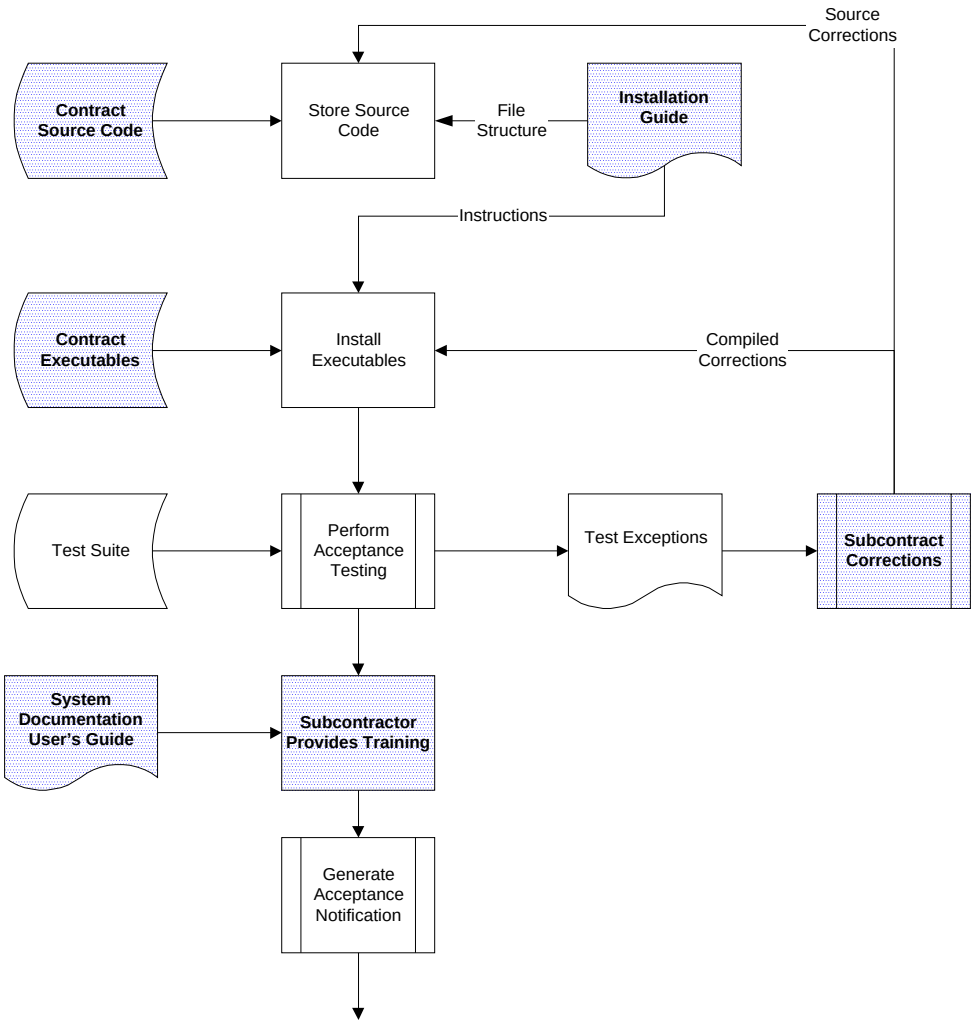
Subcontract Change Procedure



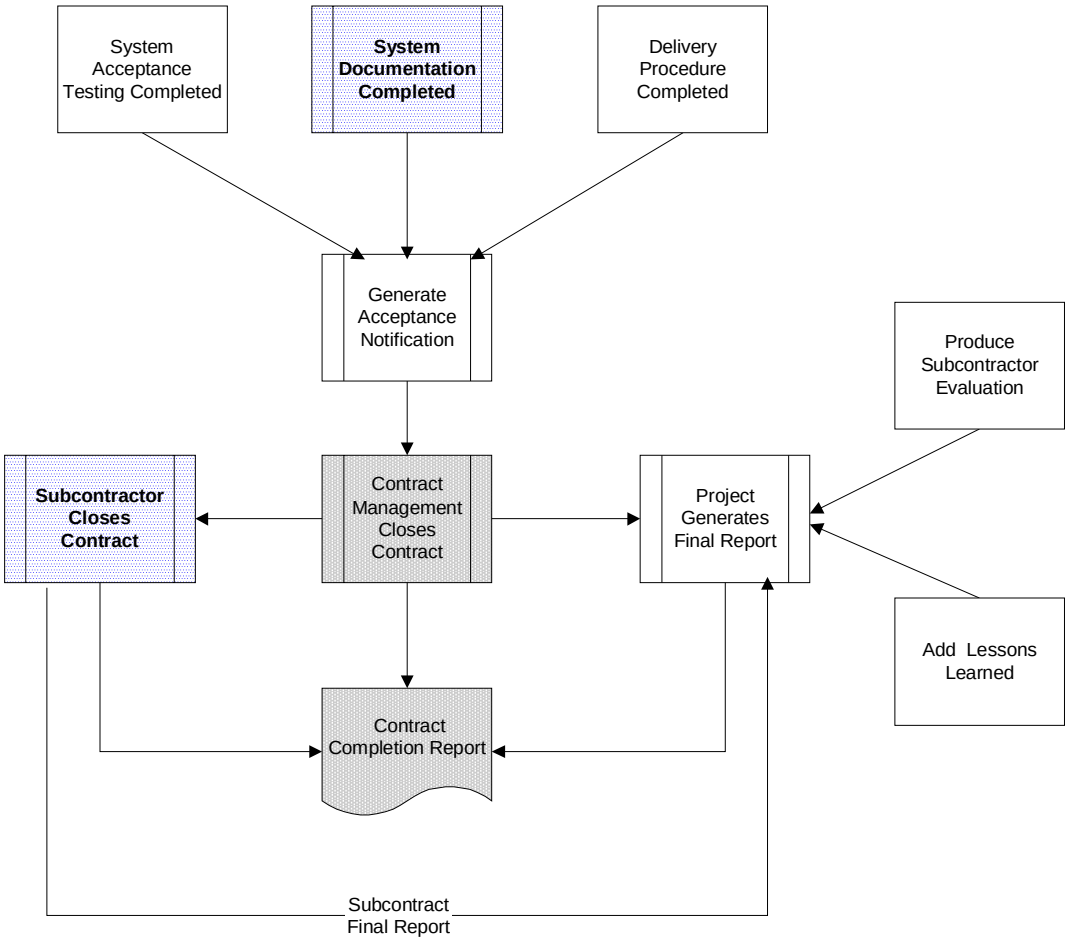
Subcontract Corrective Action Procedure



Subcontract Delivery Procedure



Subcontract Project Closure Procedure



QW2001 Paper 4A1

Mr. Henk Keesom , Dr. John Musa
(Ortho-Clinical Diagnostics)

Using Test Data to Calculate Software Reliability Growth

Key Points

- Software Reliability Engineering
- CASRE - Computer Aided Software Reliability Engineering
- Use of Product Development Historical Data

Presentation Abstract

This case study describes how one company used existing product development test history to calculate the software reliability growth of three embedded real-time software products. It includes techniques that were used and can be used by others to do the same calculations with their existing product development test data. It also discusses techniques to produce better data and thus better software reliability information in the future.

The techniques described are useful to embedded product software developers and their organizations as many have access to extensive product development test performance data. They can use this data and these techniques to determine their software reliability growth during testing and their product's current software reliability. In addition, suggestions are provided to provide better future tracking of software reliability.

Ortho-Clinical Diagnostics develops and manufactures blood analyzers for professional use in Clinical Chemistry Laboratories - containing custom developed embedded real-time control system software. The three products used as examples have been in field use and software upgraded with additional features and functions for over ten years. Thus these products provide an extensive amount of product development test history of software reliability.

The existing product development test history was used with CASRE (Computer Aided Software Reliability Engineering). CASRE is a widely accepted tool used to calculate and graph software reliability. CASRE is available on the CD-ROM that is provided with the Handbook of Software Reliability Engineering, by Michael R. Lyu.

The normalized MTTF from the three products that were used are presented and show the software reliability growth over time - and that with new upgrades

(software versions) there were short term decreases (as one would expect with new additional features) in the reliability.

Techniques for converting product development test history data from calendar time to execution time are provided. This data was in calendar time but it required conversion to an approximate execution time before CASRE would produce meaningful results. Methods for handling data sets larger than CASRE can handle are included. Steps are provided to take CASRE output and produce graphs not included in CASRE.

Recommendations on how to enhance test data in the future to make calculations of software reliability more predictive are described.

About the Author

Hendrik J. Keesom is a manager of Software Verification and Validation for Ortho-Clinical Diagnostics in Rochester, NY. Keesom is a software verification and validation engineer and software engineer with over 20 years of real-time embedded software and hardware experience. Keesom is one of the co-editors of the HL-7 Chapter on Clinical Laboratory Automation and the NCCLS standard: Laboratory Automaton: Communications with Automated Clinical Laboratory Systems, Instruments, Devices , and Information Systems.

John D. Musa is one of the creators of SRE, with more than 30 years varied and extensive experience as a software development practitioner and manager. Principal author of the highly-acclaimed pioneering book Software Reliability and author of the practical Software Reliability Engineering, Musa has published more than 100 papers on SRE. Elected IEEE Fellow in 1986 for many seminal contributions, he was recognized in 1992 as the leading contributor to testing technology. His leadership has been noted by every recent edition of Who’s Who in America and American Men and Women of Science. Musa, widely recognized as a leader in SRE practice, initiated and led the effort that convinced AT&T to make SRE a “Best Current Practice.” Musa has helped a wide variety of companies with a great diversity of software-based products deploy SRE. He is an experienced international speaker and teacher (over 200 major presentations) A founder of the IEEE Technical Committee on SRE, he is closely networked with SRE leaders, providing a broad perspective.

Using Internal Product Development Test Data to Calculate Software Reliability Growth

Henk Keesom, John D. Musa

Quality Week 2001

Introduction

- Ortho-Clinical Diagnostics develops and manufactures blood analyzers for professional use in Clinical Chemistry Laboratories - containing custom developed embedded real-time control system software.
- This case study demonstrates how Software Reliability Engineering was applied in the bio-technology field and can be applied elsewhere as well.
- We describe how existing product development test history was used to calculate the software reliability growth of three embedded real-time software products.
- This case study shows that existing product development test data can be used now.
- It includes techniques that were used and can be used by others to do the same calculations on their existing product development test data.
- It also discusses techniques to produce better data and thus better software reliability information in the future.

Uses of the Project MTTF Histories

- The study of existing Project MTTF histories has helped determine data collection improvements needed.
- The existing MTTF histories can also be used to guide testing by :
  - Identifying reliability status early in test and allowing early action to be taken.
  - Helping to establish a release criteria (failure intensity objective).

SRE Background

- Software Reliability Engineering is a proven standard, widespread best practice in industry.
- Software reliability uses many of the terms and methods used in hardware reliability
- Through the use of software failure data and modeling tools, software MTTF can be determined.
- Software reliability growth can be measured during a software project and can be used to guide testing.
- Software reliability engineering techniques help projects find software failures more quickly (and thus fixed earlier).

CASRE Background

- CASRE - Computer Aided Software Reliability Engineering
- This tool takes failure data (existing product dev. test history), performs modeling to determine the MTTF growth over time.
- CASRE is a widely accepted tool used to calculate and graph software reliability and is available on the CD-ROM with the *Handbook of Software Reliability Engineering*, by Michael R. Lyu.
- Although there are a number of models that are included with CASRE, only 2 are recommended by Musa in his book *Software Reliability Engineering*:
  - The Musa-Basic model models a system that has finite failures at infinite time
  - The Musa-Okumoto model models a system that has infinite failures at infinite time.
  - The two models are at the extremes (finite/infinite) and help bound the model range.

Test Data Collection

- Software product development test data of software failures was extracted.
- The relevant information included:
  - Calendar Date/Time entered - execution time was not available
  - Priority - High and medium only
  - Status - exclude "No Change" (i.e. No Change implied that no failure was found)
  - Subsystem - Software only - other subsystems were tracked and needed to be excluded.
- The data was filtered and sorted by Date and Time and reformatted for the CASRE (Computer Aided Software Reliability Engineering) tool as follows:
  - Failure number
  - "Execution" Hours since previous failure

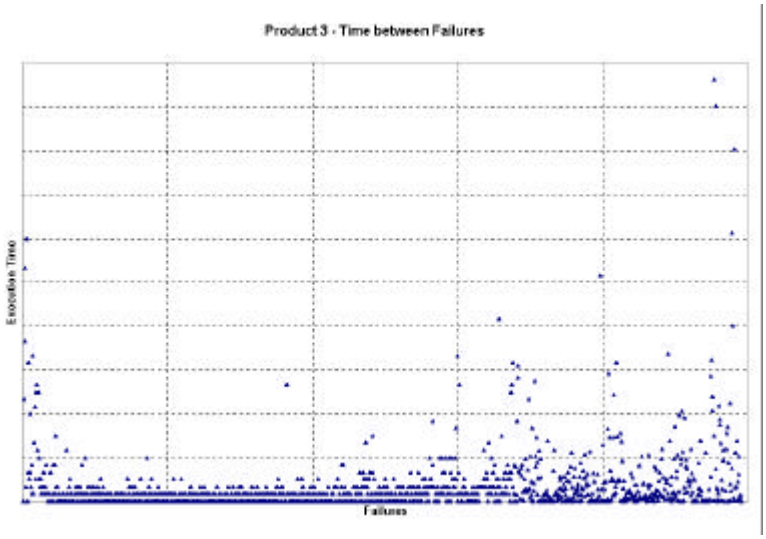
Calendar Time to “Execution Time”

- Calendar Time was converted to “execution time” by removing idle periods (nights and weekends).
- The hours since previous failure was calculated by subtracting the current date/time from the previous data/time based on a 50 hour work week.
- So, for example
  - a failure recorded at 5 p.m. on Friday
  - a failure at 8 am on Monday is considered only 1 hour apart
  - (rather than calendar time of 53 hours (24+24+15))
- We also improved the precision of the data by re-computing the failure intervals in hours to three significant figures.
- These two measures substantially improved the quality of the data and the results we could draw from it.

MTTF Data Presentation

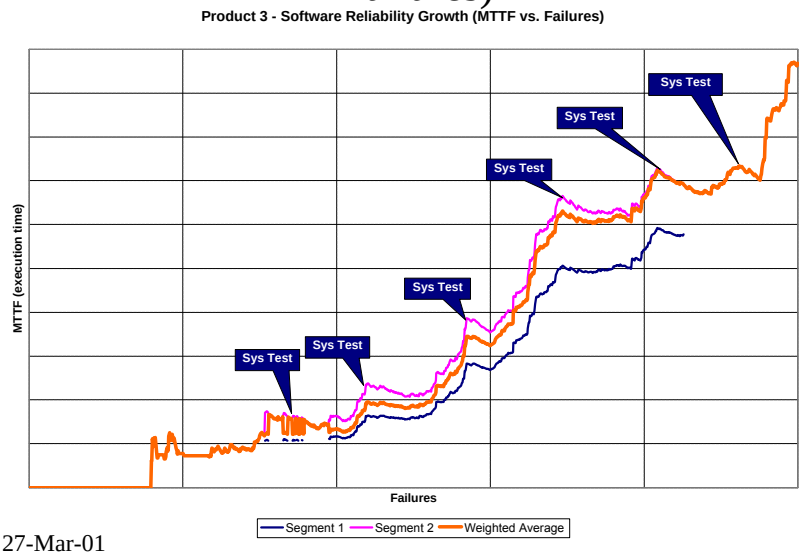
- Initially, both models were used, but after discussions it was decided to only use the Musa-Okumoto model because the test data collected was against development failures of software that was being upgraded - i.e. it was a system that has infinite failures at infinity (because it is always under change).
- The following is presented for product 3:
  - Execution Time between Failures vs. Failures - the raw data used as input to CASRE
  - MTTF vs. Failures
  - MTTF vs. Execution time
- This technique was used for all three products.

Execution Time Between Failures



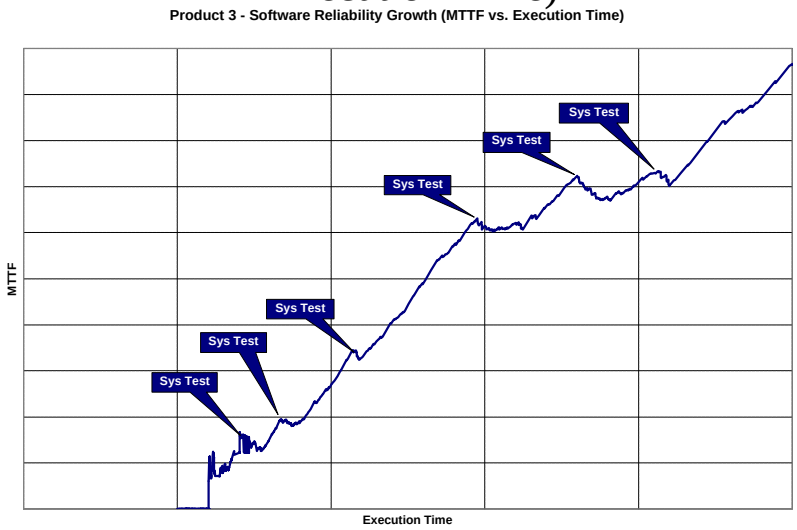
27-Mar-01 9

Software Reliability Growth (MTTF vs. Failures)



27-Mar-01 10

Software Reliability Growth (MTTF vs. Execution Time)



MTTF Data Analysis

- The failure intervals experienced by OCD were compared with Musa's experience on many other projects.
  - Product 1 data is consistent with data from other projects,
  - but the early data on product 2 and especially product 3 showed many long failure intervals that are unusual.
  - One possible explanation is that system test had not really started when the early data was collected. If so, data prior to system test should be removed from the analysis.
- We looked for ways to improve the existing data so that we could draw the best information possible from it.

MTTF Data Analysis-2

- The ideal is to record failures at the time they actually occurred in execution time or natural units.
  - The existing data is recorded in calendar time of the reporting of the failure.
  - The use of calendar time gives MTTF results that are a fairly constant multiple of the true MTTF when taken over the entire test period.
  - However, it can distort short term (and hence early) results and add considerable noise to them.
  - Also, on most projects, the calendar time to execution time ratio is typically large at start of test, decreasing by mid-test to a number that approaches the ratio found at the end of test.
  - Thus the use of calendar time yields overly optimistic estimates of MTTF during early test.

Data Analysis Observations

- Calendar time translates to approximate execution time by the 168 (7x24) hours in a week to 50 (5x10) in the work week ratio - over large data.
- MTTF results for these products are a rough approximation .
- The CASRE Model is not able to converge at the early failures - including start of System Testing.
- The CASRE Model does not converge in all situations
- The data set chosen for CASRE determines the reliability growth curve.
- Product 3 Saw-tooth pattern correlates start of system test on versions of Software.

Future Data Collection and Analysis

- Develop the operational profile and use it to drive development - Operational Profile and Load Testing are part of Software Reliability Engineering.
- Use random sampling of test cases; avoid testing feature by feature.
- Start collecting data at the beginning of system test
- Record actual time of failure.
- Record to the nearest minute (vs. hour) or natural unit (of similar granularity).
- Record product usage over time.
- Record the execution time.
- Use a weighted average scheme for more than 1000 data points to workaround the limitation of CASRE.
- Use the Musa-Okumoto model for analysis

CASRE - Procedure

- The following procedure is a slight modification of the standard CASRE procedure described in Appendix F.3 of *Software Reliability Engineering*.
- 1. Follow the standard CASRE procedure, Steps 1 and 2
- 2. For each data range you have selected:
  - A. Click on Model, then Select data range. Set the parameters as follows:

	Lower range run	Upper range run
First data point	1	N-999
Last data point	1000	N
Parameter estimation endpoint	1	$N_L + 1$

  - (above shown for 2 ranges, analogous for more)

CASRE - Procedure - 2

  - B. Click on Model, then select and run module. Double click on Musa-Okumoto model, click on Run models.
  - C. Click on Results and then Model results table.
  - D. The next step predictions are the MTTF history. If a prediction is missing, approximate it by dividing elapsed time by number of failures. Caution: If first data point > 1, you must also add the elapsed time from the previous data points.
- 3. Export the MTTFs from all ranges to a spreadsheet for further analysis.

CASRE - Procedure - 3

- 4. Using Excel set the data up as follows:
  - Column A is the failure number
  - Column B is the Hours since the last failure
  - Column C is the Next Step Prediction for the first range
  - Column D is the Next Step Prediction for the second range
  - Column E is the weighted average per the above algorithm
  - Column F is the model prediction - NOT USED.
  - Column G is the cumulative execution hours
  - Column H is the cumulative execution months
- Note: Columns A, B, C, D, F, G come directly from CASRE outputs. Column E uses the above weighted algorithm with the addition that if the model does not converge for a given data point that it uses the previous converged value. Column H is the number of hours since failure converted to months.

CASRE Workarounds

- CASRE is a Windows V3.1 compatible program
- Under certain circumstances, you will receive a message "Program Error - C RUNTIME ERROR" and have to restart CASRE.
  - The cause appears to be trying to divide by zero. It occurs when the first data point and parameter estimation endpoint are equal.
  - Never record two failures occurring at exactly the same time.
- Another problem with CASRE occurs when it sees a blank line.
  - Eliminate occurrences of 2 consecutive paragraph marks (carriage returns).

QW2001 Paper 4A2

Mr. Erik Simmons
(Intel Corporation)

Product Triage: A "Medical" Approach To Predicting And Monitoring Product

Key Points

- The medical field of trauma care provides interesting and valuable metaphors for product quality.
- By adapting the tools and concepts used within trauma care to product quality rather than patient outcome, a new and disciplined approach to predicting and monitoring product quality emerges.

Presentation Abstract

The medical profession has long relied on triage to predict which patients require enhanced care or monitoring based on presenting risk factors or indicators. By comparing the field of trauma triage to software product development, several new and useful insights can be gained into risk assessment and monitoring. In addition, the methods and work done in developing the triage criteria used to classify and assess patients can be translated to software engineering, providing a robust way to establish, utilize, and maintain criteria for software. See the slides for more details. The paper will follow the general order and content of the slides, but in more detail.

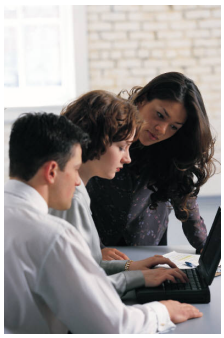
About the Author

Erik Simmons has 15 years experience in multiple aspects of software and quality engineering. Erik currently works as a Platform Quality Engineer within the Corporate Quality Network at Intel Corporation. He leads the corporate Software Engineering Process Team that is charged with improving software development capabilities across Intel's product development groups, and is responsible for Intel's product requirements engineering practices.

Product Triage

A “Medical” Approach to Predicting and Monitoring End-Product Quality

Erik Simmons, Intel Corporation



For information, contact:
Erik.Simmons@Intel.com

Trauma Triage

Triage is the process of prioritizing responses to patients based on the severity of the presenting symptoms.

In only a few seconds, an EMT on scene must decide whether a trauma system activation should be called for an injured patient.

This decision is based on experience and about two dozen Triage Criteria – true/false conditions that are associated with the need for trauma system activation.

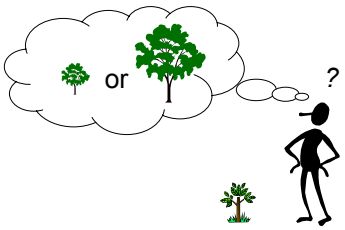
Truths about Prediction

There are two fundamental but conflicting truths about predictions and estimates:

- The value of the prediction increases with distance from the event being predicted

BUT…

- The difficulty of making accurate predictions also increases with distance from the event being predicted



Measures, Metrics, and Indicators

A **measure** establishes the "extent, dimensions, capacity, etc. of anything, especially as determined by a standard"*

A **metric** is a typically a composite of two or more measures

An **indicator** is the result of a comparison of a measure or metric with a baseline quantity or expected result

For Example

Measures: Number of software failures, time

Metric: Mean Time Between Failures (MTBF)

Indicator: MTBF < 200 hours

*Adapted from *The Handbook of Software Quality Assurance, 3rd Ed.,* by Schulmeyer and McManus, PTR 1998

Measures, Metrics, and Indicators

There are two type of indicators: *Sentinel* and *Rate-based*

Sentinel indicators are triggered by any occurrence of the condition

Rate-based indicators are triggered when a metric falls above or below established limits

Example:

Sentinel Indicator: Any patient death occurring outside of the Intensive Care Unit

Rate-based Indicator: Average on-scene time of more than 10 minutes prior to patient transport

Sensitivity and Specificity

The **sensitivity** of an indicator or test is how well it detects the problem of interest – the percentage of all true cases captured.

The **specificity** of an indicator or test is how well it excludes test subjects without the problem of interest – the percentage of all false cases rejected.

A good indicator has high sensitivity *and* high specificity

ROC Curves

Receiver Operating Characteristic (ROC) Curves were created to plot signal-to-noise ratios in electronics, but they have also been widely used in medicine to analyze diagnostic tools.

ROC Curves used in trauma care usually contain Sensitivity plotted against Specificity Loss (1-Specificity) for various probabilities of a case being positive.

A ROC Curve shows how Specificity Loss varies with Sensitivity.

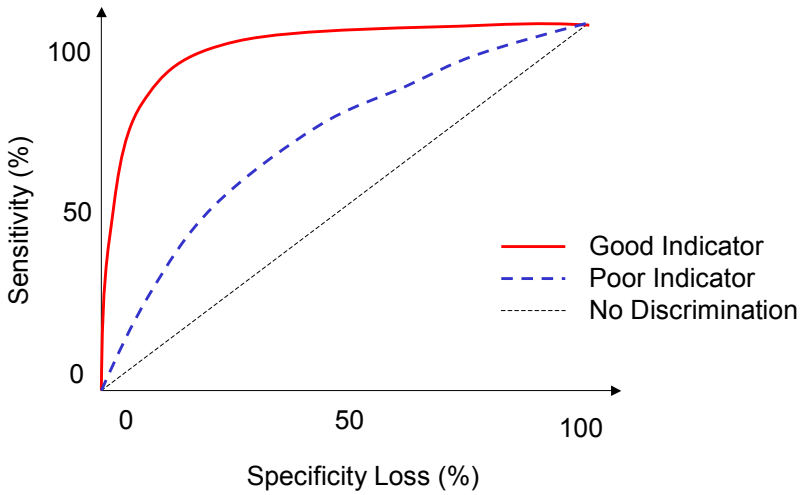
Sample ROC Curve Data

Probability Cutoff for Positive Cases	Resulting Sensitivity	Resulting Specificity Loss
90%	15%	5%
75%	50%	15%
50%	75%	25%
25%	89%	35%
10%	93%	95%

In this example, the “sweet spot” is at 25% probability of occurrence, where sensitivity is 89% and specificity loss is 35%.

Increasing sensitivity to 93% increases specificity loss to 95%.

Sample ROC Curves



Using ROC Curves

A **good indicator** comes close to the upper-left corner of the plot. This means that a large gain in sensitivity is available without much specificity loss.

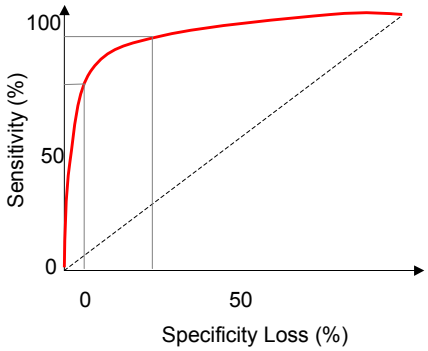
A **poor indicator** lies closer to a 45-degree line in the plot, indicating that sensitivity gain and specificity loss occur at similar rates.

Where to set the probability cutoff depends on things like the cost of false negatives and false positives.

For example, two HIV tests were developed:

- An inexpensive first pass with high sensitivity and poor specificity
- An expensive follow-on with high sensitivity and high specificity

Using ROC Curves



Q: What is the change in specificity loss when moving from a sensitivity of 75% to 90% for this indicator?

A: About 25%.

If we increase the sensitivity, what will the increase in false positive cases (associated with greater specificity loss) cost the system?

Using ROC Curves

Like human diseases, most problems in product quality affect only a small proportion of the base population.

An increase in sensitivity of a certain percent (say 15%) will cause some additional specificity loss (say 25%).

Since the negative population is so much larger than the positive population, an additional 20% false positives is a large number compared to additional true positives, and may burden the system heavily.

True Population	75% Sensitive, 5% Spec. Loss	90% Sensitive (+15%), 30% Spec. Loss (+25%)
N=1000, 100 Positive 900 Negative	75 pos. cases, 45 false pos.	95 pos. cases (+15), 270 false pos. (+225)

15 more positive cases found, but at what price?

Implications for Product Quality

A Few Tough Questions

- Are the indicators used to measure product health known to be associated with customer and end-user quality?
- Which indicators are the best predictors? How do we know?
- Are the indicators evaluated for predictive validity on a regular basis?
- How are new indicators located and deployed?
- Are there combinations of indicators that are more powerful than individual ones?
- Does the experience of a Quality Assurance Engineer, Project Manager, etc. contribute to (or even outweigh) the power of indicators?

Implications for Product Quality

Imagine a world where:

- A model with high sensitivity and low specificity loss is used early and often in the product lifecycle to predict the need for enhanced quality monitoring, quality assurance, or project management.
- The model is not only proven accurate, but is re-evaluated periodically and adjusted for optimum performance.
- This performance adjustment is made based on a ROC Curve or similar data-driven device so that the expected effects of the adjustment are known before the fact.
- The quest for new indicators is ongoing, and indicators are added based on known predictive validity.

Product Triage Criteria

Trauma Triage Criteria are broken into 4 categories:

- **Physiology** – Vital signs & alertness problems
- **Injury Anatomy** – Burns, longbone fractures, paralysis, and other types of dangerous & life-threatening injury
- **Injury Mechanism** – Falls > 20 feet, vehicle rollover, high-energy transfer, etc.
- **Co-Morbid Factors** – Pregnancy, age < 12 or > 60, hostile environmental conditions, etc.

What are the parallel conditions in product development?

Product Triage Criteria

Possible Parallels

- **Product "Physiology"**: Team size, product size, estimated project length, total budget, budget expenditure rate, etc.
- **"Injury" Anatomy**: Schedule slippage, change in scope, significant rework, project restart, missed milestone, loss of sponsorship, excessive turnover, etc.
- **"Injury" Mechanism**: Market changes, unrealistic estimates, inadequate requirements engineering, uncontrolled baselines, & other poor engineering practices
- **Co-Morbid Factors**: Development in multiple locations or cultures, complex products, inexperienced team, inadequate stakeholder access, green business, multi-tasked resources, etc.

Creating a Model

1. Develop a list of candidate Product Triage Criteria.
2. Define precisely what is meant by ‘bad outcome’ – cancelled project, late by x%, overbudget, etc.
3. Understand the costs involved (tests, false positives, and false negatives) in order to make a good model.
4. Measure the ability of the candidate criteria to predict products that are at risk of a ‘bad outcome’
5. Develop and tune a predictive model based on costs, acceptable sensitivity, and acceptable specificity loss.
6. Pilot and deploy the model, adjusting as needed.
7. Maintain, re-evaluate, and tune the model over time with new data.

Product Triage:

A "Medical" Approach to Predicting and Monitoring End-Product Quality

Erik Simmons
Intel Corporation
JF1-46
2111 NE 25th Ave.
Hillsboro, OR 97214-5961
erik.simmons@intel.com

Version 1.0, 03/29/01

Prepared for Quality Week 2001

Key Words: Product Quality; Metrics; Leading Indicators; Risk

Author Biography

Erik Simmons has 15 years experience in multiple aspects of software and quality engineering. Erik currently works as Platform Quality Engineer in the Platform Quality Methods group, part of the Corporate Quality Network at Intel Corporation. He is responsible for Requirements Engineering practices at Intel, and lends support to several other corporate software and product quality initiatives. Erik is a member of the Pacific Northwest Software Quality Conference Board of Directors. He holds a Masters degree in mathematical modeling and a Bachelors degree in applied mathematics from Humboldt State University in California.

Abstract

The medical field of trauma care provides interesting and valuable metaphors for product quality. By adapting some of the tools and concepts used within trauma care to product quality rather than patient outcome, a new and disciplined approach to predicting and monitoring product quality emerges. As in trauma care, the decision to render specialized care and interventions in product triage is governed by the use of triage criteria. These criteria are binary conditions that are known through analysis and studies to be correlated with poor outcomes and the need for enhanced care. Sensitivity and specificity are defined and used to quantify the performance of indicators. Receiver Operating Characteristic (ROC) Curves are introduced as a very good way to quantify the predictive power and cost of a system of indicators. Parallels between trauma triage and product development are presented and explored. A process for developing a set of product triage criteria is given.

Introduction

Motor vehicle crashes are the leading cause of death in the United States among children and young adults 5 to 27 years of age, and crashes ended more than 41,000 lives overall in the US in 1999 [NHTSA99]. More than three million people were injured in collisions during that same period. Aside from motor vehicles, thousands more people die each year from falls, burns, intentional violence, and other sources of injury.

Trauma care began on the battlefield where severe injury is common, and rapid, effective treatment is essential. The practices and techniques have been refined and extended over time, and many states now have trauma systems established to deal with the challenges posed by traumatic injuries.

Differentiated care for victims of serious injury within trauma systems has been demonstrated to lead to improved outcomes and reduced mortality. Patients with minor injuries tend to do well regardless of the level of treatment rendered, but severely injured patients require treatment in specialized centers containing skilled specialists, enhanced equipment, and advanced techniques to reduce the risk of death and long-term disability. Beyond improved outcomes, trauma centers also shorten the recovery time of patients.

As part of systematized trauma care, Emergency Medical Technicians (EMTs) arriving on the scene assess the condition of the victims using triage. Triage is the process of prioritizing treatment to patients based on the severity of the presenting symptoms. EMTs use a set of triage criteria to perform an assessment of observable or easily measurable risk factors known to be associated with the need for treatment within a trauma center. Although this assessment must be done very rapidly, the accuracy of the system is impressive. During a study performed in the state of Oregon in 1995, 87% of the patients that needed trauma system care were correctly entered into the system by EMTs using a combination of the triage criteria and their experience [Simmons95].

New product development is commonly a risky endeavor, judging by the number of high-profile failures that continue to occur – especially in the field of software engineering. Trauma triage provides some interesting insights into how risk for failure might be measured early, quickly, and accurately. Before these insights can be explored, a few concepts need to be defined and discussed.

Measures, Metrics, and Indicators

Though they are sometimes used interchangeably, the terms measure, metric, and indicator are all distinct. Good definitions are provided in sources like [Schulmeyer98]:

- A **measure** establishes the “extent, dimensions, capacity, etc. of anything, especially as determined by a standard”
- A **metric** is a typically a composite of two or more measures
- An **indicator** is the result of a comparison of a measure or metric with a baseline quantity or expected result

For example,

Measures: Number of software failures, time
Metric: Mean time between failures (MTBF)
Indicator: MTBF < 200 hours

Meaningful decisions are made from metrics through indicators. There are two major categories of indicators: sentinel and rate based. Sentinel indicators are triggered by any occurrence of the condition, while rate-based indicators are triggered when a metric falls outside of established limits. For example, trauma hospitals often have indicators like the following:

- **Sentinel indicator**: Any death occurring outside the Intensive Care Unit (every one is reviewed)
- **Rate-based indicator**: Average on-scene time of more than 10 minutes before patient transport (procedures are reviewed when the rate is exceeded in any review period)

Most indicators are not definitive in demonstrating a problem or condition, but instead define levels of metrics that are associated with that problem or condition (that is, they *indicate* either healthy or unhealthy conditions, but *prove* nothing). A "normal" temperature or blood pressure reading indicates a healthy condition, while a high temperature or blood pressure reading is associated with many different conditions, almost all unhealthy.

The level chosen for an indicator is sometimes set based on a goal or opinion rather than any solid clinical evidence – a practice that can lead to serious problems. Indicators like those used in trauma triage are proven associated with conditions requiring the capabilities of a trauma system hospital. Many indicators have been created in the realm of software engineering, but only a few good efforts to evaluate them have been made, most notably the COCOMO II model for creating cost estimates [Boehm01]. The COCOMO II variables encompass risk reduction efforts and a host of factors known to affect cost, and might serve as a point of departure for any efforts to create a system as described in this paper. Other items from which to work include the published risk taxonomies from various sources, and the program management work of the NASA Software Engineering Laboratory (SEL).

Sensitivity and Specificity

The performance of an indicator can be objectively measured using sensitivity and specificity. The **sensitivity** of an indicator or test is how well it detects the problem of interest – the percentage of all true cases captured. The **specificity** of an indicator or test is how well it excludes test subjects without the problem of interest – the percentage of all false cases rejected.

A perfect indicator would have 100% sensitivity and 100% specificity. However, in practice, such indicators rarely if ever exist, and some specificity loss is accepted in order to achieve an acceptable sensitivity. For example, in the previously cited study on trauma triage criteria [Simmons95], the system performed at 87% sensitivity, and 83% specificity. The 17% specificity loss represents patients entered into the trauma system unnecessarily (also called *over triage*). These patients represent a significant cost to the system, as do the 13% of patients that were not entered into the system but needed the care it provides (*under triage*). In this case, the price of under triage can be human life, but tuning a system of indicators to improve sensitivity can have serious side effects on the system, as demonstrated later.

Individual indicators are Boolean; that is, a condition is either present or absent or a rate is either exceeded or not. This means that individual indicators have only a single sensitivity and a single specificity (though there may be some diagnostic errors or similar factors at work). Using a individual indicator, there are only two possible probabilities for a case being positive: zero or one. In systems where many indicators are used together, such as the one used in trauma triage, each indicator contributes to the sensitivity and specificity of the overall measurement. This means that there are several possible probability levels for positive cases, each with a corresponding sensitivity/specificity pair.

This is important because the various possible sensitivity/specificity pairs can be plotted together in order to understand how the indicators work together as a system. This allows the optimal probability for a case being positive, but more than that, it allows the system to be tuned based on an understanding of what effect changing the probability threshold for positive cases will have. This can be represented visually using Receiver Operating Characteristic (ROC) Curves.

ROC Curves

ROC Curves originated in electronics to plot the signal to noise ratio of receivers, but since an indicator or indicator system seeks to separate signal (positive cases) from noise (negative cases), they work well in this arena also [Hanley82]. A ROC Curve contains sensitivity plotted against specificity loss (1-specificity) for all possible probabilities for positive cases within a given indicator system. Simplified example ROC Curve data are shown in Table 1.

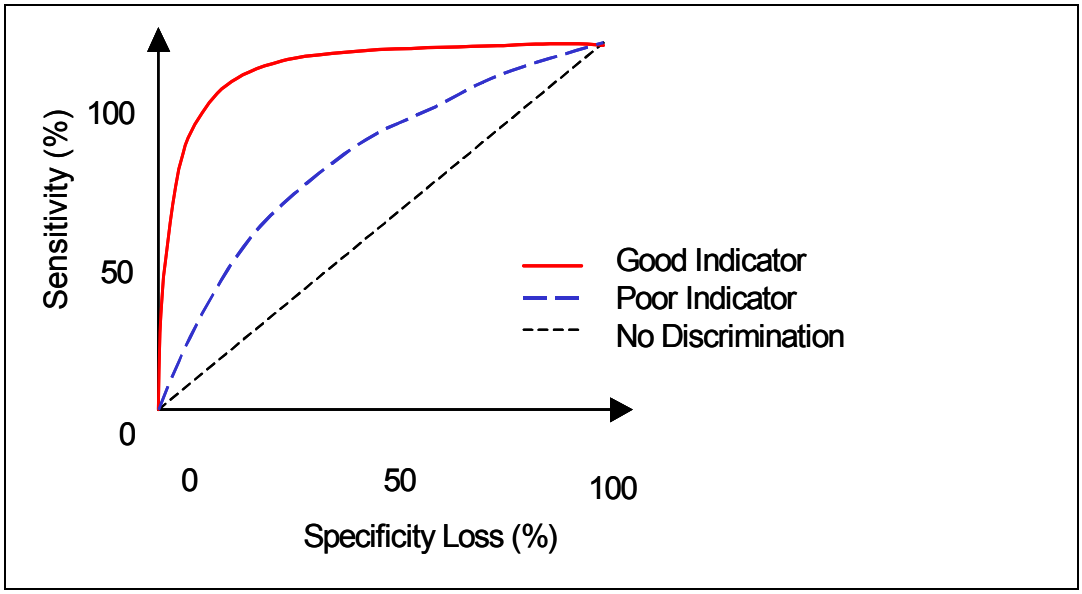
Table 1

Probability Cutoff for Positive Cases	Resulting Sensitivity	Resulting Specificity Loss
90%	15%	5%
75%	50%	15%
50%	75%	25%
25%	89%	35%
10%	93%	95%

In this data, the “sweet spot” is located at 25% probability of occurrence, which yields 89% sensitivity with 35% specificity loss. Note that decreasing the threshold probability to 10% does increase sensitivity to 93%, but at the cost of increasing specificity loss to 95%. This is akin to a system that flags almost every case as positive, locating all but 7% of the positive cases but also taking in all but 5% of the negative cases. In this example, there is a practical limit to what the indicators can do without exorbitant cost to the system.

Figure 1 shows two ROC Curves. In this example, the curve represented by the upper solid line represents a good indicator system like the one in Table 1, and the one represented by the middle dashed line represents a poor indicator system. The diagonal line represents a system with no discrimination power at all – that is, increases in sensitivity are matched with equal loss in

Figure 1



specificity

.

The closer that a ROC Curve gets to a diagonal line, the worse the corresponding indicator system. The more the ROC Curve deviates from the diagonal towards the upper left corner of the plot, the greater the sensitivity gain available for a small specificity loss.

Once the sensitivity/specificity data have been plotted, the question of what threshold probability to use can be answered. The decision is usually made based on several factors, including the cost of diagnostics associated with the indicators, the cost of false negatives, and the cost of false positives. For example, consider a situation where there are two tests used as indicators of the same condition. The first test is inexpensive, highly sensitive, but also has a high false positive rate. The second test is expensive, highly sensitive, and highly specific to the condition. Under these conditions, the first test might be used as an initial screen, followed by the more expensive but more accurate test performed on just the positive population from the first test.

Tuning Indicator Systems and the Cost of Specificity Loss

Like human diseases, most problems with product quality affect only a small percentage of the total population. While sensitivity is very important, specificity loss is often overlooked (at least at first) when designing and tuning indicator systems. Suppose that increasing the sensitivity of an indicator system by 15% from 75% to 90% (through adjusting the threshold probability of a positive case) results in a specificity loss of 25% from 5% to 30%. This sounds OK on the surface, but as Table 2 shows, the effects are startling when the true positive population makes up only 10% of the total:

Table 2

	Before:	**After:**
True Population N=1000	75% Sensitivity 5% Specificity loss	90% Sensitivity (+15%) 30% Specificity loss (+25%)
100 positive 900 negative	75 positive cases found 45 false positives	90 positive cases found (**+15**) 270 false positives (**+225**)

In this example, an additional 15 positive cases were located, but at the cost of 225 new false positives! This would overwhelm a quality organization tasked with deeper diagnosis, monitoring, or intervention. A better test or indicator system created to identify the true at-risk population must be found.

Product Triage

Trauma triage is based on several categories of indicators. Though there are variations between the indicator sets used in different trauma systems, the set used in Oregon is typical and is given in Table 3.

Table 3

Group	**Criterion**	**Description**
Physiology	1*	Systolic blood pressure < 90 mm Hg
	2*	Respiratory rate < 10 or > 29 per minute
	3*	Glasgow Coma Scale < 13
Injury Anatomy	4*	Penetrating wound mid-thigh to head
	5*	Burns to more than 15% of total body surface, or to face, feet, hands, or genitalia (in conjunction with other injuries)
	6*	Amputation proximal wrist or ankle
	7*	Spinal cord injury or paralysis

	8*	Flail chest
	9*	Two or more obvious proximal long-bone fractures
Mechanism of Injury	10*	Death of same car occupant
	11*	Ejection from an enclosed passenger space
	12*	Complex extrication lasting more than 20 minutes
	13	Fall of 20 feet or more
	14	Pedestrian hit at 20 mph or more or thrown at least 15 feet
	15	Vehicle rollover
	16	Motorcycle, ATV, or bicycle crash
	17	Significant impact or intrusion to passenger space
Co-morbid Factors	18	Age < 12 or > 60
	19	Hostile environment (e.g., extreme heat or cold)
	20	Medical illness (e.g., chronic lung diseases, heart failure)
	21	Presence of intoxicants
	22	Pregnancy

* Mandatory system entry for these criteria; for all others system entry is at EMT discretion.

Notice that the triage criteria are broken into four categories: physiology, injury anatomy, injury mechanism, and co-morbid conditions (complicating factors). What parallels can we find to these criteria within product development?

Physiology
These criteria represent the patient's physiology – the 'vital signs'. The vital signs of a product development effort might consist of items like:

- Development team size
- Product size
- Project length
- Total budget
- Budget expenditure rate

Injury Anatomy
These criteria describe the visible signs of the underlying injury. Visible signs of problems for product development include:

- Schedule slips
- Scope changes
- Significant rework
- Project restart
- Missed milestones
- Loss of executive sponsorship
- Excessive turnover

Injury Mechanism
Certain mechanisms are known to be associated with a likelihood of severe injury. Some well-known injury mechanisms for product development efforts are:

- Changing market conditions or stakeholder needs
- Inadequate requirements engineering
- Pressured or otherwise unrealistic estimates (or no estimates at all)
- Uncontrolled baselines
- Other inadequate software engineering processes or practices

Co-morbid Factors
While not injury causes or symptoms themselves, these factors can exacerbate or complicate injuries. For product development efforts, such factors include:

- Development in multiple locations or cultures

- Highly complex products
- Inexperienced team
- Insufficient stakeholder access
- Multitasking resources on more than one project
- Lack of product domain experience

These categories represent new ways of thinking about product risk. In triage, the emphasis is on early, predictive risk assessment. Using the results, product development efforts could be entered into a kind of "trauma system" offering differentiated care as indicated by the project's symptoms.

Imagine a world where:

- A model with excellent sensitivity and low specificity loss is used early and often in the product lifecycle to predict the need for enhanced quality monitoring, quality assurance, project management, or similar activity during development.
- The model is not only proven accurate, but is re-evaluated periodically and adjusted for optimum performance.
- This performance adjustment is made based on a ROC Curve or similar data-driven device so that the expected effects of the adjustment are known before the fact.
- The quest for new and improved indicators is ongoing, and indicators are added based on known predictive validity.

While establishing such a vision will not be simple, the process to create the system is straightforward:

1. Develop a list of candidate Product Triage Criteria.
2. Define precisely what is meant by 'bad outcome' – cancelled project, late by x%, over budget, etc.
3. Understand the costs involved (tests, false positives, and false negatives) in order to make a good model.
4. Measure the ability of the candidate criteria to predict products that are at risk (as defined in step 2) and need enhanced monitoring, management, or other intervention[1].
5. Develop and tune a predictive model based on costs, acceptable sensitivity, and acceptable specificity loss.
6. Pilot and deploy the model, adjusting it with each experience.
7. Maintain, re-evaluate, and tune the model over time with new data.

References

NHTSA99 *Traffic Safety Facts 1999,* National Highway Traffic Safety Administration DOT document HS 809 092

Simmons95 Simmons, Erik, et al, *Paramedic Injury Severity Perception Can Aid Trauma Triage*, Annals of Emergency Medicine 26:4, October 1995

Hanley82 Hanley, J.A., and McNeil, B.J., *The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve*, Radiology 1982; 143:29-36

[1] Logistic regression, classification and regression trees, and other statistical techniques can be used for this step, depending on the nature of the indicator data.

QW2001 Paper 6A1

Mr. Steve Whitchurch
(Mentor Graphics Corp.)

Trials and Tribulations Of Testing a Java/C++ Hybrid Application

Key Points

- What are the issues testing a C++ / Jave Hybrid Application?
- Low tech methods to achieve a high quality product.
- Using Java to test a Java GUI.

Presentation Abstract

My paper is about a project that was to build a viewer (Streamview) that would take a GDSII (Graphical Design Data) Stream File as input, and display it using as little system memory as possible, and as fast as possible. GDSII design files can be Giga Bytes in size, and use a lot of system resources. The underlying code would be written in C++, while the user interface would be written in Java. These two layers would then communicate using the JNI. This combination of C++, Java, and the complexities of an application like this, generated a whole lot of questions on how to test such an application. This paper will talk about some of the issues, and how as a team, we solved them. The paper addresses the following areas:
1. QA & Developemnt Roles.
2. C++/Java Testing Issues.
3. Test Automation Tools.
4. Project Documentation.
5. Problem Reporting within the Project.
6. Alpha/Beta Tetsing.
7. Conclusion/Future Work.

About the Author

Steve Whitchurch has been in the Software QA arena for 17 years. During that time he has worked at Intel, Mentor Graphics, Summet Design, Tektronics, and is currently the lead QA Engineer for a new product line in the Custom IC Devision of Mentor Graphics. Steve has been involved in testing everything from Real Time Operation System Software, Vidio Editing and Special Effect Software, to Electronic Design Automation Software. Steve has also been active outside of the work environment as a Speaker at PNSQC and STAR. Steve was also the creator and publisher of the Software QA Magazine (now known as Software Testing &

Quality Engineering Magazine, published by SQE). Steve as writes for the QQ/Testing Web Site, Sticky Minds.

Trials and Tribulations of Testing a Java/C++ Hybrid Application

By
Steve Whitchurch

Topics

- The StreamView Project
- Future Plans
- Current Status

In The Beginning

- Proof of Concept Project
- One Senior Development Engineer
- One Senior QA Engineer

QA's Role

- Test Planning
- Unit Testing (C++ Code)
- Test Coverage
- Design Feedback
- No Java Yet

StreamView Team

- Three Development Engineers
- Two QA Engineers
  - ➤One Full Time QA Engineer
  - ➤Two Half Time QA Engineers

New QA Tasks

- Traditional QA Roles
- Testing Tasks bubbled up to GUI Level
- Customer Designs as Test Cases

Project Release Roles

- QA Lead All Testing Tasks
- Development Engineers Helped Test
- Customer Support Helped Test

Testing Issues

- Test the C++ Code
- Java Testing
- Cross-Platform Issues
- Testing Tools

Testing C++ Code

- Unit Testing
- Testing Through the JNI
- Inspections
  - ➤Engineering Documentation

JAVA Testing

- JAVA Look and Feel Design Guidelines
  - ➤Sun Microsystems / Addison Wesley
- GUI Bloopers
  - ➤Jeff Johnson / Morgan Kaufmann
- Inspections
  - ➤Engineering Documentation
- Testing Task List

Test Automation

- JavaSTAR
  - ➤Sun Microsystems
- XRunner
  - ➤Mercury Interactive
- QA Partner
  - ➤Segue

Cross-Platform Issues

- Need to Test on Three Platforms
  - ➤Linux
    - – Kernel Versions
  - ➤Solaris
  - ➤HPUX
- GUI Look and Feel

Testing Task List

- Low-Tech way to Automate Manual Testing
- Easy Transition to Automated Test Cases
- Focused Inspection of Functionality

Problem Reporting

- Paper System
- Email
- Low-Tech System
- Works Very Well with Small Teams

Future Plans

- Test Automation
- More Inspections
  - ➤ Engineering Documentation
  - ➤ Code
- Continue to Involve Others
  - ➤ Team Testing
  - ➤ Customer Support
  - ➤ Field Engineering

Two Test Tool Designs

- Intrusive
  - ➤ Hooks in the Application
- Non-Intrusive
  - ➤ Java Reflection

Test Automation

- Java Test Tool
  - ➤Simple Test Language
  - ➤Easy to Use
  - ➤Team Usage
  - ➤Record / Playback
  - ➤Test Management
  - ➤Log File

Test Tool Language

- Comments
- Loops
- Snooze
- Easy to Edit
- Supports Functionality

Current Status

- Prototype Test Tools
  - ➤Intrusive
  - ➤Non-Intrusive
- Stress Testing Current Release
- Development Using Test Tool

Good Info Sources

- Java Look and Feel Design Guidelines
  - ➤Sun Microsystems / Addison Wesley
- GUI Bloopers
  - ➤Jeff Johnson / Morgan Kaufmann
- Sun Java Site
  - ➤www.javasoft.com

Trials and Tribulations of Testing a Java/C++ Hybrid Application

by
Steve Whitchurch
Mentor Graphics, Inc.
Email: steve_whitchurch@mentorg.com
503-685-7945

Abstract

The project was to build a viewer (StreamView) that would take a GDSII (Graphical Design Data) Stream File as input, and display it using as little system memory as possible, and as fast as possible. GDSII design files can be Giga Bytes in size, and use a lot of system resources. The underlying code would be written in C++, while the user interface would be written in Java. These two layers would then communicate using the JNI.

This combination of C++, Java, and the complexities of an application like this, generated a whole lot of questions on how, to test such an application. This paper will talk about some of the issues, and how as a team, we solved them.

About the Author

Steve Whitchurch has been in the Software QA arena for 17 years. During that time he has worked at Intel, Mentor Graphics, Summit Design, Tektronics, and is currently the lead QA Engineer for a new product line in the Custom IC Division of Mentor Graphics. Steve has been involved in testing everything from Real Time Operating System Software, Video Editing and Special Effects Software, to Electronic Design Automation Software. Steve has also been active outside of the work environment as a Speaker at PNSQC and STAR. Steve was the creator and publisher of the Software QA Magazine (now known as Software Testing & Quality Engineering Magazine, published by SQE). Steve was also involved in starting the Software Association of OregonÕs QA Special Interest Group.

QA & Development Roles

As with any new product, StreamView started life as a proof of concept project. The team consisted of two engineers. A very senior Development Engineer, and a senior QA Engineer. For most proof of concept projects, itÕs very unusual for management to assign a QA Engineer to a project while itÕs in the prototype phase.

While in the proof of concept/prototype phase, QAÕs role was to perform mostly unit testing. These unit tests were written in C++, as the Java layer was not yet part of the project.

As the project moved from a proof of concept project to a real product project, more people were added to the team. The additions included a GUI Development Engineer, a Middleware Development Engineer, and two part time QA Engineers.

As the project made this transition, most of the QA tasks bubbled up to the GUI layer, as well they should have. With the Development Engineers now focusing on more of the unit testing, QAÕs tasks needed to focus not only on the wellness of the application, but on the way a user would use the application.

In the case of StreamView, the user test cases were a variety of differing GDSII Stream Files (Customer Designs). These Stream Files would prove to be very valuable during the testing phases of the project., Even with the transition of QA to a more traditional role, there was still some unit testing being performed by QA. This testing was now assigned to a new collage grad that, long term, would move from QA to Development. This was a very good fit for Unit Testing, and the new Engineer.

At the start of the project, it made sense to have a QA Engineer take on the role of Unit Tester. It gave QA the opportunity to learn the application, and to have input into the design process. It also made sense to move the Unit Testing responsibility to a new Development candidate later in the project. This was a win-win opportunity for the team and for the new Development Engineer.

As the application moved closer to a release date, some of the Development Engineers performed testing tasks. Roles were once again changed to fit the needs of the project. Everyone on the project wore different hats at different times. Customer Support was also asked to be involved in the testing cycles. Again, this was a win-win opportunity for the core team and for Customer Support.

When assigning roles, donÕt just assign roles based on preconceptions such as "QA Engineers only test applications at a high level", "Development Engineers donÕt test", etc. Each member of the team can have a positive impact at all levels of the project. In fact, a sign of a well functioning team, is where all team members participate at all levels of the project. We all have something to bring to the table. We should not be pigeon - holed just because of our title.

C++/Java Testing Issues

The testing issues surrounding this seemingly simple application (simple from a functionality point of view) were huge. The following is a small sample of questions that were asked by QA:

- Are there any tools on the market to test a JAVA GUI?
- Can we get at all the underlying C++ functionality from the GUI?
- What about testing the C++ code standalone?
- What about testing the JNI layer?
- What about testing the IPC layer?
- If we are using a Beta version of Java, will this have any impact on the testing tools we use?
- What Java Standard should we follow?
- How do we verify the graphics on the JPannel?

- What about tool tips? How do we validate them?
- What about on-line documentation? How do we test it?
- What about cross -platform dependencies?
- Will the Java GUI really look/act the same on a PC as it does on a Unix box?

A lot of these questions are common questions that should be asked of any project/product that needs to be tested. So from that point of view, this project was not all that unusual. But whenever you add more than one programming language, or more than one supported platform, or more than one whatever, the level of testing complexity increases. It's just a fact of life. Let's take a look at a couple of these issues close up.

What about testing the C++ code standalone?
We determined early in the project that we would realize a big benefit by testing the application from the C++ side. This would help us flush out problems like memory usage, database issues, function call problems, etc. This assumption proved to be correct. There were a lot of problems found just by writing test cases in C++ that would exercise the C++ application code standalone. You may say that this is just Unit testing, but that is not true. In many cases our C++ tests would call the C++ code just like the Java user interface would. An example; we had one test that checked the drawing functions that were written in C++. This test used the JNI to make these drawing calls, and then created a JPanel to display the graphics. This test was very useful in finding C++ functions that had problems or that were missing functionality. A simple unit test would just exercise one function at a very low level. What we had here was a kind of test harness for the C++ layer of the application. You could also call this level of testing API testing.

What Java Standard should we follow?
We used the Sun Java Look and Feel standard. As long as we followed this standard, we could, for the most part, be assured that the GUI would look and run the same on any supported Java platforms. Following this standard was very useful when questions came up about a look and feel of the GUI. I would recommend this standard to anyone building an application GUI in Java. The book is Java Look And Feel Design Guidelines, Addison-Wesley, ISBN 0-201-61585-1.

How do we verify the graphics on the JPannel?
This was a big issue for us because our application is very graphics intensive. ThereÕs always the old bit map method. But bit maps have all kinds of problems with platform environmental issues. We did not choose this method.

We decided on two methods of verifying the drawing on the JPannel canvas. The first was an automated way of verifying what we thought should be on the drawing canvas. Since there is a one-to-one correlation between what is in the graphics database and what is drawn on the canvas, we were able to check the contents of the database to verify the contents of the drawing. We did this by writing out the GDS data and doing a compare with the original design file. If the compare was good, then the translated data in the database was good. And most likely the graphics were good. If the compare failed, then

we took a closer look at the graphics on the drawing canvas. Most of the time we found a drawing problem by using this method.

However, this was only part of the answer. As our second method, we also needed to visually check the drawing canvas. There could always be a translation problem between what was in the data base and what was drawn on the canvas. Both these methods proved to work quite well.

Since GDSII Design data can have millions of shapes in one file, visual checking methods could be a nightmare. To help automate this testing, we created small GDSII design files that focused on one type of shape. For example paths, we created a couple of small design files that had every type of path possible, based on the GDSII Standard.

With any application that does some type of drawing, there is going to be some amount of manual inspection of the data. I donÕt think there is any way around it.

What about testing the IPC layer?
StreamView has the ability to interface to a IC design debugging tool called Calibre-RVE. The way these two applications talk is through an IPC socket. Calibre-RVE reads in a list of design errors, the user selects one of these errors, and then Calibre-RVE sends a message to StreamView to display and highlight the error on the GDSII design.

We tested this mechanism two ways. The first was by just using Calibre-RVE to send commands to StreamView.

The second was an internal tool that would just send Calibre-RVE commands to StreamView. This tool proved to be very valuable in debugging problems found. Any time you're testing communications between two applications, it's very helpful to have a test fixture that can simulate the communications between the applications.

What about cross platform dependencies?
Even though Java is supposed to be platform independent, we did find a couple of platform dependencies. For the most part the Java GUI worked out well. The biggest problem we had was with system fonts. Java has what is called "font.properties files" that define the fonts used for the platform the application is running on. This does not always work.

The other problem that we saw was with the different windowing environments you can have on one platform. For Example, OpenwWindows and CDE in the SUN environment. Sometimes Java would act different on OpenWindows than on CDE.

For the most part I was very happy with the way Java worked. If you follow the guidelines in the Look-And-Feel book, most of the time the Java application will perform the same on all supported platforms.

A good resource for the known Java bug and general information on Java can be found at www.javasoft.com. You can find lots of useful information on this web site that can help you test or develop a Java application.

What about tool tips?
All Tool Tips and on line documentation were tested manually by inspection. The Tool Tips were included as part of the Testing. This proved to be a good way of verifying the Tool Tips and on-line documentation.

Test Automation Tools
One of the problems facing the QA team was test automation. The GUI was based on Java, the underlying code was C++, and the two talk via the JNI. How do you automate this mess? The first step was to find a commercial tool that would fill our needs.

As StreamViews GUI became more robust, we decided to test more of the application from the GUI, the Java side. There would still be some Unit Testing performed, and those tests would be written mostly in C++.

We looked at three tools, JavaStar by Sun Microsystems, QA Partner by Segue, and XRunner by Mercury Interactive. We needed a tool that would work with the latest version of Java, would run on two platforms (to include a 3rd platform in the future), and would be able to effectively test a graphical application. The only tool that fit these requirements was JavaStar by Sun Microsystems. This tool was written in Java, so it would run on any Java supported platform, and it would support the latest version of Java. It did everything we wanted, with one problem. Part way through the project, I received an email from JavaStar Customer Support that said, Sun was dropping the development and support for JavaStar. Not a good day. This basically left us with no testing tool to automate the testing of our new application. The other two tools (QA Partner / XRunner) either did not support Java, or supported an older version of Java, or did not support the platform we were testing on. ThatÕs one of the problems you have when your application is using cutting edge technology.

So now what? In comes the Testing Task List, a pure paper way to automate your testing, and what I consider the most valuable tool any QA/Test Engineer can use. I know most everyone thinks of test automation as a push-button set of tests that run on their own, but that is not the only reason we automate. One of the biggest reasons is repeatable tests. The Testing Task List will give you repeatable test cases. ItÕs also a very good source to drive those push-button test cases when youÕre ready to automate.

Functionality	**Test Information**	**Completed**
View Panel	View Panel Functions	
Pan Functions		
View-All		

Icon (Left Side Palette)	View-All Click Icon	
Tool Tip	Check Tool Tip	
F1 (Help)	Function Help	
Help Key	Help Key Help	
Key Board	View-All from Key Board	
Ctrl-F	View-All using Ctrl-F	
Pan-Up		
Icon (Left Side Palette)	Pan-Up Click Icon	
Tool Tip	Check Tool Tip	
F1 (Help)	Function Help	
Help Key	Help Key Help	
Key Board	Pan-Up from Key Board	
Up Arrow (Arrow Key Pad)	Pan-Up using Up-Arrow key on arrow key pad.	
Page Up	Pan-Up using Page-Up Key	

As you can see from this example, the Testing Task List is very detailed, and simple to execute. This tool can be given to any team member and you will get the very same level of testing from everyone that uses it. It's a very effective and low-tech way to automate testing tasks.

This is what we ended up using for our test automation for the first release of StreamView. We found a lot of bugs using this method. It also provided us with repeatable test cases that could be handed to anyone on the team to perform.

If you are not using something like the Testing Task List, to drive your push-button automated tests, or to drive your manual testing, you are missing the boat as far as having good, comprehensive test cases.

Project Documentation
All our project documentation was written in html. This allowed us to have a web based version of all our project documents on line for anyone to read/review anytime.

This documentation consisted of the following:
1. Project Plan
2. Development Task List. This worked very well. At any time you could see the status of the project. Again a very simple, low tech way to communicate the project status. This task list also had the QA tasks listed.
3. Project Test Plan

4. Testing Task List
5. Problem List

All documents, with the exception of the Project Plan, were living documents. They changed as the project evolved.

One of the things I did as Lead QA was add text links from the Development Task List to the Testing Task List. This way all the QA tasks had examples of what was being tested and how. Anyone reading the test Development Task List could click the QA task link, taking them to the testing Task List for that functionality being tested.

Problem Reporting within the Project
Mentor Graphics has a commercial Problem Tracking System. We chose to not use it early in the project. As a small team it made more sense to use an Email/Paper system instead.

Our low-tech system consisted of email and a html document that was updated weekly, or sometimes daily, depending on how frequently updates were needed. This system worked very well for the team. Again, a very simple, low-tech solution to a problem.

The system worked like this: QA or Development would find a problem. That reporting engineer would send out an email with info about the bug to a team mail group. The engineer responsible for the code would then respond to the email. Once a week the Lead QA Engineer would update the html document with that week's bugs. The responsible engineer would then respond to the bug as fixed by changing a status field in the document.

When the project hit itÕs first major milestone, Code Freeze, the team did switch over to MentorÕs in house Problem Tracking System. By doing this we could officially track bugs. This was important for project management to be effective.

Alpha/Beta Testing
The build person for the project was the lead QA Engineer. This was a carry over from the early part of the project. This worked very well until we got closer to the release date, then this duty was transferred to the group's build engineer. Just like the bug reporting system, there comes a time when a project must conform to the companyÕs standards and/or processes. The build person would build and distribute the Alpha/Beta builds.

The request for new Alpha/Beta releases always came from Marketing. The Marketing group was the interface between Engineering and the Alpha/Beta customers.

Conclusion/Future Work
Something that can be learned from this project is, that you donÕt need an arsenal of expensive tools to produce a high quality product. We used things like the Testing Task List (a very valuable tool), a paper/email form of a problem tracking system, and good

communications among team members, all low-tech methods of software development that proved to be very effective.

When putting together a project team, look for people that can work well together. This is probably the most important and most forgotten aspect of project management. A well oiled team with a good spec and the right skill levels can produce a high quality product. The StreamView Team worked very well together.

If you are testing or developing an application in Java, there are all kinds of on-line and book form resources available. There is also a local group called the Portland Java Users Group www.solidware.com/pjug. And donÕt forget the Sun Java site www.javasoft.com.

One of the future issues that IÕm looking at, is test automation. There comes a time that you just canÕt keep up with the testing task without test automation. One of the problems that we have, and will face constantly as we are developing new applications using cutting edge technology, is that commercial test tools will not be able to keep up. So whatÕs the answer?

IÕm in the process of looking at two methods of writing a test driver using the Java language. By using Java to drive the testing of a Java GUI, I have all the power of the Java language to help with my testing effort.

Intrusive Test Tool
The first test-driver uses a custom script language that enables me to write repeatable test cases. An example of this test scripting language:

```
// Sample Test Case
logfile testlogfile
load design.gds
push viewall
push panup -t
close
exit
```

This test script creates a test log file named “testlogfile, it then loads a GDSII design called design.gds, it then clicks the View All icon on the StreamView View Panel, it then get the Tool Tip for the Pan Up icon on the StreamView View Panel, it then closes the design, and then exits the application. While this test is running, all the test results are written to the test log file.

The Java code to do this is very simple. If you take advantage of the Java language all you need is a hook in the application under test that gives you a handle to (in this case a JButton) to ViewAll and PanUp. The test driver code would look something like this:

To push the ViewAll Icon:

getViewAll().doClick();

do.Click() is a JButton method that presses the JButton on the GUI. The result of the button press is the call to the action listener associated with the JButton.

To get the Tool Tip for the Pan Up Icon:

logPrint.println(getPanUp().getToolTipText());

getToolTipText() returns the Tool Tip text for the associated JButton. In this example, the result is sent to the test log file.

One other feature that I have added to my test-driver language is the concept of looping. This allows me to write test scripts that can repeat a set of test commands n times. An example of this looping feature is:

```
// Sample Test Case
logfile testlogfile
load design.gds
// Loop 4 times
loop 4
{
push viewall
push panup -t
}
close
exit
```

This will loop through the push viewall and push panup -t 4 times then close and exit the StreamView.

I have also added C like comments to my test script language.

Non-Intrusive Test Tool
The second test-driver uses Java reflection to connect to the application under test. This method requires no hooks into the application.

At this point I have this test tool working with one of the Java demo/example applications that comes with the JDK (SimpleExample.java). The tool currently allows me to examine and execute the application, as well as press buttons. This is all done without hooks into the application.
The way the tool works, is by using Java Reflection to gain access to the main constructor.
Once I have a handle to the main constructor, I can access methods, fields, contructors, interface information, etc.

The information about the application can then be used to manipulate Java components. This is a lot like what is happening in the first tool that I described. The two tools are very similar in how they work once I have a handle to the objects that I want to test.

If this second non-intrusive method of testing Java continues to work out. I will add the same test script language to help automate testing.

This is only a sample of what is possible using Java to test Java. My plans are to continue to explore the possibilities. The prototypes of my test-drivers has shown a lot of promise.

I think with one of these new test-drivers, and a good test coverage tool. we are well on our way to a good test automation solution for the StreamView project.


QW2001 Paper 6A2

Mr. Juris Borzovs & Mr. Martins Gills
(Riga Information Technology Inst.)

Software Testing in Latvia: Lessons Learned

Key Points

- Academic research programmes on software testing have led Latvia's IT sector to large scale awareness of testing and quality assurance resulting in adeqate standard development and systematic QA practices at largest IT companies.
- Special role in testing plays the independence of software test team - both inside the project project and as an external assesor.
- Experience of training for testers is shown, pointing to the requirements to select a good team for testing.

Presentation Abstract

Software testing is an integral part of software development process and one of the most efficient quality assurance methods. This presentation reflects the main issues of testing in Latvia’s IT industry: a brief look at the history, an analysis of current day problems and lessons learned, and the vision for the next decade. The experience has been gained from three largest local software companies - DATI, SWH Technology and IT Alise.In Latvia, there is a strong scientific background in the field of software testing. Early research was related to automatic construction of test cases - both theoretical and practical approach. Courses on software testing are included into undergraduate computer curricula of Latvian Universities for more than a decade. Recent and current research in testing is related to software test tools, universal symbolic interpretation, software process improvement (testing issues), practical manual methods of software testing [1-6].A software development as an industry began to develop approximately a decade ago when in 1991 Latvian software designers won a bid for tenders to set up an information system for the social insurance of artists in the German state of Bremen. The work was done successfully, and this early achievement confirmed the fact that Latvian specialists are entirely competitive in Western markets. From year to year, the volume of information technology service exports to the West has grown, and in 1998 Latvia’s two largest software producers - DATI and SWH Technology - exported products worth a total of Ls 5.3 million, and in 1999 - Ls 7 million.

Initial approach was more ad hoc based, and the importance of testing was underestimated. Currently the picture has considerably changed - early starters are now the largest and the most experienced companies with ISO 9001 certified quality systems. The aim to raise the quality criteria was motivated by competition

in IT market and by lessons from projects with unstable success. A considerable evolution of quality requirements both from the suppliers and acquirers is observable. The way towards unified approach in the software engineering was supported by individuals who worked on adoption of IEEE standards for use in a company DATI. As a result, a number of company IT standards were developed, covering fields of software quality assurance, testing, specification, planning and documentation. In parallel a large portion of IT terminology was developed. Most of current Latvian National IT standards were approved in 1996, adopting DATI internal standards.For couple of years in Latvia, major companies large business software development, especially documentation, base on IEEE J-STD-016-1995 standard. Along with that, during product development, largest companies follow a well defined life cycle model with defined processes (adoption of ISO/IEC 12207).

While over the past years systematically working on testing issues, these companies have learned a number of lessons. They are:
1) Testing in the company could be organized both by means of a separate testing institution such as laboratory or department, or by creating a small test team within each project.
2) Testing should not be regarded as just another project activity. Testers should have a background equal to system analyst or designer, plus a specific knowledge of a software testing. As a result a system with a qualification levels for testing professionals has been developed. Within a SWEPE (Software Engineering Professional Education) project, a knowledge area for testing professionals is undergoing definition phase [7].
3) Many software acquirers are poorly informed about testing. To avoid the project failures one should educate a customer about the testing and quality assurance.
4) Well defined test process is the only way to achieve quality with minimal time and staff resources.
5) Although test automation has considerably increased over the past couple of years there is no evidence that it can replace the manual testing activities. Tools for testing can give a strong support, but the test planning and design are almost purely creative activities that do not undergo the automation. Tasks like GUI, user manual testing, help or localization testing almost purely rely on manual testing effort. Just like there is no software that writes the code (semantically), there is not one for test creation.
6) A number of measurements should be made to estimate the efficiency of testing and to check the hypothesis. For example, there has no evidence been found of Pareto principle (where 20 percent source code contains 80 percent of problems [8]) in porting projects.
7) Along with the overall awareness of software testing, the unsatisfied demand for testers is becoming higher than for any other IT profession;

According to various studies, there is a shortfall of between 130,000 and 500,000 software specialists in the world right now, and the deficit may increase to some 1 million people over the course of the next 5-10 years [9-11]. Latvia’s vision for the next decade is closely related to rapid development of IT industry. Rough estimation shows extreme potential of the three Baltic states to become a major

software development and maintenance region able to employ up to 120 thousands IT engineers and to export software and maintenance services. Latvian IT companies, as a result of exporting their services, will reach turnover of Ls 4 billion each year, while the three Baltic States in total can reach a level of Ls 12 billion. Analyzing the software engineering activities that can be performed remotely from the acquirer, the main ones are: design, coding, testing, maintenance, reengineering and porting. Taking into account the overall increase of the awareness of the role of quality assurance and independent testing, the software testing is the IT industry sector to experience a rapid development.

About the Author

Mr. Juris Borzovs was born in 1950 in Finland. He graduated from the University of Latvia (UL), Riga, Latvia in 1973, received his candidate of science degree from the Institute of Mathematics, Belorussian Academy of Sciences, Minsk, Belarus in 1989, Dr. and Dr. habil. degrees from the UL in 1992 and 1999, both in computer science. He was with the UL in 1973-1992. Now he is a Director with Riga Information Technology Institute (RITI) and half-time Docent with the UL and with Riga Technical University. Since 1992 he was the Head of Software Testing Lab within RITI dealing with testing for industrial software projects. Mr. Borzovs is a Member, Strategic Group, Coordinating Council, National Programme "Informatics"; Chairman, IT Terminology Subcommittee, Terminology Committee, Latvian Academy of Science; Vice Chairman, IT Standardization Technical Committee; Member, Legislation Drafting Task Forces, Ministries of Transport, Culture, Welfare; Invited Expert, Government Task Force on Millenium Problem; Member, Swedish-Latvian IT Council, The Baltic Sea IT Fund, Swedfund; President, Latvian Information Technology and Telecommunication Association (LITTA); Founder and First President, Latvia Chapter, Information Systems Audit and Control Association (ISACA); Vice President, Latvia Chapter, International Federation for Automated Control (IFAC); Member, IEEE Computer Society and IEEE Standards Association. He has published over 60 scientific papers. His interests include software quality, software testing, software engineering, software copyright, and terminology.Participated with presentation in ICSTEST conference in Bonn, April 5-7, 2000.

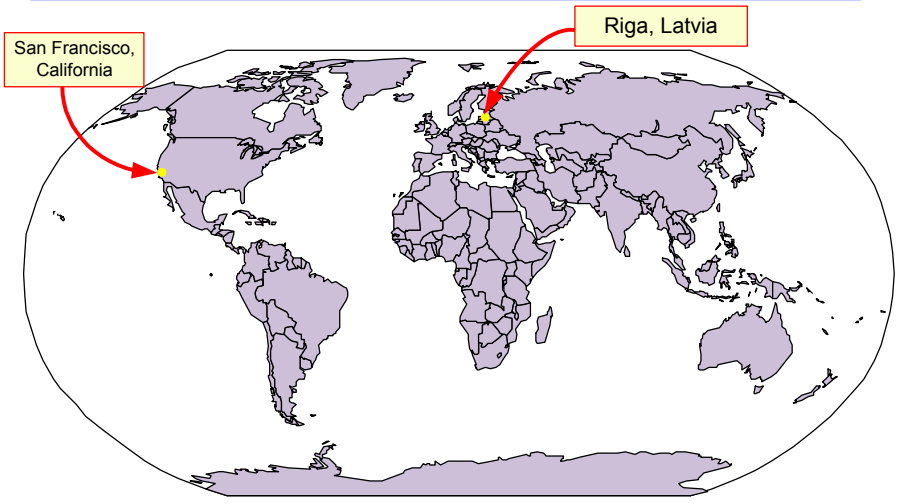
Mr. Martins Gills received his MSc cum laude in Computer Science in 1999 from University of Latvia. For several years he has been working in the testing field. Currently he is a test team leader in Riga Information Technology Institute (RITI). One of the most challenging projects he managed was CANON printers software driver GUI testing comprised 18 natural languages and performed distantly. During last two years, he mostly participated in year 2000 testing projects. His main testing interests are related to test method efficiency. Participated with presentation in SQM and ICSTEST conferences in Bonn, April 5-7, 2000.

Software Testing in Latvia: Lessons Learned

Juris Borzovs, Martins Gills
Riga Information Technology Institute

QW 2001 conference. San Francisco, 29 May - 1 June 2001

Where are we?



Testing in Latvia: major publications (1)

- *J.M.Barzdins, J.J.Bicevskis, A.A.Kalnins*. **Construction of Complete Sample Systems for Correctness Testing**._ In: Mathematical Foundations of Computer Science, Berlin: Springer, 1975, pp. 1-12.
- *J.M.Barzdin, J.J.Bicevskis, A.A.Kalninsh*. **Automatic Construction of Complete Sample Systems for Program Testing**._ In: Proc. IFIP Congress, 1977, North-Holland, 1977, pp. 57-62.
- *A.Auzins, J.Barzdins, J.Bicevskis, K.Cerans, A.Kalnins*. **Automatic Construction of the Test Sets: Theoretical Approach**._ In: J.Barzdins, D.Bjorner (eds.) Lecture Notes in Computer Science: Baltic Computer Science, No. 502, Springer-Verlag, 1991._ pp.286-359.
- *J.Bicevskis, J.Borzovs, U.Straujums, A.Zarins, E.F.Miller,jr*. **SMOTL - A System to Construct Samples for Data Processing Program Debugging**._ IEEE Transactions on Software Engineering, vol. SE-5, No.1, 1979, pp. 60-66.

Testing in Latvia: major publications (2)

- *J.Borzovs, A.Kalnins, I.Medvedis*. **Automatic Construction of Test Sets: Practical Approach** ._ In: J.Barzdins, D.Bjorner (eds.) Lecture Notes in Computer Science: Baltic Computer Science, No. 502, Springer-Verlag, 1991._ pp.360-432.
- *Z.Bicevska, J.Bicevskis, J.Borzovs*. **Regression Testing of Software System Specifications and Computer Programs**._ Proceedings of the 8th Software Quality Week, San Francisco, 1995, paper 5-T-1 (9 p)
- *J.Borzovs, M.Gills, A.Adamsone, S.Linde, J.Plume*. **Software Testing in Latvia: Lessons Learned**. 1st International Conference on Software Testing ICSTEST - Conference Proceedings. Bonn, April 5-7, 2000. Track A, Paper 2. p 15.

IT in Latvia: a snapshot (1)

1991 - First international development project by Latvian software designers to set up an information system in the German state of Bremen. The successful result gave the basis for the local software industry.

1991 - First professional test group in SW company.

1993 - First independent testing project outside SW company. Customer - telecommunication company Lattelekom.

1996 - First independent testing project outside Latvia. Customer - Canon Systems Management Europe (UK).

IT in Latvia: a snapshot (2)

1999 - First ISO 9001 certification of Software Testing Laboratory in country. Widespread Y2K testing.

2000 - Latvia's two largest software producers - DATI and SWH Technology - exported development services worth a total of over 15 million USD.

2001 - Up to 100 local companies are engaged in SW development. Three major of them, DATI, SWH Technology and IT Alise, employ more than 650 IT professionals.

- 5 computer manufacturers and 7 software companies in Latvia are ISO 9001 certified.

Local IT standards: the origin

1993-1995 Leading IT company group launched the initiative to examine internationally used software engineering standards.

Goals:

- Search for internationally recognized standards
- Assessment of the experience from IT projects
- Development of guidelines, standards and templates

Output:

- Annotated List of industry standards
- IT terminology in Latvian
- Company standards, guidelines; mostly based on IEEE
- Changes in SW development process management

National IT standards

Most of Latvian National **IT standards** were approved in 1996, adopting company internal standards based on IEEE standard family.

Areas covered:

- Software QA, configuration management, reviews and audits
- Testing, verification and validation
- Software documentation
- SW requirements specification, Operational concept description
- Guidelines for Software design descriptions
- Project management plans
- User documentation

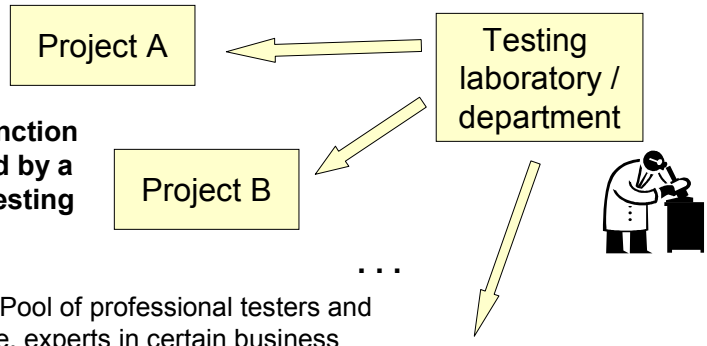
IT standards: industry practice

For several years in Latvia, large business software development, especially documentation part, is based on ***IEEE J-STD-016-1995*** *Software development: Acquirer-supplier agreement standard*.

Main features:

- All software development project activities specified
- Major document templates given
- Includes guidance for tailoring to a certain project
- Built-in quality assurance mechanism

Who performs testing? (1)



Testing function is provided by a separate testing laboratory

Advantages: Pool of professional testers and QA people, experts in certain business areas, methodological support.

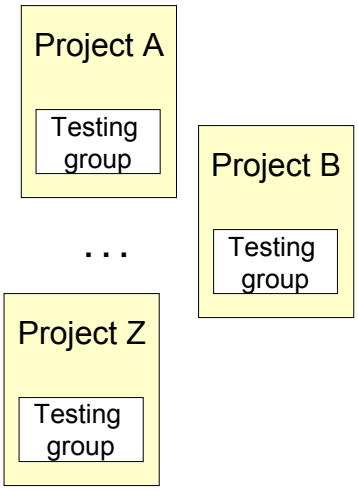
Disadvantages: Testing mostly impossible at unit or integration level.

Who performs testing? (2)

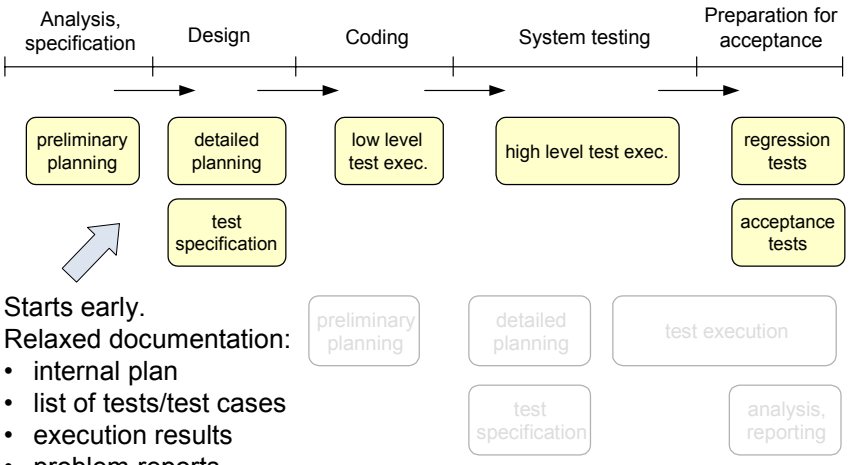
Each project has a small testing group

Advantages: Testing at all levels, high knowledge about software under test.

Disadvantages: Reduced independence, may not be testing professionals but rather programmers, designers or analysts.



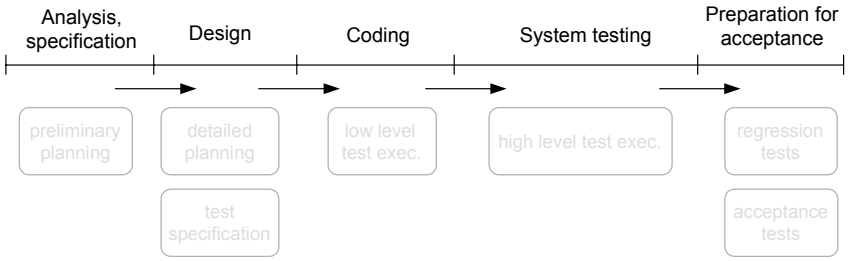
Testing inside a project



Starts early.
Relaxed documentation:

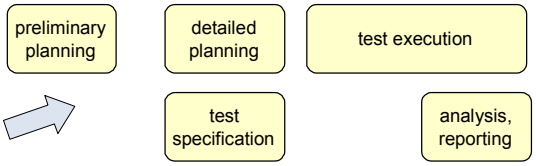
- internal plan
- list of tests/test cases
- execution results
- problem reports

Independent testing



Relatively late activity.
Defined documentation:

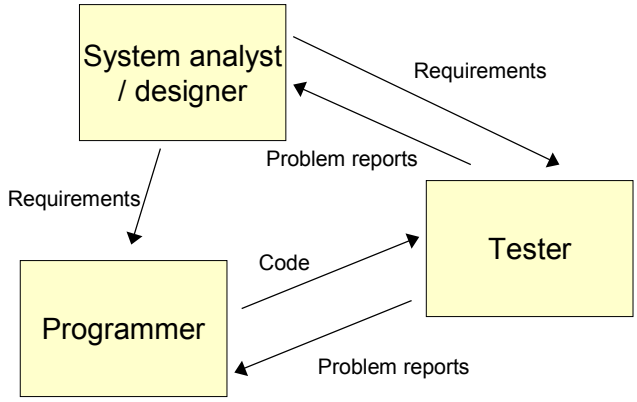
- Plan
- Test description
- Execution results, log
- Problem reports
- Test Report



Test levels and performers

Test level	Structural tests	Functional tests	Performed by
Unit testing	+		developers
Integration testing	+	+	developers
System testing		+	developers, independent testers
Qualification testing		+	developers, independent testers
Acceptance testing		+	customer, independent testers

Collaboration among the developers



What developers do expect?

- Fast integration into project scope
- Recommendation for QA and testing, adopted to local needs
- Problem/bug reports with competent evaluation of the cause and consequences
- Introduction of test automation

Not recommended:

Too high abstraction from technological issues of programming, focusing only on pure functionality checking.

Tasks for the project test team

- Definition of test process
- Ensuring the developer testing and QA activities
- Independent evaluation of the product
- Preparation for user acceptance

Body of knowledge for individuals

- Basic concepts and definitions of testing
- Analysis of test levels
- Test techniques
- Test automation
- Test related measures

Background of people in the team

- Long IT and SE experience
- University graduates, fresh programmers
- Temporary worked in non IT sphere
- Non software development experience

There is no distinct type of person that has come to testing.

But there is an evidence that person has to practice testing work for at least one year prior to definitively feel his/her true capabilities in testing. This time period is critical

Tasks of tester

SW development activity	Tasks for tester
System analysis, specification of requirements	Verify whether requirement specification and other documents are suitable for testing. Plan the testing: its tasks and schedules.
Design	Plan in detail the tests: type, coverage. Make test descriptions.
Coding	Refine plans and test descriptions. Execute tests. Make automation and regression tests.
Testing	Thoroughly implement the system, installation and documentation testing. Do regression testing.
Prior to delivery (preparation for the acceptance)	Check-up the corrections. Run the regression tests.

Qualification levels of the tester

Level	Tasks	Required skills
1	Execution of previously specified tests and other tasks that do not require special knowledge or training	Computer usage skills - advanced used level and ability to understand specified tests.
2	*Tasks of 1st level +* Test description according to system documentation.	*Skills of 1st level +* Ability to analyze system documentation and to write a testing documentation.
3	*Tasks of 2nd level +* Test planning and managing of test team	*Skills of 2nd level +* Ability to identify required tests, skills in team and resource management.
4	*Tasks of 3rd level +* Training of other testers and development of testing methods	*Skills of 3rd level +* Tutor skills and expert knowledge in testing theory

Training concerns

When schedules are pressing, the priorities can be set of what first and what later can be taught.

Groups:

- New testers
- Existing testers
- Programmers expand their qualification

Components:

Lectures, workshops, sample problems, guideline documents, software for training, final test.

Software for training

Aim - to simulate real testing process, system testing level

Sample - specially adopted cargo delivery accounting system:

- Small fully functional application with built-in defects
- Development and user documentation available
- Source available, if necessary

Tasks for trainees:

- to plan tests and to specify good test cases,
- to write documentation,
- to work with problem database

Who fits best for testing?

Testing professional is a software engineer specialized in testing.

- Anybody can be trained fast for basic activities
- Good testers are those who know well at least one non-IT area (finance, technical, social, etc.)
- Testing itself is an art, but only sometimes testers should be artists

Tester's main task is to have a different look at development activities.

Bad experience

Personnel issues:

- Bad programmer as tester
- Testing as temporary punishment
- "Very independent" testing - too distant from SE issues
- Self-narrowing of the tasks to pure execution

Plans *versus* reality:

- Developers - expect too much
- Testers - promise too much

Career perspectives

Tester forever? - how long can person be involved in testing.

Options:

- Reaching different professional levels
- Becoming a QA professional
- Switching to programming or system analysis

The evolution of the SE itself makes the exact career perspectives open. One has to diversify the tasks for each individual.

Testers are not grave and problem-fault-focused people. They can and they are creative personalities!

Customer: QA awareness problems

Problem: customers refuse to pay for testing activities (assuming that product will be perfect).

Commonly asked customer questions are: *What is testing? Why is it necessary for the particular project? Why does it cost so much?, etc.*

Incomplete knowledge about quality related activities within the software development life-cycle →

- identification,
- documentation,
- tractability,
- verification, validation,
- reviews, audits,
- testing,
- problem resolution
- ...

Educating the customer

How does the developer benefit from a well informed and competent customer?

Main results:

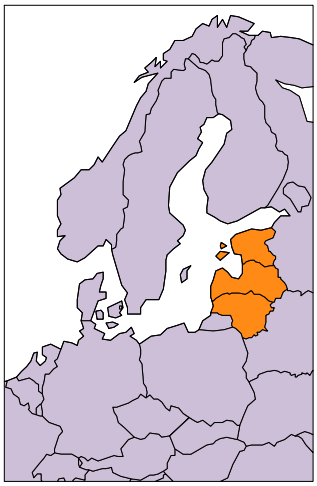
- Appropriate testing strategy for the project
- Better collaboration between acquirer and supplier on reporting and solving the problems
- Customers as testers
  - high business domain competence
  - acceptance as the main feasible testing activity

Baltic States - emerging IT power

The goal for the next 10-20 years

The three Baltic States - a "second IBM" - a unified "concern" with 120,000 highly qualified specialists.

- Baltic States - exporter of software services
- Design and maintenance of information systems and software products - a "trademark" for the Baltic States



Testing - perfect remote activity

Benefit from the time zone difference. America -Europe

San Francisco	Riga
6 p.m. code developed →	4 a.m. night
11 p.m. good sleep	**9 a.m.** code testing starts
8-9 a.m. test results received ←	**6-7 p.m.** test results delivered
6 p.m. new code developed →	4 a.m. night
…	…

Test results - overnight!

Questions?

Contact:

Riga Information Technology Institute
Kuldigas iela 45
Riga, LV-1083
Latvia

http://www.riti.lv

e-mail: Juris.Borzovs@dati.lv
Martins.Gills@dati.lv

Software Testing in Latvia: Lessons Learned

Juris Borzovs, Martins Gills

Riga Information Technology Institute
Kuldigas iela 45
Riga, LV-1083
Latvia

phone: +371-7611522
fax: +371-7619573
{juris.borzovs; martins.gills}@dati.lv

Abstract

Software testing is an integral part of software development process and one of the most efficient quality assurance methods. This paper reflects the main issues of testing in Latvia's IT industry: a brief look at the history, an analysis of current day problems and lessons learned, and the vision for the next decade. The experience has been gained from the largest local software companies.

Historically, there have been numerous research studies made on software testing, and this tradition has been integrated into quality assurance and testing principles of the local IT companies. There are two main options how to organize testing for numerous projects within a large enterprise - either by organizing this activity within the project, or by assigning this task to internally independent testing laboratory. Tester's qualification concerns, training and staff selection issues are analyzed.

Latvia's IT industry commonly with neighbor Baltic countries has an orientation towards countries with a high demand for IT solutions. Experience shows that large part of software engineering tasks can be done remotely, and the testing is geographically well separable from the rest of development.

Introduction

Geographically, Latvia is located in Europe, near the Baltic Sea, and it is often called as one of the three Baltic States. This country has a strong scientific background in the field of software testing [7]. Early research was related to automatic construction of test cases - both theoretical and practical approach. Courses on software testing are included into undergraduate computer curricula of Latvian universities for more than a decade. Recent and current research in testing is related to software test tools, universal symbolic interpretation, software process improvement (with a focus to testing issues), practical manual methods of software testing [1-6, 8]. As the IT industry began to develop decade ago, mainly from university staff, a number of lessons have been learned on how the testing

theory fits to praxis and what are the main concerns in assuring the quality within the software development projects.

The software development as an industry began to develop approximately a decade ago when in 1991 Latvian software designers won a bid for tenders to set up an information system for the social insurance of artists in the German state of Bremen. The work was done successfully, and this early achievement confirmed the fact that Latvian specialists are entirely competitive in Western markets. From year to year, the volume of information technology service exports to the West has grown.

Further milestones were the creation of the first independent test group that since has been fulfilling the tasks of testing laboratory and methodology development center, the independent testing for customers outside IT company and remotely for customers outside Latvia. In recent years, the local IT industry has considerably focused on implementing quality practices, especially ISO 9001 requirements.

Standards and quality practices

Initial approach for development groups was more ad hoc based, and the importance of testing was underestimated. Currently the picture has considerably changed - early starters are now the largest and the most experienced companies with ISO 9001 certified quality systems. The aim to raise the quality criteria was motivated by competition in IT market and by lessons from projects with unstable success. A considerable evolution of quality requirements both from the suppliers and acquirers is observable.

The way towards unified approach in the software engineering was supported by individuals who worked on adoption of IEEE standards for use in the largest IT company group. As a result, a number of company IT standards were developed, covering fields of software quality assurance, testing, specification, planning and documentation. Later, in 1996, most of them were approved as Latvian National IT standards.

In parallel to adoption of IT standards, a large portion of IT terminology was developed. Currently there are more than 4500 IT terms integrated into Latvian language [14].

For couple of years in Latvia, major companies base large business software development, especially documentation, on IEEE J-STD-016-1995 standard. Also, during product development, largest companies follow a well-defined life cycle model with defined processes (adoption of ISO/IEC 12207).

Test organization within IT company

While over the past years systematically working on testing issues, Latvian IT companies have learned a number of lessons. They are:

1) Testing in the company could be organized both by means of a separate testing institution such as laboratory or department, or by creating a small test team within each project.

2) Testing should not be regarded as just another project activity. Testers should have a background equal to system analyst or designer, plus a specific knowledge of a software testing. As a result a system with qualification levels for testing professionals has been developed. For company internal training the knowledge area is identified similar the SWEBOK requirements [13]. Also, the development of Software Engineering Professional Education project gave results of identifying a knowledge area for testing professionals [9].

3) A well-defined test process is the only way to achieve quality with minimal time and staff resources.

4) Many software acquirers are poorly informed about testing. To avoid the project failures one should educate the customer about the testing and quality assurance.

5) Although test automation has considerably increased over the past couple of years there is no evidence that it can replace the manual testing activities. Tools for testing can give a strong support, but the test planning and design are almost purely creative activities that do not undergo the automation. Tasks like GUI, user manual testing, help or localization testing almost purely rely on manual testing effort. Just like there is no software that writes the code (semantically), there is not one for test creation.

6) Along with the overall awareness of software testing, the unsatisfied demand for testers may become higher than for any other IT profession;

From the organizational point of view, **testing laboratory** existence is practiced only in the largest local IT companies. The unique feature is the availability of resources for testing. This organizational unit concentrates the testing professionals, also the experts in some certain business area, and they are ready to provide the methodological support for projects. Typically, when a request is received from a project that it will require the testing, a person is assigned to evaluate the situation and to plan the further testing activities. In this case the laboratory staff may plan the testing where active participants are also the developers, but there will be a certain set of questions that will be evaluated independently.

Quite different is the case when projects themselves perform the testing tasks. According to this scheme system analysts and programmers within project try to swap the roles for some time to review and evaluate the achieved results. Here "the tester" is not a separate person, but merely a temporary role. It may be difficult to achieve a real benefit when there is a small team where everyone knows already everything about the problems, and there is a lack of fresh look at the problem.

If testing is made inside the development project, may be with a help of testing laboratory, the real gain is that **internal testing** begins early - one can start as early as first requirements are defined. The specific feature of this testing is the reduction of documentation requirements. Authors have found out that in some cases the strictly set

requirements for documentation can be relaxed. For example, test plan can be part of internal project or quality plan, tests and test cases are not specified at the very detail, but only the most important data are kept. The form - list of tests/test cases. Results are registered in a simplified test log, but each problem is accordingly reported as a separate record in problem tracking database. Typically, there is no final report produced, and upon request the status reports can be prepared.

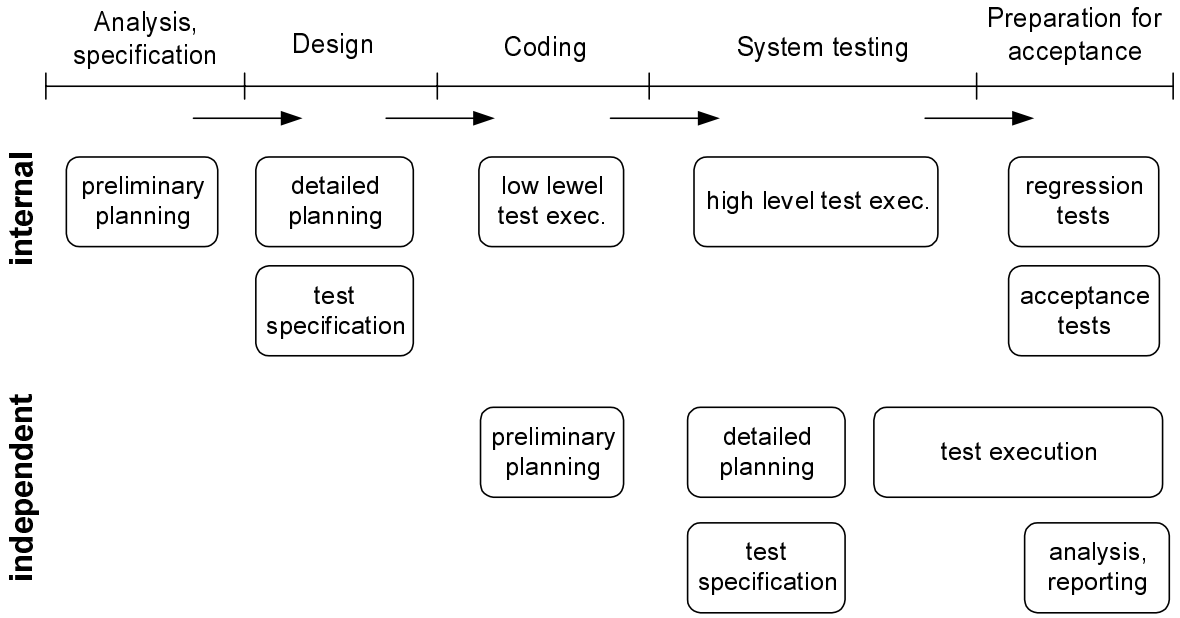


Figure 1. Internal and independent testing along with project development.

Independent testing is different with the state that it starts late, usually when the software code is available, and there the time that initially was intended for testing is mainly devoted for this independent checking and evaluation. Here, typically, the relations between the testing laboratory and supplier of the software are more formalized, contract based. Therefore documentation requirements are more strict and formal. The typical complete set consists of: plan, test description, execution result log, problem reports and test report.

Test personnel role

Looking exactly at what types of testing are practiced at various test levels and who performs this, one can see the dominance of independent testing for higher test levels (Table 1).

Tester's duty is to identify problems. The criteria can be based on one source of information (e.g. some kind of software description), but it is better when there is a possibility to compare various sources of it. For example, one can include into scope user's requirements, obtain information from system analyst and see the programmer's interpretation of it in the form of source code. Adding his/her own judgement, tester produces reports on quality of the software. This basic collaboration mechanism is the one that dominates inside project testing process.

Table 1. Tasks and test levels

Test level	Structural tests	Functional tests	Performed by
Unit testing	+		developers
Integration testing	+	+	developers
System testing		+	developers, independent testers
Qualification testing		+	developers, independent testers
Acceptance testing		+	customer, independent testers

There are number of questions developers usually expect from testers to solve. They are:
- fast integration into project scope;
- recommendation for QA and testing, adopted to local needs;
- problem/bug reports with competent evaluation of the cause and consequences;
- introduction of test automation.

Project test team has to define test process, ensure quality assurance activities within project, maximally independent approach and support to prepare system for user acceptance. There is a certain optimal body of knowledge for a typical tester. The scope should cover:
- basic concepts and definitions of testing,
- analysis of test levels,
- test techniques,
- test automation,
- test related measures.

Developers usually expect from testers expertise knowledge in all project technical questions, and that requires latter to learn a lot of new facts and techniques.

For some the learning is an easy task and especially for those who have a diversified background of knowledge. The unprofessional impression of testing may lead to an idea that testing is just a play with software, and there is no need to know programming at all. Closer examination may reveal that this activity is quite technically related, and there is no place for technically unqualified. The truth is somewhere between. Looking at personnel profile of a real testing laboratory one can see that part has a long IT and software engineering experience, quite many are recent university graduates, some have temporary worked in non IT sphere, and there are also colleagues who had not had any experience in software development. Surprisingly, but the latter mentioned factor is not a minus. Plus is when IT and non-IT experience is combined. Experience is very important. One can train to certain methods quite quickly, and for easy tasks there one cannot observe any difference. But analysis-consuming tasks reveal the worthiness of a broad knowledge and original approach to problems.

Qualification issues of testers

Looking at development activities inside a project, it is important to identify the tasks for the tester within each development stage (Table 2).

Table 2. Tasks of tester along with project phases.

SW development activity	Tasks for tester
System analysis, specification of requirements	Verify whether requirement specification and other documents are suitable for testing. Plan the testing: its tasks and schedules.
Design	Plan in detail the tests: type, coverage. Make test descriptions.
Coding	Refine plans and test descriptions. Execute tests. Make automation and regression tests.
Testing	Thoroughly implement the system, installation and documentation testing. Do regression testing.
Prior to delivery (preparation for the acceptance)	Check-up the corrections. Run the regression tests.

Description of these tasks may be more or less formalized. For example, company DATI has developed a test process description. It covers various aspects of testing, presenting procedures, guidelines, templates and examples.

Table 3. Qualification levels.

Level	Tasks	Required skills
1	Execution of previously specified tests and other tasks that do not require special knowledge or training	Computer usage skills - advanced used level and ability to understand specified tests.
2	*Tasks of 1st level +* Test description according to system documentation.	*Skills of 1st level +* Ability to analyze system documentation and to write a testing documentation.
3	*Tasks of 2nd level +* Test planning and managing of test team	*Skills of 2nd level +* Ability to identify required tests, skills in team and resource management.
4	*Tasks of 3rd level +* Training of other testers and development of testing methods	*Skills of 3rd level +* Tutor skills and expert knowledge in testing theory

For over a year one of Latvian IT companies is working on introduction of the qualification level scheme among testers. There are four levels in total (Table 3), and additional "0" level could correspond to an amateur tester - a person who is just trying play with the software.

Such identification could serve as motivator for developing the tester career, and at the same time it could be a valuable means for locating the right people for various types of testing projects.

Training

Although there are possibilities to attend a course on software testing in some of Latvian universities that does not provide enough knowledge and skills to start practical activities. Therefore companies practice internal training for people related to testing. For example, RITI has developed curricula for three different audiences: new testers, existing testers and programmers. Each group has a slightly different motivation, and, respectively, there is a different composition of topics covered. There are predefined training programs ranging from 10 till 26 hours. In calendar terms it may range from two days till couple of weeks.

The course for new testers mainly focuses on the testing basics, and it is being customized to the initial knowledge level in testing and software engineering in general. The aim of such course is not just to prepare a team that executes previously specified tests (level 1), but also to train the candidates for qualification level 2. The training is composed of formal lectures, workshops, tasks to solve problems, individual study of guideline documentation, and at the very end of course there is a final test.

Practicing testers are interested to improve their knowledge, to learn some new common principles and to gain the knowledge about other duties of testing professional. Typically, these courses are held in a form of workshops where people both educate themselves and fulfill the tasks of tutor.

Testing course for programmers is oriented to add the knowledge for efficient development rather than for professional testing activities. These courses are accorded to the type of projects the trainees are working in.

Taking into account that learning pure theory is not sufficient for training, a work with a special software is included into course. It is devoted to develop and to check the testing skills, but actually it is not a program that somehow specifically tests the person, it is a base for sample testing. Everybody has the opportunity to go through all the main stages of testing - planning, specifying tests/test cases, executing them, reporting problems, summarizing the results. The current software is a small database application that fulfills cargo delivery accounting functions. It has several complexity versions - covering only some or a complete set of business functions, and with various types of bugs integrated. Each student may have a specific task of what exactly has to be tested. According to this task, the solutions are being sought. Both functional and structural methods may be applied, but typically only the black-box approach is practiced. This training software proved to be excellent for students to see what they really have learned.

Still looking at candidates for test team, one has to realize that the testing professional is a software engineer specialized in testing. Anybody can be trained fast for some basic activities. Having historically formed testing laboratory with a considerable proportions of

people that know well also some other area, not related to IT, authors have observed an interesting correlation - the more diversified knowledge people have, the more successful is the testing. At the same time our IT field is quite strict in terms of quality records and application of some well-planned methods. Therefore although the testing itself may be considered as art, the persons who participate should not be artists. It still is more advisable to have the technically oriented people.

Alongside with the positive experience of forming a good test team, there are also lessons of bad experience. A typical problem is that some managers want to move the low qualified programmers to testing, thus thinking that they have found a perfect solution in terms of resources. But the reality is that bad programmer is also a bad tester. This is both true for voluntary solutions when some manager wants to give a temporary punishment for inadequately worked person. Such approach does not give any good results. Only the opposite. Projects may loose trust in testing personnel, and the widely unspoken idea that testing is less prestigious activity than programming just strengthens.

Another fault is approaching the task very independently - in a way that lots of information is being lost or ignored just because of unwillingness to become too familiar with the system, its development issues and to compare the developer's point of view with the one of customer. Also, there may be testers, especially starters who over a time have not broadened their view on testing. They may expect that always there will be someone who will prepare systematic test cases, and the solely task then would be to execute them or to hunt some problem without a systematic ground beneath.
One of the main problems in developer-tester interaction may be the overestimation of testing. Sometimes developers expect too much, and the cause sometimes is that testers have promised too much. Some myths have to be cleared prior to real project.

Selecting the personnel for test team sometimes is tied with questions about the career perspectives in this profession. Quite a lot of people are slightly afraid that they could be destined to work in this field for all the duration of the project. There is a hidden stereotype that finding bugs requires lower qualification that writing the code (and often with lots of errors in it!). Actually, a qualified tester has both to be an expert in programming and in the application area of the software, there has to be a lot of analysis knowledge applied, and rarely one could say that this is easier than implementing the functionality. The obvious career steps may be related to reaching different professional levels (Table 3).

Another important point - customers. IT companies have experienced a considerable increase of quality awareness from customer side, but at the same time they may have a lack of important software engineering concepts or misinterpretation of some quality principles. Therefore for some time there are courses held both on software quality issues and for general information about testing. The typical problem may be that there is a principal understanding that testing is necessary, but one may fail to recognize the scale of necessary time and resources, as well as related activities to make the testing become efficient.

Future vision

According to various studies, there is a shortfall of up to 500,000 software specialists in the world right now, and the deficit may increase to some 1 million people over the course of the next 5-10 years [10-12]. Latvia's vision for this decade is closely related to rapid development of IT industry. Rough estimation shows extreme potential of the three Baltic states to become a major software development and maintenance region able to employ up to 120 thousands IT engineers and to export software and maintenance services.

Analyzing the software engineering activities that can be performed remotely from the acquirer, the main ones are: design, coding, testing, maintenance, reengineering and porting. Special case of remote software development is due to time zone difference. For example, time difference between America and Europe makes perfect solution for test result delivery overnight thus giving the daily quality record for software under development. Taking into account the overall increase of the awareness of the role of quality assurance and independent testing, the software testing is the IT industry sector to experience a rapid development.

References

1. J.M.Barzdins, J.J.Bicevskis, A.A.Kalnins. Construction of Complete Sample Systems for Correctness Testing._ In: Mathematical Foundations of Computer Science, Berlin: Springer, 1975, pp. 1-12.
2. J.M.Barzdin, J.J.Bicevskis, A.A.Kalninsh. Automatic Construction of Complete Sample Systems for Program Testing._ In: Proc. IFIP Congress, 1977, North-Holland, 1977, pp. 57-62.
3. A.Auzins, J.Barzdins, J.Bicevskis, K.Cerans, A.Kalnins. Automatic Construction of the Test Sets: Theoretical Approach._ In: J.Bārzdiņš, D.Bjorner (eds.) Lecture Notes in Computer Science: Baltic Computer Science, No. 502, Springer-Verlag, 1991._ pp.286-359.
4. J.Bicevskis, J.Borzovs, U.Straujums, A.Zarins, E.F.Miller,jr. SMOTL - A System to Construct Samples for Data Processing Program Debugging._ IEEE Transactions on Software Engineering, vol. SE-5, No.1, 1979, pp. 60-66.
5. J.Borzovs, A.Kalnins, I.Medvedis. Automatic Construction of Test Sets: Practical Approach ._ In: J.Barzdins, D.Bjorner (eds.) Lecture Notes in Computer Science: Baltic Computer Science, No. 502, Springer-Verlag, 1991._ pp.360-432.
6. Z.Bicevska, J.Bicevskis, J.Borzovs. Regression Testing of Software System Specifications and Computer Programs._ Proceedings of the 8th Software Quality Week, San Francisco, 1995, paper 5-T-1 (9 p).
7. J.Borzovs, M.Gills, A.Adamsone, S.Linde, J.Plume. Software Testing in Latvia: Lessons Learned. 1st International Conference on Software Testing ICSTEST - Conference Proceedings. Bonn, April 5-7, 2000. Track A, Paper 2. p 15.
8. J.Borzovs, M.Gills. Building-up a team for manual testing. To be published in Conference Proceedings of 2nd International Conference on Software Testing ICSTEST -. Bonn, April 4-6, 2001.

9. B.Apine, J.Borzovs, A.Jautrums, A.Joma, E.Kalnina, A.Klints, S.Linde, J.Plume, U.Sukovskis, M.Vitins. The Future of IT Professional Education in Latvia._ Proceedings of Intern. Conf. IT Skills- & Vocational Certification, Tallinn, Estonia. 2000, pp 40-43.
10. W.Strigel. What's the Problem: Labor Shortage or Industry Practices?_ IEEE Software, no. 3 (May/June) 1999, pp. 52-54.
11. C.Jones. The Euro, Y2K, and the US Software Labor Shortage._ IEEE Software, no. 3 (May/June) 1999, pp. 55-61.
12. S.Baghchi. India's Software Industry: The Peoples Dimension._ IEEE Software, no. 3 (May/June) 1999, pp. 62-65.
13. Guide to the Software Engineering Body of Knowledge. http://www.swebok.org
14. Terminology, Riga Information Technology Institute. http://www.riti.lv/en/terminology.htm

QW2001 Paper 7A1

Dr. Holger Schlingloff & Dr. Jan Bredereke
(Technologie-Zentrum Informatik)

Specification Based Testing Of the UMTS Protocol Stack

Key Points

- Development of flexible testing environment for UMTS protocol stack software
- Black-box testing of parallel real-time systems based on formal specifications
- Testing results prove increase in performance/cost ratio by automated test case generation

Presentation Abstract

Our paper is organized as follows: first, we present a brief introduction to specification based testing and to the UMTS protocol stack. Then, we give an overview of the functionality and properties of the RLC layer, and its implementation in SDL. The main part deals with out automated testing environment: formal CSP specifications for the RLC; interfacing between SUT and RT-TESTER, and testing of multiple instances in parallel and real-time. Finally, we describe and interpret the testing results and summarize our work.

About the Author

Bernd-Holger Schlingloff currently is managing director of the Bremen Institute of Safe Systems within the Center of Computing Technologies in Bremen University. He received his PhD in 1990 from the Technical University of Munich. After that, he spent a year at Carnegie Mellon University. He then became assistant professor at the computer science department of the Technical University of Munich, and in 1996 transitioned to Bremen. His research interests include software quality assurance, logic in computer science, and the application of formal methods to industry projects. He has written several articles and surveys on temporal logic model checking, and recently completed a book on partial state space analysis of safety-critical systems.

Jan Bredereke received his PhD degree (Dr. rer. nat.) in computer science from the University of Kaiserslautern, Germany, in 1997, and his Diploma in computer science from the University of Hamburg, Germany, in 1992. From 1992-93, he became a research assistant there, and from 1994-97, at the University of Kaiserslautern, in the group of Reinhard Gotzhein. From 1997-98, he was a

post-doctoral fellow at McMaster University, Canada, in the Software Engineering Research Group of David Parnas, and from 1998-99, he was a researcher at the University of Oldenburg, Germany, in the semantics group of Ernst-Ruediger Olderog. Since 1999, he works at the Bremen Institute of Safe Systems, Germany, in a project with Siemens on UMTS. His current research interest is in the design of telecommunication systems with formal methods.

Specification-Based Testing of the UMTS Protocol Stack

Jan Bredereke and Bernd-Holger Schlingloff
Bremen Institute of Safe Systems
University of Bremen, Germany
{brederek,hs}@tzi.de

Quality Week, San Francisco, May 31st, 2001

Bremen Institute of Safe Systems

- Technology transfer institute within Bremen university
- 35 scientists, 5 professors, 1.1 M Euro annual revenues
- Application of formal methods in industrial contexts
- Verification projects: Airbus, German Aerospace (EADS), OHB Satellites, DLR, SiemensVT, INSY Rail, *Siemens Telecom*, ...

Introduction

Current methods for testing embedded real-time control software:

- Structural or code-based
- Specification based

Application to the *Radio Link Control* (RLC) protocol layer of the *Universal Mobile Telecommunication System* (UMTS).

- Siemens AG, Salzgitter: code development
- Technologie-Zentrum Informatik (TZi), Bremen: testing support

UMTS: distributed development of standards, user equipment and base stations.
Consistency checks cannot be based on particular implementation!
$\Longrightarrow$ **specification based testing rather than structural testing!**

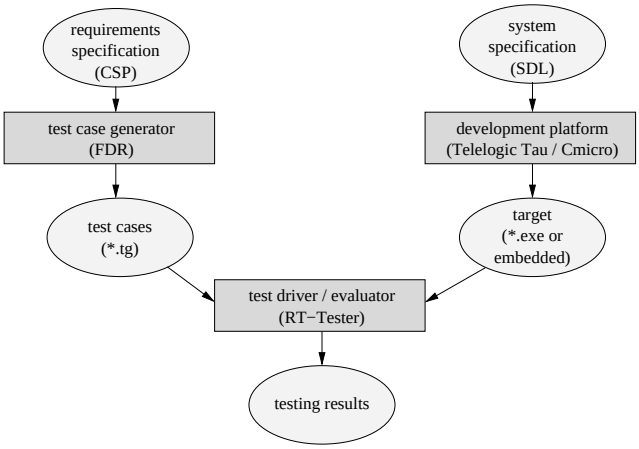
Specification Based Testing

Specification based methods:
treat system under test (SUT) as *black box*
focus on *properties* of the system.

Advantages:

- Concentration on functionality aspects
- Exhibition of ambiguities in the requirements
- Detection of misinterpretations, omissions and missing cases
- Generation of arbitrary length test scripts
- Reusability and maintainability

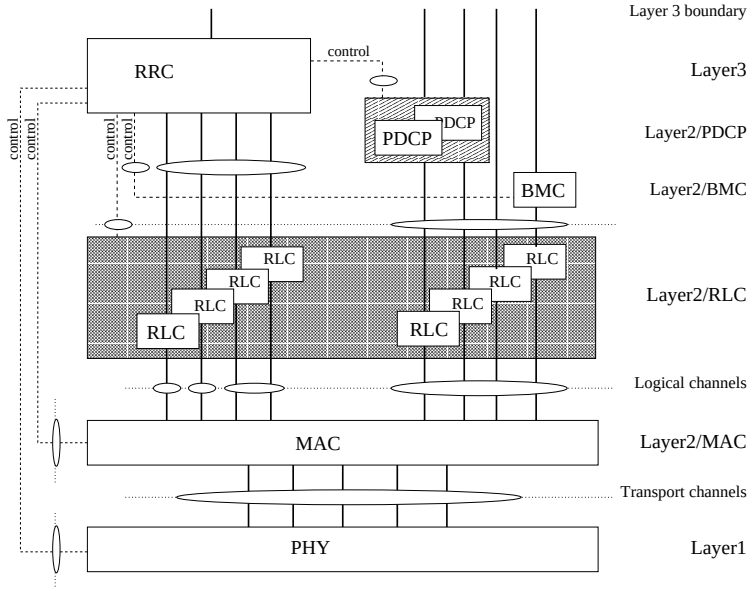
Testing Approach



High degree of automation reduces the overall development time!

UMTS Protocol Architecture

UMTS standard as developed by 3GPP consortium:



UMTS Protocol Architecture (cont.)

- *Layer 1: physical layer*; hardware services provided by the chip-set
- *Layer 2: data link layer*; provides point-to-point connection concept
  - *Medium Access Control (MAC)*; provides unacknowledged transfer of service data units, reallocation of parameters such as user identity number and transport format, and local measurements such as traffic volume and quality of service indication
  - *Radio Link Control (RLC)*; segmentation and reassembly of long data packets from higher layers into fixed width protocol data units, respectively. This includes flow control, error detection, retransmission, duplicate removal, and similar tasks
  - *Packet Data Convergence Protocol (PDCP)*; *Broadcast / Multicast Control (BMC)*
- *Layer 3: network layer*; provides network services such as establishment / release of a connection, hand-over, broadcast of messages in a geographical area, and notification of information to specific users. Radio Resource Control (RRC) assigns, configures and releases wireless bandwidth (codes, frequencies etc.)
- *Layer 4+: application layers*; e.g. Call Control (CC) and Mobility Management (MM)

The RLC Layer of UMTS

Tasks:

- Correction of *transmission errors*
- *Segmentation* of variable-length data packets received from the upper layer into fixed-length PDUs
- *Reassembly* of received PDUs according to the attached sequence numbers
- Optional *cipher mechanism* preventing unauthorized access to message data

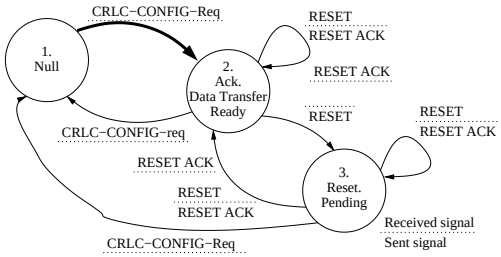
Three modes of data transfer:

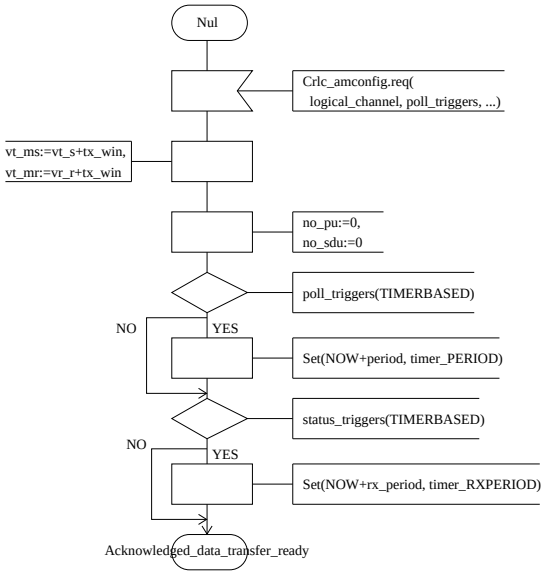
- *Acknowledged mode* (error-free transmission guaranteed)
- *Unacknowledged mode* (immediate; deletion of erroneous and duplicate packets)
- *Transparent mode* (unchanged data transfer)

Possibility of several RLC instances at the same time!

Implementation in SDL

3GPP standard is given in a *mixture of formalisms*: plain English, annotated figures, bitmap-tables, state transition diagrams, message sequence charts, SDL diagrams





Formal CSP Specifications

CSP (Communicating Sequential Processes):

- Abstract description of *requirements*
- Especially designed to describe the behaviour of *parallel*, *communicating*, *embedded real-time* systems
- Operators reflecting the *structure* of requirements,
  e.g., *sequential* and *parallel* composition, *choice*, *iteration* and *hiding*
- Communication by (synchronous) *exchange of events*, real-time modelling with *timers*
- Rich and well-established *theory* ([Hoare 1978], [Hoare 1985], [Roscoe 1997])

Example: Vending machine

```
include "timers.csp"

pragma AM_INPUT
channel coin, buttonCoffee, buttonTea
nametype MonEv = { coin, buttonCoffee, buttonTea }

pragma AM_OUTPUT
channel coffee, tea
nametype CtrlEv = { coffee, tea }

OBSERVER = (  (coin -> HAVE_COIN)
           [] (buttonCoffee -> OBSERVER)
           [] (buttonTea -> OBSERVER))
HAVE_COIN = (  (coin -> HAVE_COIN))
            [] (buttonTea -> AWAIT({tea}) ; OBSERVER)
            [] (buttonCoffee -> AWAIT({coffee}); OBSERVER)

RANDOM_STIMULI = (|~| x: MonEv @ x -> PAUSE; RANDOM_STIMULI)

TEST_SPEC = RANDOM_STIMULI [| MonEv |] OBSERVER
```


CSP Testing Tool RT-Tester

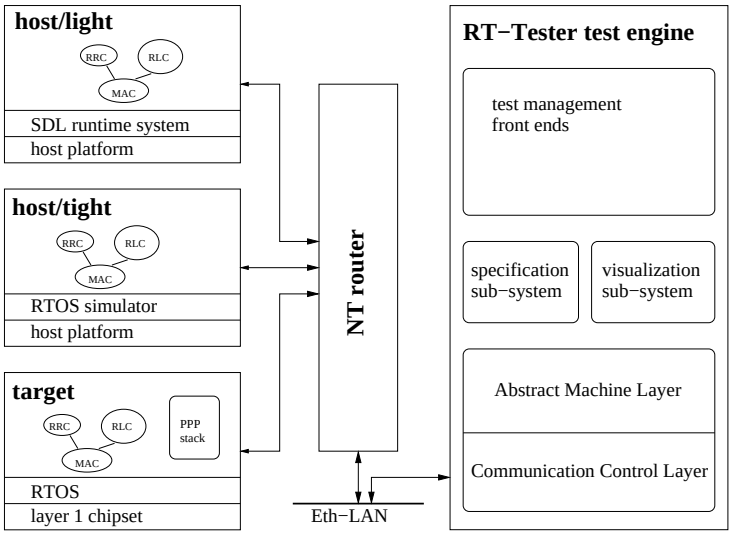
Joint development; now distributed by Verified Systems International GmbH, Bremen.

- Hardware-in-the-loop and component *black box tests*
- Tests of *arbitrary length* with steadily increasing coverage
- Testing of both *functional* and *hard real-time* properties
- Various specification and visualization possibilities

Execution of formal CSP test specifications proceeds in two stages:

- Automatic transformation of CSP input into *graph of admissible state transitions*
- *Generation*, *execution*, *monitoring* and *evaluation* of test sequences
  (automatically and in real time)

Configurations for Testing during Development



Dealing with a Moving Target

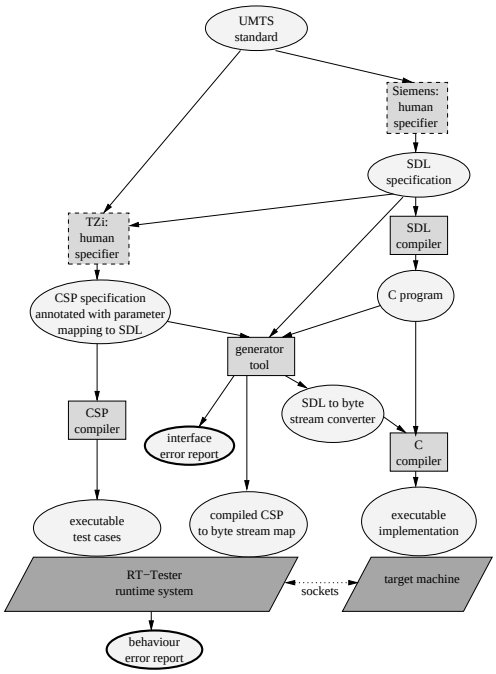
Problem: Both standard and implementation still subject to considerable change!

- Parameters of service primitives
- Details of internal data structures
- Behaviour of the protocol machines, in particular for error handling
- Machine representation of data at the interfaces

Solution: highly flexible testing environment

- *Interface definitions in terms of SDL* signals and data structures instead of low level descriptions,
- *Automated consistency check* between the SDL description of the interface and the formal CSP specification of the interface
- *Modularization* of formal behaviour specifications into largely independent functional requirements.

Automated interfacing of SDL and CSP



An example for matching CSP channels with SDL signals

CSP

```
nametype Rb_identity = {0 .. maxRb_count}
datatype Rlc_data = dummy_value
channel rlc_tr_data_req :
pragma SDL_MATCH PARAM 1!RB_Id
      Rb_identity .
pragma SDL_MATCH TRANSLATE dummy_value 0x99 * 16
pragma SDL_MATCH PARAM 1!RLC_SDU_Data SUBSET_USED
      Rlc_data
pragma SDL_MATCH SKIP 1!length DEFAULT_VALUE 16
```

SDL

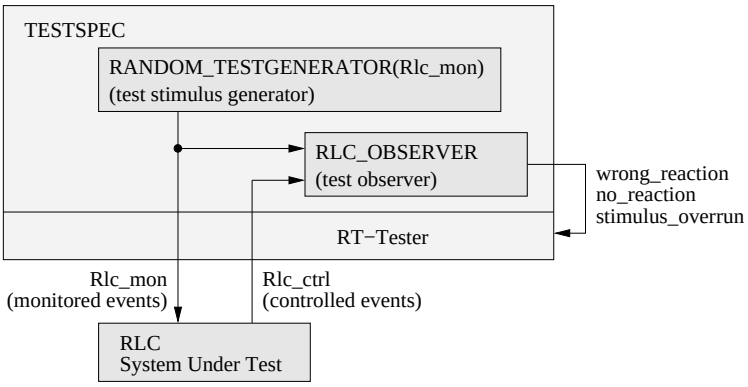
```
syntype RB_Identity = integer
      constants 0:MaxRb_Count
endsyntype;
synonym RLC_MAX_SDU_SIZE integer = 512;
newtype RLC_SDU_A
      carray(RLC_MAX_SDU_SIZE, octet)
endnewtype;
newtype RLC_SDU struct
      RB_Id RB_Identity;
      RLC_SDU_Data RLC_SDU_A;
      length integer;
endnewtype;
signal RLC_TR_DATA_Req(RLC_SDU);
```

Compiling Target and Test Scripts

- The mapping of *CSP events* to *byte strings* for the interface adapter in the RT-TESTER runtime system is produced automatically from the CSP specification.
- The *SDL specification* is compiled into a *C language program*, including C language header files and a *C language template file* for the input and output of SDL signals from and to the environment, respectively.
- This template is filled by automatically inserting code for the test interface adapter. In particular,
  - the *SDL signal* and data type definitions are *matched* with the annotated *CSP channel* parameters, and
  - the *byte representation* is determined from the generated C language header files and corresponding C language data types and parameter names
- The *template and C files* are compiled into an *executable implementation* for the target machine, and the *CSP specification* is compiled for the RT-TESTER *runtime system*.

Modular Structure of the Formal Behaviour Specification

Example: separation into *test stimulus generator* and *test observer*



Formal CSP Specification of RLC Connection Initialization

CSP code for part of the initialization of a connection in acknowledged mode:

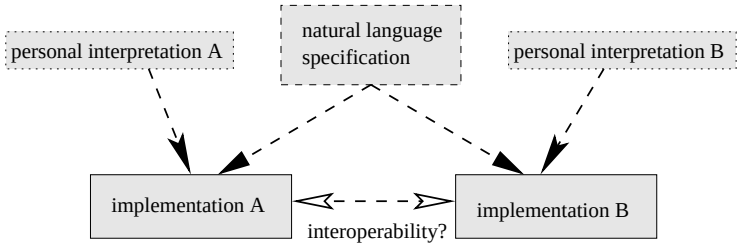
```
-- The null state of the RLC (AM) entity:
RLC_AM_NULL(instance_id) = instate_rlc_am_null.instance_id ->
(crlc_config_req.rbSetup.1.instance_id?dummy ->
        RLC_CONFIG_AM(instance_id,rlc_to_rrc)
 [] ([] x : diff(Rlc_mon,
            {| crlc_config_req.rbSetup.1.instance_id |}) @
        x -> RLC_AM_NULL(instance_id))
 [] ([] x : Rlc_ctrl @ x -> warn_spontaneous_event -> RLC_AM_NULL(instance_id))
)
```

Comment: The observer for the RLC instance with the number `instance_id` starts in the state `RLC_AM_NULL` and waits for a configuration request event. If the event occurs, the instance goes to the next state. All other events to the SUT are ignored. Any spontaneous output from the SUT would be an error and is flagged.

Management of Test Specifications

- **Separation of testing concerns:**
  - different layers of the UMTS protocol stack
  - data type definitions, channel definitions, and behaviour definitions
  - each property of the SUT in a separate requirements module
- **Implementation of various testing strategies via CSP:**
  - *completely random* selection of test stimuli
  - random choices with different *probabilities*
  - fixed, explicitly specified *trace* of events
- **Family of test specifications in a directory tree structure**
  - specification modules, stimulus generators, test suites etc. separated
  - need for intelligent configuration facilities

Testing Results 1: Ambiguities in the Standards Document



Example: In the implementation, a split of RLC SAP depending on the destination of a PDU was made (which is obvious but not standardized); as a consequence the RLC can be used only with MAC and upper layers which add the same parameter and which use the same SAP split.
Documentation of these design decisions can help in the integration phase.

Testing Results 1: Ambiguities in the Standards Document (cont.)

No precise definition of the properties of a SAP in the standard

- Buffer lengths, queueing discipline (FIFO, . . .)
- Queue delivery dependences between different SAPs
- Minimum/maximum delays between delivery and availability
- Handling of signals that cannot be received

Design decisions on these items might set a de-facto standard.
Documentation of the ambiguities can help to adapt to alternate implementations.

Testing Results 2: Deviations in the Behaviour

- Example for an *unexpected (spurious) reaction*:
  - an RLC protocol machine is created by the event `crlc_config_req.rbSetup` from the upper layers
  - becomes operational after receiving the event `mac_status_ind` from the underlying MAC layer
  - immediately after that, the test stimulus generator randomly generates another (nonsensical) `mac_status_ind` event
  - the RLC layer sometimes, but not always, reacted by generating a data packet, i.e., with a `mac_data_req` event
- There were also *interactions between different instances* of the RLC protocol machines; *deadlock situations* in spite of error recovery mechanisms appeared
- *Safe return* to an initial state was not always guaranteed

Summary

- For quality assurance, specification of *requirements* should be in a formal *(unambiguous)* language and *independent* from the implementation
- *Automatic generation and execution of tests* from these specifications with appropriate testing tools can reveal subtle (otherwise undetectable) errors
- *Automating the connection* between the *system under test* and the *requirements specification* allows maximal flexibility in the design process

Specification Based Testing of the UMTS Protocol Stack

Jan Bredereke Bernd-Holger Schlingloff

Universität Bremen, TZi · P.O. box 330 440 · D-28334 Bremen · Germany
{brederek,hs}@tzi.de · www.tzi.de/{~brederek,~hs}

Abstract. We present a specification based testing setup for the RLC layer of the UMTS protocol stack. Requirements are specified in the formal language CSP. From this we automatically generate real-time test scripts for the RLC, which is developed from SDL sources. Our testing approach is highly adaptable to changes in the UMTS standard and implementation: we developed an interface code generator with automated consistency checks, and modularized the requirements according to functional properties. We report on testing results and experiences with this setup.

1 Introduction

Current methods for testing embedded real-time control software can be classified as structural or specification based. *Structural testing* methods try to execute as many different parts of the *program code* as possible, where coverage is measured in terms of statements, conditionals, branches, function calls, and so on. *Specification based* methods treat the system under test as a black box and focus on testing the required *properties* of the system.

We have applied the latter approach in the development of the Radio Link Control (RLC) protocol layer of the Universal Mobile Telecommunication System (UMTS), a new generation of high-speed, multi-media mobile phone systems. The work is part of an ongoing cooperation between Siemens AG, Salzgitter, and Technologie-Zentrum Informatik (TZi), Bremen, where Siemens develops the code for the RLC layer, and TZi provides testing support.

In the case of UMTS, a standard is being defined by the 3GPP consortium (the 3rd Generation Partnership Project [1]). User equipment and base stations are to be developed by different companies and at different sites. Moreover, even the development of the software for different layers of the protocol often is distributed between several teams. For the correct functioning of the whole system, it is extremely important that the standard is implemented by all participating developers in a consistent way. Therefore, in order to ensure inter-operability between devices from different providers, it is mandatory to base test suites solely on the 3GPP standards (plus additional site-specific requirements) rather than on individual program code from specific developers. For such systems, specification based testing is more appropriate than structural testing.

Specification based methods treat the system under test (SUT) as a *black box* and focus on testing the required *properties* of the system. This has a number of significant advantages:

- the testing process concentrates on the user requirements and functionality aspects rather than on implementation details,
- ambiguities of the informal requirements (here, the UMTS standard) are exhibited,
- misinterpretations can be found, including errors arising from omissions and missing cases,
- a formal requirements specification implicitly contains test scripts of arbitrary length, the test coverage is limited only by the time available for a test run,
- a change in the SUT does not affect the test suites, and a change to a requirement affects one requirements module only.

In this project, without even running the tests, already in our first formalizations we found a number of deviations between our interpretation of the standard and the actual implementation. For example, frequently problems stem from cases which are under-specified in the standard (i.e., 3GPP did not state precisely what the required reaction to certain sequences of inputs should be) and which are interpreted differently by different readers. An annotated list of such deviations is a valuable documentation of design decisions arising from different views onto the standard. A formal specification can even be used as a reference which helps to achieve consistency and correct interoperability between components developed at different sites.

In conventional testing approaches, test cases are often formulated using a set of *test scripts*. These are explicit sequences of test inputs and of expected system outputs, describing in a step by step manner how the test should proceed. Specification based approaches do not need these long test scripts. Instead, the test scripts are implicitly contained in much shorter *formal specifications*. Tools can expand their powerful choice and concurrency operators automatically on the fly to conventional test scripts. They may run over long periods of time: hours, days, weeks and more - without the necessity of manually writing test scripts of an according length. Test results are evaluated on the fly in real time during the run of the SUT.

With structural testing, all scripts have to be revised after each change in the SUT. Therefore, the testing process is costly and time-consuming. In specification based testing, due to its black box nature, all test cases can be re-used even if the implementation is changed. Thus errors can be corrected and regression tests can be done at virtually no additional costs. Since in the distributed development of the UMTS protocol stack inconsistencies are very likely, this feature is extremely important.

If some requirement is changed, with structural testing this usually means that several steps of several test scripts have to be changed, such that many test scripts need to be re-worked. In specification based testing, all these points of change are folded into the same few lines of the formal specification of this requirement. The formal specification is modular, each module describing a different aspect of the system's behaviour. Only the module which describes the new requirement has to be updated. For the UMTS protocol stack this is especially important since a number of items in the interpretation of the 3GPP documents are expected to be subject to change at any time.

Figure 1 describes our overall approach, where requirement specifications are formulated in CSP (Communicating Sequential Processes) [2], and system specifications

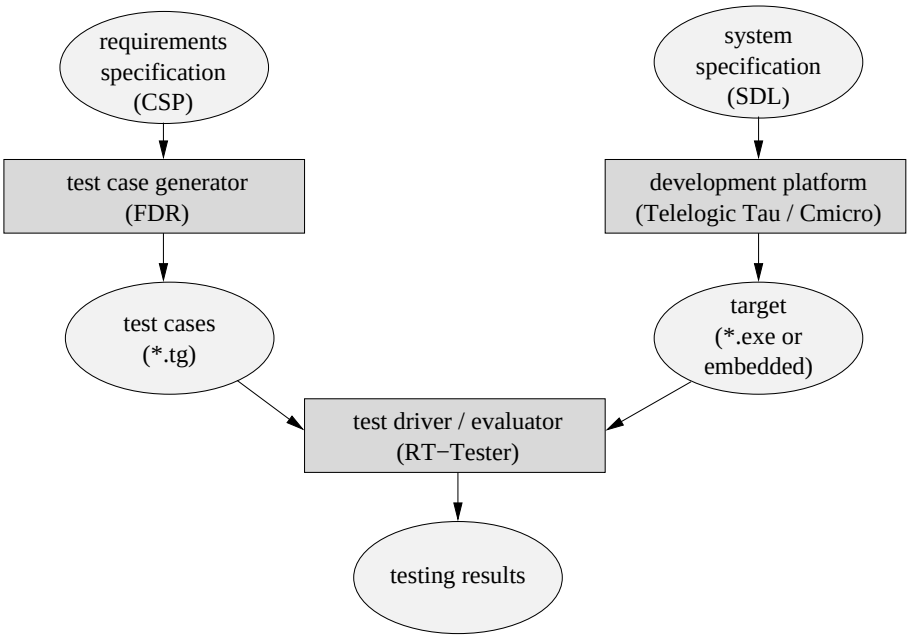


Fig. 1. Overall specification based testing approach.

are formulated in SDL (Specification and Description Language) [3]. From this, automatic tools are used for the generation of code and for the generation of test cases, and the system is tested automatically. This high degree of automation reduces the overall development time, which is particularly important in the current UMTS race.

Our paper is organized as follows: in Section 2, we give an overview of the functionality and properties of the RLC layer, and its implementation in SDL. Section 3 is the main part and deals with our automated testing environment: formal CSP specifications and the testing tool RT-TESTER, interfacing between the SUT and the testing tool, the formal CSP specification of the RLC layer, and testing of multiple instances in parallel and real-time. In Section 4, we describe and interpret the testing results, and in Section 5 we summarize our work.

2 The RLC Layer of UMTS

UMTS is a new international wireless telecommunication standard developed by the 3GPP consortium [1]. The standard comprises a layered architecture, where each layer relies on primitive services from the layer below and provides complex services to the layer above. Conceptually, each layer in the user equipment communicates with the same layer in the UMTS terrestrial radio access network.

- Layer 1 is the physical layer of hardware services provided by the chip-set.
- Layer 2 is the data link layer. It provides the concept of a point-to-point connection to the network layer above.
- Layer 3, the network layer, provides network services such as establishment / release of a connection, hand-over, broadcast of messages to all users in a certain geograph-

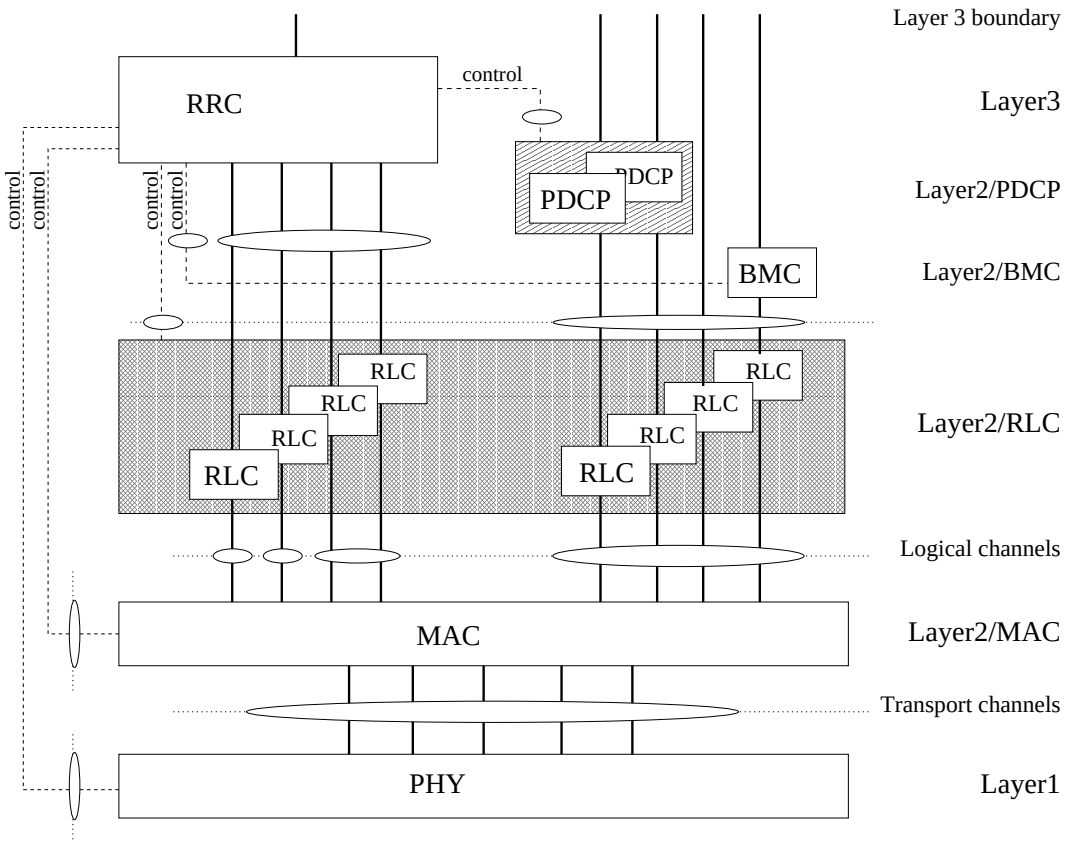


Fig. 2. Overall architecture of the UMTS radio interface protocol stack.

ical area, and notification of information to specific users. It includes the Radio Resource Control (RRC), which assigns, configures and releases wireless bandwidth (codes, frequencies etc.).

– Above layer 3 there are application layers containing functionality such as Call Control (CC) and Mobility Management (MM).

Layer 2 is split into several sub-layers: Medium Access Control (MAC), Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP), and Broadcast and Multicast Control (BMC). The MAC provides unacknowledged transfer of service data units, reallocation of parameters such as user identity number and transport format. It furthermore reports local measurements such as traffic volume and quality of service indication to the RRC. The main task of the RLC is segmentation and reassembly of long data packets from higher layers into fixed width protocol data units, respectively. This includes flow control, error detection, retransmission, duplicate removal, and similar tasks. An overview of this architecture is given in Figure 2, which is from the 3GPP standard.

2.1 Overview of Functionality and Properties

The RLC layer of the UMTS protocol stack [4] provides three modes of data transfer: acknowledged (error-free), unacknowledged (immediate), and transparent (unchanged)

mode. In acknowledged mode, the correct transmission of data is guaranteed to the upper layer; if unrecoverable errors occur, a notification is sent. In unacknowledged mode, erroneous and duplicate packets are deleted, but there is no retransmission or error correction: messages are delivered as soon as a complete set of packets is received. In transparent mode, higher layer data is forwarded without adding any protocol information; thus no error correction or duplicate removal can be done.

In all of these modes, the variable-length data packets received from the upper layer must be segmented into fixed-length RLC protocol data units (PDUs). Vice versa, for delivery to the higher layer, received PDUs have to be reassembled according to the attached sequence numbers. As additional services, the RLC offers a cipher mechanism preventing unauthorized access to message data. Thus, to transmit data, the RLC reads messages from the upper layer service access points (SAPs), performs segmentation and concatenation with other packets as needed, optionally encrypts the data, adds header information such as sequence numbers, and puts the packets into the transmission buffer. From there, the MAC assigns a channel for the packet and transmits it via layer 1 and radio waves. On the opposite side, packets arriving from the MAC in the receiver buffer are investigated for retransmissions, stripped from the RLC header information, decrypted if necessary and then reassembled according to the sequence numbering, before they are made accessible to the upper layers via the corresponding SAP.

A particular feature of the RLC is that there may be several instances coexisting at the same time. This is necessary since the services to the upper layers provide a variable number of connections, whereas the service of the lower layer provides a fixed number of logical channels. For efficiency reasons, however, the maximum number of parallel instances is statically fixed in the system.

2.2 Implementation in SDL

The 3GPP standard is written in a mixture of formalisms. The main part is plain English and annotated figures. These are accompanied by tables describing bit-level data formats, small state transition diagrams for the different modes and control commands, plus message sequence charts for the procedural communication between sending and receiving RLC instances. Earlier versions of the standard were accompanied by a detailed implementation suggestion described in the specification and description language SDL [3].

For example, in Figure 3 on the following page, we show the acknowledged mode states, and in Figure 4 on page 7, we show part of the corresponding SDL diagrams for the initialization of a connection in acknowledged mode (bold arrow in Figure 3).

The implementation is developed from these and similar sources with the help of suitable tools. In particular, there are commercial tools which can execute an SDL system in an emulated runtime environment, and which can compile a set of SDL diagrams plus a set of data type descriptions written in ASN.1 or as C headers into executable machine code for embedded targets.

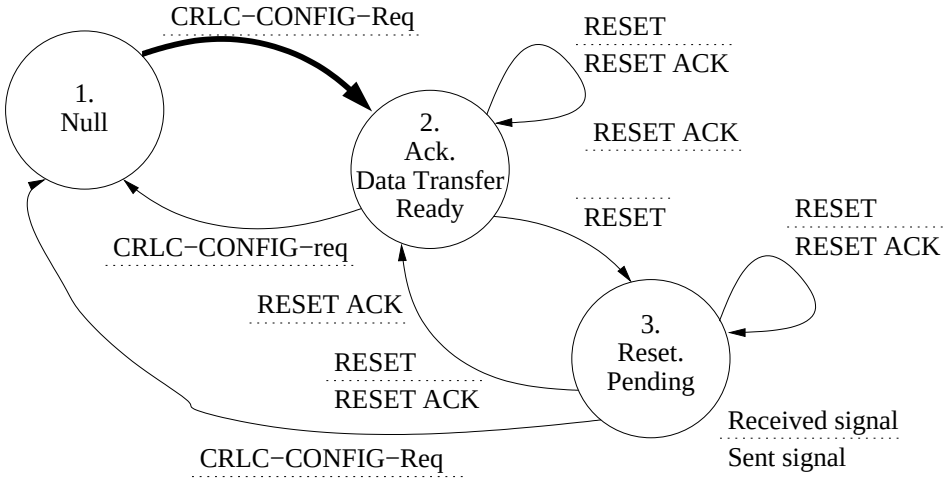


Fig. 3. RLC layer acknowledged mode states.

3 The Automated Testing Environment

In order to be able to find errors arising from misunderstandings or omission of cases, it is important that the system tests are developed independently from the system implementation. In the present project, we have formalized the requirements for the RLC using the process algebraic language CSP. These *formal specifications* describe the expected behaviour of the system under test (SUT) at its interfaces. From a CSP test specification of just a few pages, test scripts of arbitrary length are generated automatically by the supporting tool RT-TESTER [5]. In this section, we describe the general testing approach based on formal specifications, and give a detailed description of our interface between the CSP testing system and the SDL target. Then, we report on some specifics regarding the formal specification of the RLC layer, and, in particular, on testing multiple parallel instances of the RLC.

3.1 Formal CSP Specifications and RT-TESTER

CSP (Communicating Sequential Processes) [2] is a specification language which allows to give a description of a system on a high level of abstraction. The structure of the requirements is reflected by particular operators such as sequential or parallel composition, choice, iteration and hiding. Communication between the processes and with the outside is by the exchange of events. In contrast to SDL, however, this communication is synchronous (handshake); buffered communication has to be modelled explicitly. We use a timed version of CSP, where it is possible to set timers which generate events upon elapse. This way, it is possible to test real-time behaviour of applications, which is especially important for embedded systems.

Example 1 on page 8 introduces to a few basic operators of CSP, in particular to the event prefix, the external choice, and one of the parallel composition operators.

From formal CSP test specifications, test cases can be generated and executed on-the-fly. Our tool RT-TESTER reads the CSP input and generates an internal repre-

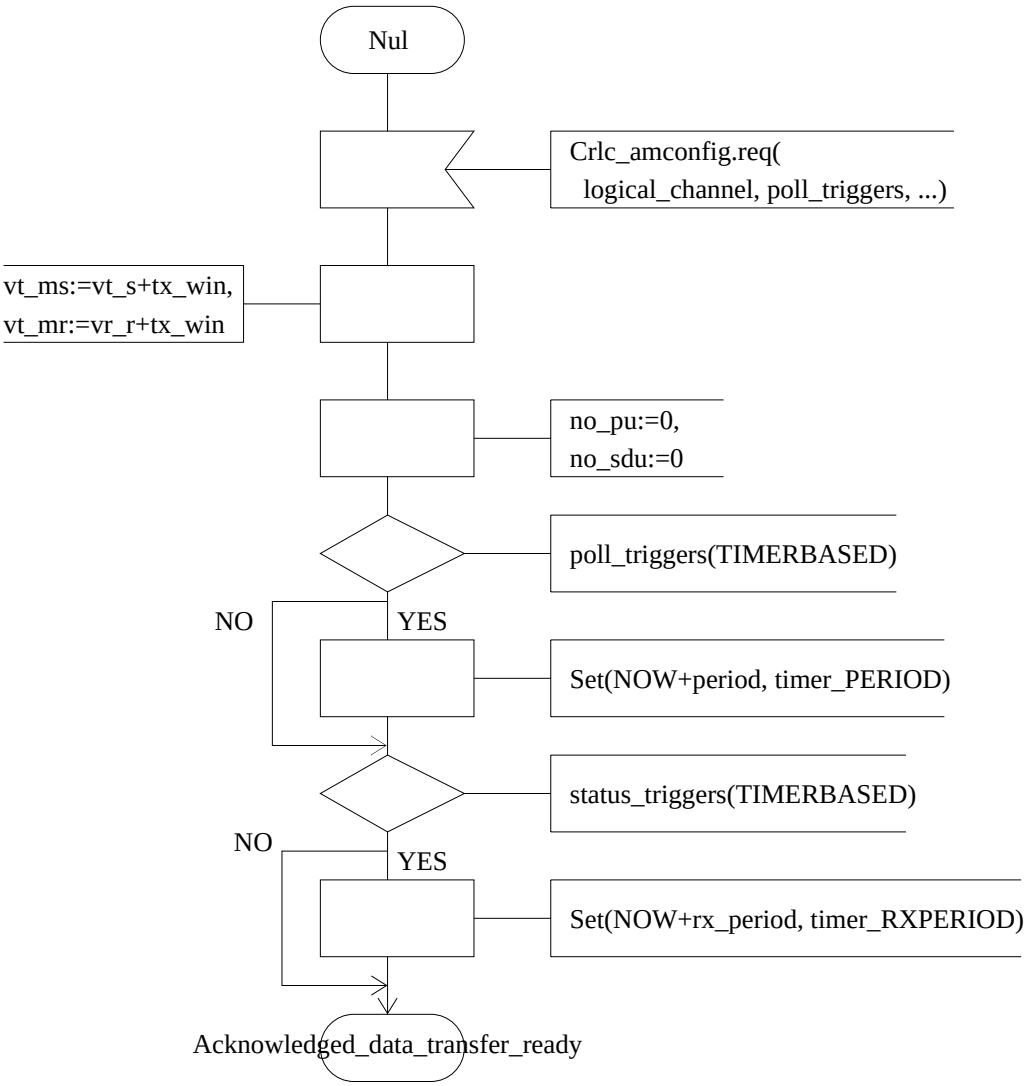


Fig. 4. SDL diagram for the initialization of a connection in acknowledged mode (bold arrow in Figure 3).

sentation from it, which is a huge graph of all allowed state transitions of the target. In a separate stage, this transition graph is used by RT-TESTER to generate test scripts, which are executed on a separate testing machine automatically and in real time. They may run over long periods of time: hours, days, weeks and more – without the necessity of manually writing test scripts of an according length. The testing machine and the SUT communicate via TCP/IP sockets, and test results are evaluated on the fly in real time during the run of the SUT, by using the compiled CSP description as a test oracle. To ensure that the tests cover the whole bandwidth of all possible system situations, a mathematically proven testing strategy is used. It guarantees that the testing coverage increases steadily, approaching a full verification of the specified requirements. Functional properties such as correct transmission and packet routing can be tested together with real-time properties such as correct time slots and sufficient performance within the same test run.

Example 1 A vending machine specification featuring a few basic CSP operators.

```
include "timers.csp"
pragma AM_INPUT
channel coin, buttonCoffee, buttonTea
nametype MonEv = { coin, buttonCoffee, buttonTea }
pragma AM_OUTPUT
channel coffee, tea
nametype CtrlEv = { coffee, tea }

OBSERVER = (  (coin -> HAVE_COIN)
           [] (buttonCoffee -> OBSERVER)
           [] (buttonTea -> OBSERVER))
HAVE_COIN = (  (coin -> HAVE_COIN))
            [] (buttonTea -> AWAIT({tea}) ; OBSERVER)
            [] (buttonCoffee -> AWAIT({coffee}); OBSERVER)

RANDOM_STIMULI = (|~| x: MonEv @ x -> PAUSE; RANDOM_STIMULI)

TEST_SPEC = RANDOM_STIMULI [| MonEv |] OBSERVER
```

A system described by the process `OBSERVER` accepts either of the three inputs listed (external choice "`[]`"). If the input is a `coin`, then the system behaves like the process `HAVE_COIN` (event prefix "`->`").

A system described by the process `HAVE_COIN` outputs the desired drink after the corresponding button press. In this, the sub-process `AWAIT` waits for any of the outputs specified (sequential process composition "`;`"). The definition of this process is listed in Example 3 on page 15 below.

The process `RANDOM_STIMULI` non-deterministically selects one event from the set `MonEv` (replicated internal choice "`|~| x : S @ P(x)`"), waits a short time, and starts all over (recursion).

The process `TEST_SPEC` describes the complete test suite: the process `RANDOM_STIMULI` provides all the test inputs, which are also tracked by the process `OBSERVER`. The latter additionally tracks the test outputs. They are combined by sharing ("`P [| S |] Q`") the input events in the set `MonEv`.

An example of a possible testing configuration is shown in Figure 5 on the facing page. In this setup, the testing machine is connected to three different development stages of the system under test: with an SDL runtime system, with a real-time operating system simulator and with the embedded target. During our actual testing, similar setups were used.

3.2 Interfacing SDL and CSP

A particular problem for the development of test suites for UMTS is that the standard describing the requirements still is subject to considerable change. Even after the "December 1999" release, which was supposed to be stable, dozens of changes were made, and more have to be expected. This concerns, for example, the parameters of the service primitives and the details of the data structures, such as protocol data units. Even the behaviour of the protocol machines is still expected to change, in particular for error handling. Similarly, not all implementation decisions have been finalized, some details of the machine representation of data at the interfaces are not yet fixed. We therefore designed the testing environment to be highly flexible by

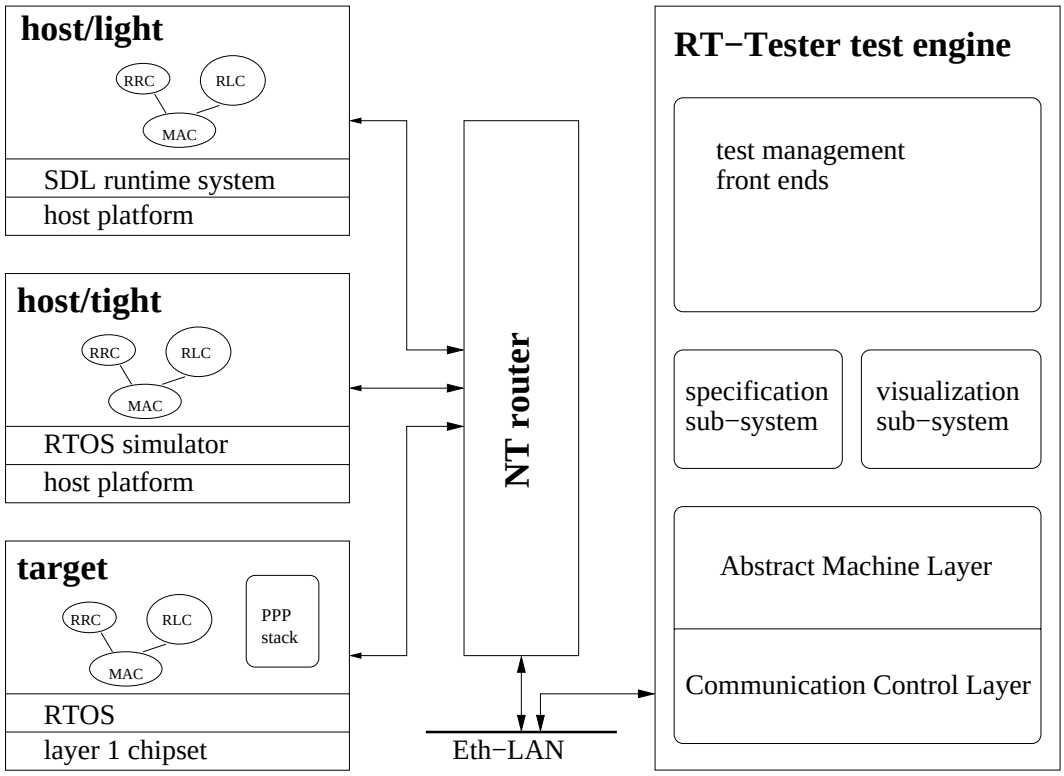


Fig. 5. Configuration for testing during development.

- defining the interface in terms of SDL signals and data structures instead of low level descriptions,
- performing an automated consistency check between the SDL description of the interface and the formal CSP specification of the interface, and
- modularizing the formal behaviour specifications into largely independent functional requirements.

The interface of the RLC layer relevant for testing is specified in SDL. Our automated approach makes it the only relevant interface. The SDL compiler generates a C language representation of the interface, and the C compiler generates a machine language representation of the interface, but both other representations are of no concern to our tests. The goal of the tests is to ensure that we can combine the tested SDL processes into a larger SDL system and achieve the desired behaviour. The actual representation of the internal interfaces between the components is not part of the visible behaviour of the combined system.

These internal interfaces have to be consistent only. On one hand, the SDL and C compilers do this type check. On the other hand, RT-TESTER can check whether the intended inter-process cooperation indeed occurs by performing (black box) integration tests of several components.

The interface in terms of SDL signals needs to be mapped to an interface in terms of RT-TESTER's native CSP channels. We need a translation between the two forms of

syntax. Due to its changing nature, we decided to automate the mapping by a generator tool. This generator tool also flags any inconsistencies between the interfaces of the two specifications. If the SDL specification is changed, the generator tool simply needs to be re-run. If it flags no error, the interface descriptions of the SDL specification and of the CSP specification are guaranteed to be consistent, and the next test run can check whether the behaviour is consistent, too. If the SDL interface has been changed in a part that is relevant to the CSP tests, the mismatched items are logged and we can directly investigate the problem. With the RLC protocol layer this feature is especially important: this module uses large and complex data structures as signal parameters, which are difficult to keep in sync manually. There are signals with more than one kilobyte of heavily structured parameter data; comparing their definitions manually would be extremely tedious and error-prone.

Figure 6 on the next page presents the concept of the automated interfacing. At the top there is the informal UMTS standard. Both the implementation team and the testing team interpret it and produce a formal specification in SDL and CSP, respectively. The names of the SDL signals and of the CSP channels must be the same in both specifications and are mapped automatically.

Channels and signals also can have parameters, which have to be mapped, too. In the simplest case, the first channel parameter is mapped onto the first signal parameter, and so on. In practice, this most simple scheme is rarely used. The mapping of the parameters usually requires user interaction, since CSP has a different concept of data structures than SDL, and since the UMTS standard provides data descriptions with huge data spaces, for example for the protocol data units (PDUs). Therefore, the CSP specification makes sensible abstractions to the state space, and the testing environment instantiates them with concrete values. Examples for sensible abstractions are blocks of, e.g., 512 bytes of data that will be transmitted transparently. Testing only a few representatives is sufficient.

We thus annotate each parameter of each CSP channel with the corresponding parameter of the corresponding SDL signal, as demonstrated in Example 2 on page 12. Particularly interesting is the "SKIP" annotation when we have data records as parameters that contain further sub-records. This is often the case for the protocol data units of the RLC layer. There are SDL signals with hundreds of sub-record components, of which most are entirely irrelevant to the protocol behaviour and thus to the testing purposes. An entire tree of such components is skipped implicitly by naming the top-most component only.

After the CSP interface has been annotated, "make" files and the generator tool can be invoked for the following steps (compare Figure 6 on the next page, again):

- The CSP specification is compiled as usual for the RT-TESTER runtime system.
- The interface adapter in the RT-TESTER runtime system also needs a (compiled) description of how the events on the CSP channels are mapped to byte strings. The generator tool produces this description automatically from the CSP specification.
- The SDL specification is compiled into a C language program, including C language header files that define C language names for the signals, their parameters, and

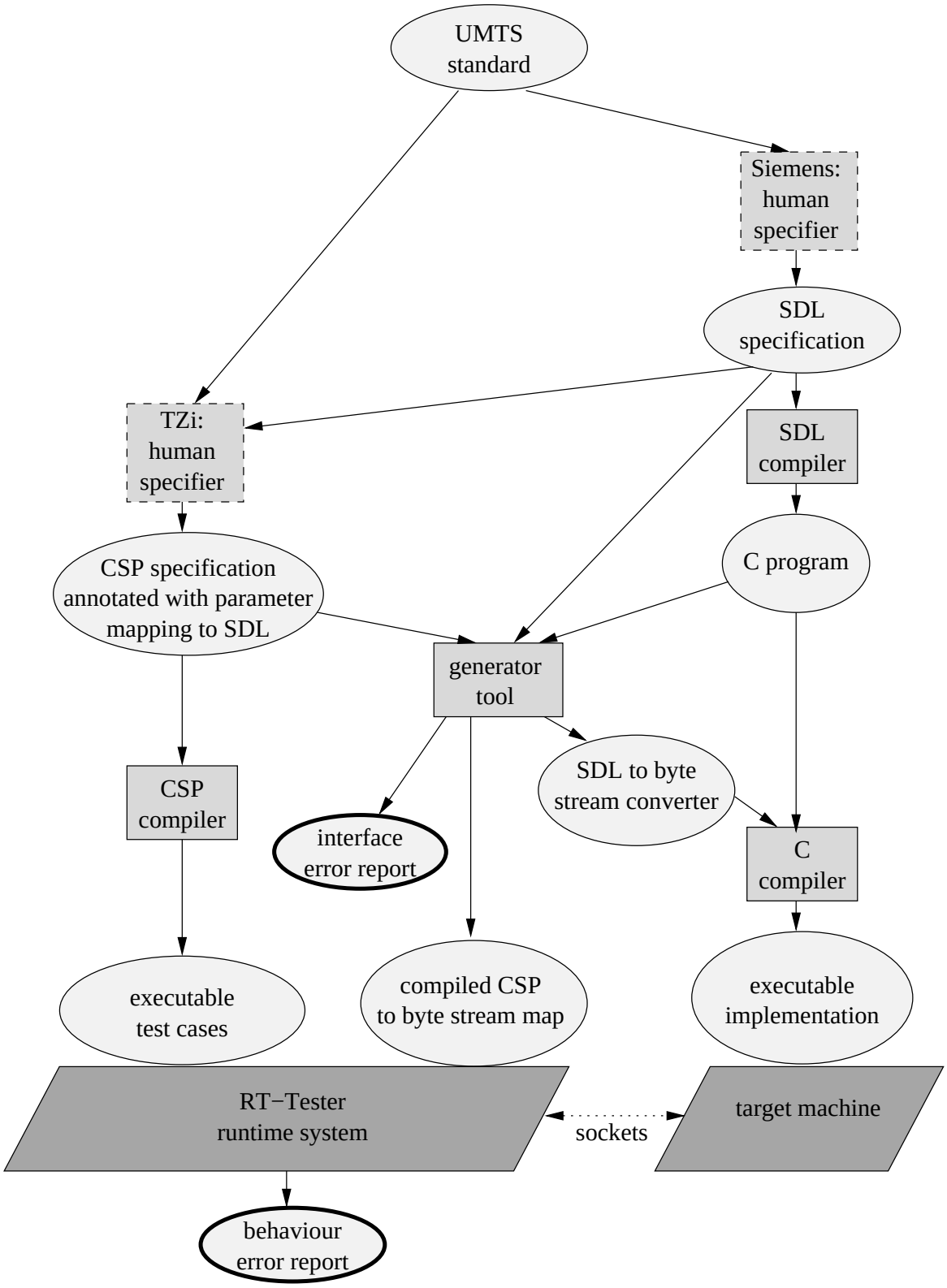


Fig. 6. Automated interfacing of SDL signals with CSP channels.

Example 2 An annotated CSP channel matching an SDL signal.

CSP

```
nametype Rb_identity = {0 .. maxRb_count}
datatype Rlc_data = dummy_value
channel rlc_tr_data_req :
pragma SDL_MATCH PARAM 1!RB_Id
      Rb_identity .
pragma SDL_MATCH TRANSLATE dummy_value 0x99 * 16
pragma SDL_MATCH PARAM 1!RLC_SDU_Data SUBSET_USED
      Rlc_data
pragma SDL_MATCH SKIP 1!length DEFAULT_VALUE 16
```

SDL

```
syntype RB_Identity = integer
      constants 0:MaxRb_Count
endsyntype;
synonym RLC_MAX_SDU_SIZE integer = 512;
newtype RLC_SDU_A
      carray(RLC_MAX_SDU_SIZE, octet)
endnewtype;
newtype RLC_SDU struct
      RB_Id RB_Identity;
      RLC_SDU_Data RLC_SDU_A;
      length integer;
endnewtype;
signal RLC_TR_DATA_Req(RLC_SDU);
```

The example defines a CSP channel named rlc_tr_data_req which has two parameters. The corresponding SDL signal RLC_TR_DATA_Req has one parameter which is a record of three components. With respect to mapping, each record component is treated like a separate parameter, in the order in which it is defined.

The first "pragma SDL_MATCH" line maps the first CSP channel parameter to the first SDL record component of the first (and only) SDL parameter, and it requires that the component is named "RB_Id". The CSP channel parameter is of type "Rb_identity" (a set of numbers serving as radio bearer ids). The SDL record component must have a type with the same set of values as "Rb_identity", otherwise the generator tool will flag an error.

The second CSP parameter is matched with the second SDL record component named "RLC_SDU_Data". This is actually an array of up to 512 octets (which are transmitted transparently). We do not want to test all possible values for this parameter, therefore we define the corresponding CSP type to comprise only one single value, called "dummy_value". In order to suppress the type mismatch warning message, we append "SUBSET_USED" to the pragma line. Furthermore, we want to use a well-recognizable pattern of data in the array. The "TRANSLATE" pragma line therefore specifies that the CSP dummy value should be sent as an array of sixteen hexadecimal bytes 99 to the SDL system under test. Similarly, only this pattern is recognized as a valid parameter value when received from the SDL system, all other patterns will result in a runtime error message in the test log.

Finally, the last "SKIP" line indicates that the third SDL record component "length" has no counterpart in CSP, but that it always should be filled with the value 16 (indicating the array length used).

the data types of the parameters, and a C language template file is generated that inputs and outputs SDL signals from and to the environment of the SDL system.

- This template is filled by the generator tool. The inserted code is part of the test interface adapter. The tool
  - takes the SDL specification and analyzes the signal definitions and the data type definitions in it,
  - matches it with the annotated CSP channel parameters,
  - takes the generated C language header files and determines the corresponding C language data types and parameter names, and
  - takes the C language template file and fills it. The encoding of the parameters into byte strings is the same as generated for the CSP side.
- The filled-out template and the other C files are then compiled into an executable implementation for the target machine.

The representation of the signals as byte strings is determined by the generator tool for both the CSP and the SDL side. The representation is independent of any machine architecture (as long as it provides strings of bytes). Each parameter is mapped to one byte, in the order in which they appear in the parameter list. If a parameter can take more values than a single byte can hold, it is mapped to more bytes. The byte order is defined by the tool to be least-significant-byte first.

The machine representations of data structures may be different on the target machine and on the machine running RT-TESTER. These machine representations may furthermore change during the project, e.g., due to changed choices of the hardware platform or of the embedded operating system.

Our automated approach for mapping signals obsoletes the need for a manual description of the machine representations, and it does not demand any user interaction after a change. Since we confine the target machine representation of the data entirely to the target machine, it is sufficient to rebuild the generated files and recompile them. The socket communication with RT-TESTER uses our own byte string format which is independent of any machine representation.

3.3 Formal CSP Specification of the RLC Protocol

We not only have to cope with changing requirements, but also with different testing scenarios. We accommodate for both by a modular structure of the formal behaviour specification.

In one scenario, we want to test an individual layer of the UMTS protocol stack. In another scenario we want to test the integration of several layers. Furthermore, some of the tests will be driven by external stimuli from the real world which are not under the control of the testing system. We therefore have to specify the test stimulus generator and the test observer as separate modules, to separate the specifications of the different layers, and to compose these modules in different ways.

Furthermore, some crucial aspects of the SUT have to be tested more thoroughly than others. We therefore encapsulate these aspects into sub-modules, compose suitably tailored instances to complex test suites as needed, and run them using the RT-TESTER tool. In all these cases, the actual test scripts are generated automatically on the fly.

The vending machine toy example in Example 1 on page 8 above composes the components already in the same way as the CSP specifications of the RLC layer (Figure 7): a test stimulus generator, written in CSP, generates an input to the implementation under test and waits some defined amount of time, then it loops and generates the next input. Concurrently, a test observer process, also written in CSP, observes both the input stimuli and the output reactions of the system under test. If the behaviour of the system under test is incorrect, an error is flagged. Example 3 on page 15 shows the definition of the CSP process `AWAIT` from Example 1 which assures that one of the specified set of legal outputs occurs in due time.

We can see how the definition of the process `AWAIT` is separated into a different file, and how it is integrated into the toy example by an `include` statement. We use a similar process in our test suite specifications in the same way.

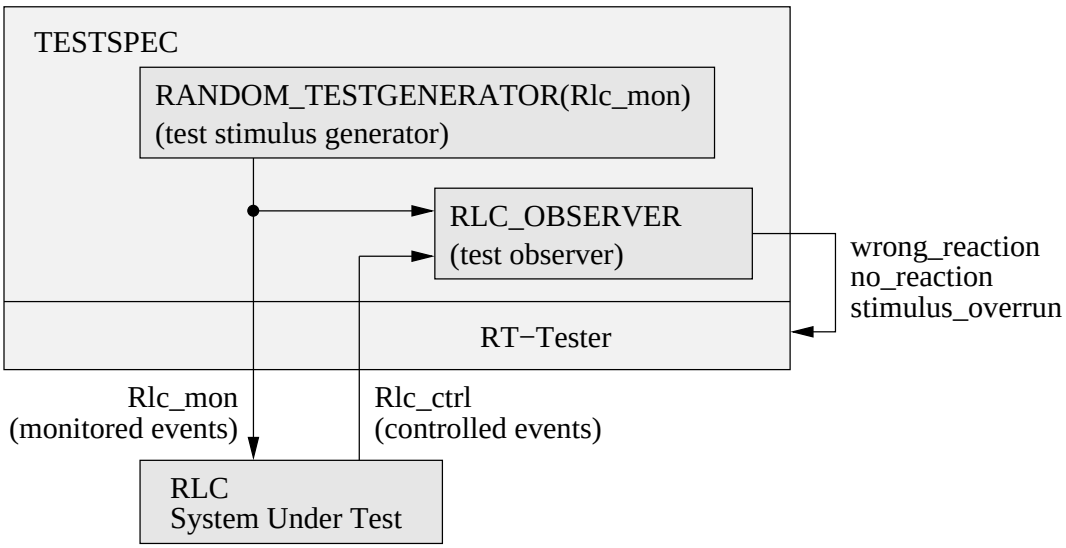


Fig. 7. The split between the stimulus generator and the test observer.

We separated the definitions of the different layers of the UMTS protocol stack into separate files. Beyond this, we also separated the data type definitions, the channel definitions, and the actual behaviour definitions. For several of these definitions, we designed different variants that can be composed in various ways.

For example, a test stimulus generator similar to the one in Example 1 generates test stimuli completely randomly. Another variant performs random choices with different probabilities, such that "interesting" states are reached more often, and yet another variant performs a fixed, explicitly specified trace of events. The latter has no internal choice operators at all, it just consists of a long sequence of event prefix operators ("`->`").

Example 4 on page 16 presents a CSP text fragment from our RLC layer specification. It describes part of the initialization of a connection in acknowledged mode (bold arrow in Figure 3 on page 6, SDL code in Figure 4 on page 7).

3.4 Testing Multiple RLC Instances in Parallel

The RLC layer comprises a number of protocol machine instances that run independently. This is true at least at the black box behaviour level, even though the implementation in SDL has a different, less decoupled structure. Therefore we specify and perform the testing of the entire RLC layer by replicating the CSP process for a single protocol instance, and by parameterizing it with the instance number. These processes run concurrently in parallel, each generating test stimuli for one of the RLC instances, and checking the reactions.

This testing setup is an example of a testing setup with several components tested at once. It is also possible to combine different kinds of components, for example RLC and MAC instances. Such a testing setup can be built by combining the CSP processes for the test observers into one test specification, and by selecting a suitable test generator.

Example 3 The timer-related CSP processes for Example 1 on page 8.

```
pragma AM_SET_TIMER
channel setTimer : { 0, 1 }
pragma AM_ELAPSED_TIMER
channel elapsedTimer : { 0, 1 }
pragma AM_ERROR
channel wrong_reaction, stimulus_overrun
pragma AM_WARNING
channel no_reaction

PAUSE = setTimer!0 -> elapsedTimer.0 -> SKIP

AWAIT(ExpectedEvSet) =
(   -- start timer and wait for things to come:
    setTimer!1 ->
    (   -- Accept correct reaction:
        ([] x: ExpectedEvSet @ x -> SKIP)
     [] -- Flag wrong reaction:
        ([] x: diff(CtrlEv, ExpectedEvSet) @
            x -> wrong_reaction -> SKIP)
     [] -- Flag overrun by next monitored event:
        ([] x: MonEv @ x -> stimulus_overrun -> SKIP)
     [] -- Flag no reaction (timeout):
        elapsedTimer.1 -> no_reaction -> SKIP))
```

The process `PAUSE` sets a timer and waits until it elapses. Then it terminates and thereby returns to its calling process.

The process `AWAIT` assures that one of the specified set of legal outputs occurs in due time. if not, it performs one of the events `wrong_reaction` `no_reaction`, ... which go into the test log. This process also terminates and thereby returns to its calling process.

4 Testing Results

Our tests yielded two kinds of results: the one kind, discussed in Section 4.2 below, are deviations in the behaviour of the SUT. The other kind was due to the specification based testing approach. In writing the formal requirements specification and comparing its interface to the implementation's interface, we found several ambiguities in the UMTS standards document.

4.1 Ambiguities in the Standards Document

The ambiguities which we found can be subject to different, equally legal, incompatible interpretations. These places will need special care to avoid inter-operability problems with software developed at other sites or by different manufacturers. (Compare Figure 8 on page 17.)

For example, the RLC layer must accept several kinds of data as PDUs from the underlying MAC layer and forward it through the appropriate service access points (SAPs) of itself to its upper layers. Similarly, the RLC layer must forward data arriving through its SAPs as service primitives down to the appropriate "logical channels"

Example 4 CSP specification of the initialization of an RLC connection in acknowledged mode.

```
-- The null state of the RLC (AM) entity:
RLC_AM_NULL(instance_id) = instate_rlc_am_null.instance_id ->
(   -- Wait for crlc_config_req setup request and honour it:
    crlc_config_req.rbSetup.1.instance_id?dummy ->
        RLC_CONFIG_AM(instance_id,rlc_to_rrc)
 []
    -- No data transfer is yet possible, since the instance does not
    -- yet exist, thus no reaction is expected otherwise:
    -- (A release request is discarded in this state, too.)
    ([] x : diff(Rlc_mon,
            {| crlc_config_req.rbSetup.1.instance_id |}) @
        x -> RLC_AM_NULL(instance_id))
 []
    -- any spontaneous reaction produces a warning:
    ([] x : Rlc_ctrl @ x -> warn_spontaneous_event -> RLC_AM_NULL(instance_id))
)

-- The other states are omitted here:
RLC_CONFIG_AM(instance_id,rlc_dest) = ...

-- The main process:
RLC_AM_OBSERVER(instance_id) = RLC_AM_NULL(instance_id)
```

The observer for the RLC instance with the number `instance_id` starts in the state `RLC_AM_NULL` and waits for a configuration request event. If the event occurs, the instance goes to the next state. All other events to the SUT are ignored. Any spontaneous output from the SUT would be an error and is flagged.

of the underlying MAC layer. In both cases, the appropriate destination cannot be determined from a service primitive or PDU and their parameters alone, as given in the standard. Therefore the RLC SAP was split into two SAPs, distinguishing whether an upward bound signal should go to the RRC layer or to a different upper layer, and another parameter was added to most service primitives and PDUs which identifies the destination inside one layer. These necessary extensions have the consequence that the RLC layer can be used only with MAC and upper layers which add the same parameter and which use the same SAP split.

Another much less obvious, but potentially even more serious example is that we did not find any precise definition of the properties of a service access point (SAP), or of the MAC layer's logical channel. In particular, there is no definition of

- whether signals are forwarded instantaneously or whether they are buffered,
- a queueing discipline (FIFO, ...) in the case of buffering,
- queue delivery dependences between different SAPs (single queue/multiple queues),
- possible minimum/maximum delays between delivery and availability,
- the possibility of data loss or alteration,
- the handling of signals that cannot be received.

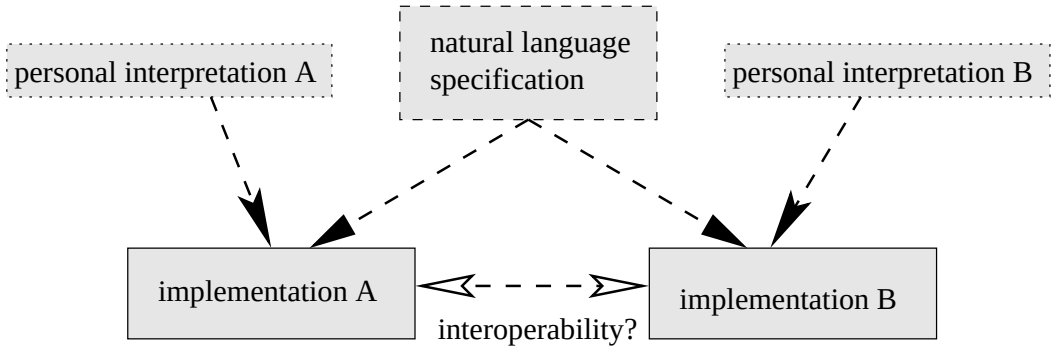


Fig. 8. Potential interoperability problems through an incomplete specification.

We can probably assume safely that neither SAPs nor logical channels lose or alter signals. But the other five issues can have an impact on the behaviour of the system. In particular, delays in the delivery of signals can result in *race situations*, or prevent them.

The implementors had to make decisions on the above issues. Several of them were made more or less implicitly by choosing SDL as the implementation language, since the endpoint of an SDL signal route or channel has a precise, specific semantics. Further decisions were made by choosing a particular structure of SDL processes inside a layer (number of independent queues per layer, ...).

Besides these ambiguities in the standard, we also found a few deviations in the implementation's interface definition from the interface defined in the standard.

4.2 Testing of the SUT's Behaviour

There were several situations in which the SUT did not behave as expected. For example, in a certain state the SUT reacts to a certain signal where no reaction was intended: an RLC protocol machine is created by the event `crlc_config_req.rbSetup` from the upper layers, and it becomes operational after receiving the event `mac_status_ind` from the underlying MAC layer. Immediately after that, the random test stimulus generator sometimes generated another (nonsensical) `mac_status_ind` event, to which the RLC layer sometimes, but not always, reacted strangely by generating a data packet, i.e., with a `mac_data_req` event. Since no data transmission requests had been issued yet, and thus no buffered data could possibly be pending, this is unexpected. With structural testing, probably no explicit test script would have been written that checks for a non-reaction in this state. The systematic random exploration of the state space in our approach found this problem automatically.

Furthermore, the tests revealed interactions between different instances of the RLC protocol machines. The requirements allow different instances of these machines which behave completely independent. Each instance could be tested separately. But we also performed a test where several protocol machines were tested at the same time, each one against its own copy of the requirements specification. It turned out that in such a setup there was the possibility that the entire SUT could deadlock. The reason for

this effect was that the the different instances of the RLC protocol machines are not implemented entirely as separate copies. Rather, there is a centralized routing SDL process which forwards data transmission requests to a set of SDL processes which implement one RLC instance each. This routing process did not handle the following case properly: if a signal arrived addressed to an RLC instance which does not currently exist, the process could loop infinitely during the instance look-up. It then ceased to perform its routing job. Even though the implementation was built with sophisticated error recovery mechanisms, this situation could not possibly have been foreseen in the development.

Another interesting observation showed up only intermittently, after a certain history of input: even though the SUT always *should* have gone into the same state, it did not. Besides the random test stimulus generation, we also used a generator which performs a fixed, explicitly specified trace of events, and which starts all over from the beginning when the listed trace is through. A correct SUT should return to the initial state at the end of the listed trace, and then it should go through the same states again, delivering the same reactions. In fact, it turned out that in the second, fourth, etc. rounds some data transmission requests `rlc_tr_data_req` from the upper layers towards the MAC were lost, which were transmitted correctly by the RLC layer in the first, third, etc. rounds. The morale is that a protocol machine instance implementation does not necessarily return to its initial state just because the instance is "freed". Implementation optimizations can retain parts of the old state, and this can lead to subtle misbehaviour. The on-the-fly generation of the test scripts allowed us to run the test suite for a long period of time, which was necessary to detect the above problem.

5 Summary

In this paper, we described a testing setup for the UMTS protocol stack. Specific features of this setup are

- test scripts are generated automatically from formal requirements specifications which are developed independently of the implementation,
- the interface between the system under test and the requirements specification is generated by an automated tool. The tool also performs consistency checks. This allows last-minute changes in the data formats.

The paper describes ongoing work. In particular, we did not yet run the test cases on the embedded target (prototype of UMTS user equipment). We expect to be able to re-use all test specifications directly since they refer to external interfaces only.

References

1. 3rd Generation Partnership Project. http://www.3gpp.org.
2. A. W. Roscoe. "The Theory and Practice of Concurrency". Prentice-Hall (1997).
3. Jan Ellsberger, Dieter Hogrefe, and Amardeo Sarma. "SDL – Formal Object-oriented Language for Communicating Systems". Prentice-Hall (1997).
4. 3rd Generation Partnership Project, 3GPP TS 25.322 V4.0.0. "RLC protocol specification" (March 2001).
5. Verified Systems International GmbH. http://www.verified.de.

QW2001 Paper 7A2

Mr. Michael K. Jones, Assistant Professor
(Dromedary Peak Consulting/Western International University)

High Availability Testing

Key Points

- High availability of computer systems is a necessity for more systems than ever.
- Testing of high availability systems must be defined in terms of software, system, and process.
- Test strategy derivation for high availability systems must be traceable to the logical system architecture and the physical system architecture.

Presentation Abstract

High availability of computer systems, especially internet computer systems, is a necessity in today's world when the absence of a computer system and/or web site will not only affect the immediate revenue stream, but will affect revenue to come by decreasing the likelihood of future use of the system by those users who encountered the absence of the system. Merely installing failover software and hardware will not ensure that high availability will be maintained under load conditions. High availability testing must be conducted wit h analysis and a structured approach to provide an accountable confirmation of reliable high availability functionality. This testing will return to the system owner not only a metric of availability versus conditions for functionality, but a means to calc ulate cost versus risk reduction for system reliability.

The paper will define availability and "high" availability. A testing typology will be discussed in detail that explores availability in terms of software, system (software and hardware), and process (software, hardware, and procedures). A testing taxonomy will then be reviewed in terms of induced failure analysis by means of increased load, singular system component termination, and multiple system component failure.

Test strategy derivation will be discussed next showing traceability to the logical system architecture and the physical system architecture, with special attention to how to discern points of criticality for system availability. Test documentation will then be reviewed with emphasis on particular points that must be documented in an availability context. Finally regression testing of high availability will be evaluated as to its value for maintenance of high availability after the initial deployment of the system in both a periodic sense and in system updates.

About the Author

Mr. Jones has a Master of Science in Computer Information Science and a Master of Business Administration. He has been through software engineering training at Boeing, Texas Instruments, and McDonnell Douglas in the past. He has worked in the software industry since 1976 and is currently the Chief Consultant at Dromedary Peak Consulting, which provides analytical direction and operational support for business. He is also an Assistant Professor in Information Technology at Western International University. His courses include Advanced Software Engineering, Advanced C Programming, Information Resource Management, Internet Business Strategy, and Web Application Development. Some of his published articles include: “Pragmatic Software Configuration Management in the E-World”, “Pragmatic Software Testing in the E-World”, “Software Configuration Management for the Web”, “Report from Captain QA from the Web”, and “Four Conceptual Attributes for Successful Web Applications”.

HIGH AVAILABILITY TESTING

Michael (Mike) K. Jones
Chief Consultant
Dromedary Peak Consulting
And
Assistant Professor
Western International University
Phoenix, Arizona USA

1

Availability is a system attribute that delineates the strength of the delivered system to be present and readily at hand for work.

High availability is the term used to describe systems that maintain a capability for use to a high degree.

This degree is usually expressed in terms of “nines”.

2

The design principles of high availability are centered on the concepts of:

- Reliability
- Failover
- Redundancy
- Replication

3

Critical path is the line through which data must travel from sender to receiver, or from one user to another.

The goal of high availability is to have no single point of failure on this line of transmission.

4

Reliability is the capability of software or hardware to consistently perform per requirements without failures or anomalies occurring.

Every component utilized in a high availability system must have demonstrated proof of reliability.

5

Failover can be manual or automatic. Automatic works and manual may work.

There are 2 types of automatic failover:

- Dedicated Backup
- Partnered

6

Redundancy is inherent in failover and replication

More accurately focused on need for redundant data channels and power conveyances.

7

Replication emphasizes the availability of data in multiple locations

Replication categories are:

- Copying of data at the file level
- Copying of data at the disk level
- Distribution of data at the transaction level
- Duplication of data at the server level

8

Availability typology:

- Software
- System (software and hardware)
- Process (software, hardware and procedures)

9

For software – reliability

For system – redundancy and failover

For process - replication

10

Formal Testing Avenues:

- Increasing load tests
- Deliberate termination of a singular component
- Deliberate termination of multiple components

11

Increasing load testing is stress testing

Drive to breaking point is based on:

- Number of users
- Number of messages
- Size of messages
- Required processing for each message

12

Deliberate termination of a singular component must address every critical path item individually in a systematic effort.

Deliberate operational termination of multiple components must be based on test strategy due to number component permutations.

13

Test strategy derivation is based on analysis of logical architecture traced to physical architecture.

Points of criticality can be identified as a result of this work.

14

Test planning is based on software, system, and process and

Stress testing, single item failure, and multiple item failure.

For each avenue, a test procedure should be produced.

15

Availability testing is best performed on a production system.

If performed in a scaled down test lab, scaling can only be conducted down to where at least one level of failover can occur.

16

Test Documentation:

- Test Plan (Resources and Schedule)
- Test Procedures (Based on avenues)
- Test Reports (Pay special attention to environment and timing metrics)

17

Regression testing should be for both new system baselines and periodic calendar points.

Regression testing should also be conducted for equipment replacement.

18

High availability visibility is growing.

High availability testing ensures high availability.

19

For feedback and/or further communication, please contact me at:

Michael K. Jones
Dromedary Peak Consulting
1451 East 8th St.
Mesa, AZ 85203 USA
Phone: 480-833-0927
Cell: 480-363-7527
Email: mjones@dromedarypeak.com
jonesaz@uswest.net

20

High Availability Testing
by
Michael (Mike) K. Jones
Chief Consultant
Dromedary Peak Consulting
Mesa, Arizona
and
Assistant Professor
Western International University
Phoenix, Arizona, USA

High availability of computer systems, especially internet computer systems, is a necessity in today's world when the absence of a computer system and/or web site will not only affect the immediate revenue stream, but will affect revenue to come by decreasing the likelihood of future use of the system by those users who encountered the absence of the system. Merely installing failover software and hardware will not ensure that high availability will be maintained under load conditions. High availability testing must be conducted with analysis and a structured approach to provide an accountable confirmation of reliable high availability functionality. This testing will return to the system owner not only a metric of availability versus conditions for functionality, but also a means to calculate cost versus risk reduction for system reliability.

Availability is a system attribute that delineates the strength of the delivered system to be present and readily at hand for work. Availability provides for uninterrupted access to the system with no loss of connectivity, processing capability, or stored data. Availability is designed in systems through comprehensive redundant no-single-point-of-failure hardware and software.

High availability is the term used to describe systems that maintain a capability of being ready for use to a high degree. This degree is based on a percentage of uptime with uptime being defined individually for every system as the system in use or ready for use by the operator. Or conversely, downtime can be defined as whenever a user cannot get their work done due to the system not being available for requisite actions. High availability is expressed in terms of the “nines”. 99.99% availability means that a system will be constantly ready for use 99.99% of the time for a year which translates to roughly only 52 minutes of downtime a year. As an added example, 99.999 availability translates to approximately 5 minutes of downtime a year.

The design principles of high availability are centered around the concepts of reliability, failover, redundancy, and replication. As these concepts are employed to an increasing extent in the architecture of a system, two other elements are always directly increased. Those two items are the technical complexity of the system and the cost not just of implementation, but maintenance as well. An important analysis that must be performed when discussing high availability is the recognition of what the critical path is in a system for data transmissions. The critical path is the line through which data must travel from sender to receiver, or from one user to another. The goal of high availability is to have no single point of failure on this line of transmission or critical path.

Reliability is the capability of software or hardware to consistently perform per requirements without failures or anomalies occurring. The use of reliability is the foundation of designing high availability system. Every component, whether hardware or software, must have demonstrated evidence of high reliability under a range of extreme conditions. Items that do not have these records are always rejected forthwith.

Failover can be executed manually, but the invariable result is inordinate downtime for the system. Consequently, failover must be implemented to occur automatically. Automatic failover can occur with two very different architecture models. In the first, failover is instrumented automatically with dedicated backup servers for all critical path servers. This mode is sometimes called asymmetric failover. In the second, failover occurs between partnered servers backing each other up. There is some degradation of performance when one server assumes the burden of its partner in addition to its own task load, but the concept is viable. This second mode may be known as symmetric failover.

Redundancy, although inherent in failover and replication design, is more accurately focused on the need for redundant data channels and power conveyances. These redundant data channels also are designed to handle addressing through careful construction of routing tables that provide for alternative paths. Power conveyances are straightforward in that alternate power sources are simply provided for all power consumers.

Replication includes redundancy as a matter of course, but has its emphasis on the availability of data in multiple locations. Replication can be implemented in a variety of ways, but they may be organized into the categories of:

- Copying of data at the file level
- Copying data at the disk level
- Distribution of data at the transaction level
- Duplication of data at the server level

Replication for high availability systems must provide no lags in time to access nor gaps in integrity for this data.

For availability test considerations, an availability typology can be structured in terms of layers of software, system, and process. Software is simply software, system is software couples with hardware, and process is software, hardware, and procedures utilized together. By using this typology in a progressive manner, testing isolation would be performed and confidence established, and then layered complexity increased for further testing.

For software, reliability is the overriding principle. Software, in one form or another, is responsible for over half of all system downtime. Software must be rigorously tested as to functionality and lead prior to its selection for inclusion in a high availability system. If software is developed in-house or evidence of validation is not available from suppliers of software whether contractors or vendors, this software must undergo very tight

testing trials to establish adequate confidence for its use in a high availability system.

For the system layer, redundancy and failover in addition to reliability must be implemented. As previously stated, the system layer consists of hardware and software jointed together. Reliability must have again have proven evidence established as a result of testing. The redundancy principle is utilized by the existence of fallback systems for every critical path item. These fallback systems are, furthermore, accessed through the institution of automatic failover mechanisms to one or more failure levels.

For the process layer, replication as a design principle comes into play. Not only must the reliability, redundancy, and failover principles be used as decision constraints, but also now the transmission of current electronic data must be secured in alternate data storage devices in parallel to the critical path data receptacles. This replication of data in redundant and failover capable components can be conducted through processes implemented by file system replication, data base replication, or service replication. It is important to note that not only the application data stream must be preserved in multiple stores, but the secondary and tertiary system data necessary for the application management and utilization must be gathered as well.

Utilizing the three layers, formal testing of high availability must be conducted by three avenues. These avenues are increasing load tests, deliberate operational termination of a singular component, and deliberate operational termination of multiple components. This taxonomy of testing allows for a logical structure for formulating documented test procedures to cover system test condition.

Increasing load testing, or stress testing as it is sometimes called, provides a close mimic of real world conditions in identifying vulnerable links in data processing. This progressive march to some component in the system being driven to the breaking point is based on increasing the number of users, the number of messages, the size of messages, and required processing for each message. The location of the component driven to failure may vary based on how the stress algorithm is structured. When a component is driven to failure, it is important, through system monitoring time, to record the time used to transfer operational presence to another component. The stress testing procedure must be documented in a formal test procedure with a deliverable test report for test results.

Deliberate operational termination of a singular component is reasonably straightforward. Every component, including equipment, data cables, and power lines that is necessary for critical path transmissions, must have deliberate failure introduced through scripted termination. This testing must be conducted systematically and address each critical path individually. As the failure of each item is brought about, the overall system must be monitored to observe and record failover time and system status.

Deliberate operational termination of multiple components is the most challenging avenue to address in testing for high availability. Due to the permutations of components found in highly complex system, it is doubtful that all permutations can be tested for a time or money efficient manner. A test strategy based on architecture

analysis will be most useful in approaching this task. Some tactics used on other systems have been to fail one of every type of component, to fail sequences of components, and to fail components based on random selection. Failing all components but the minimum, with variations on individual item selection, forces a boundary valve availability test.

Test strategy derivation is a very important task in planning for the validation of high availability for a system. By tracing the logical architecture through the physical system architecture, points of criticality for failure can be easily discerned. These points of criticality or possible single points of failure must be inspected with the design concepts of reliability, failover, redundancy, and replication in mind. Any one item or combination of items that are questionable as regards meeting these concepts must be ranked high for testing. By reviewing the logical architecture, responsibility for specific functions may be seen. After tracing the implementation of those functions in the actual physical system, vulnerabilities as to failure will be tangible when the critical path for data is traced through the system.

After the system has been reviewed and inspected with the design principles of high availability in focus, in particular the logical and physical architectures, test planning must be initiated. This test planning must schedule the resources for each type and quantity of formal high availability testing to be performed based on the three layers of the system: software, system, and process; and the three avenues: stress testing, singular item failure, and multiple item failure. For each avenue, a separate test procedure must be written based not only on the design analysis, but testing to the availability system attribute requirement found in the Technical Requirements for system. Another possible source that must be tested to, that should match the availability requirement in the Technical Requirements Document, is the availability requirement defined in the Service Level Agreement for the system, if one is in effect at time of delivery.

Availability testing is best performed against a production system, but may be tested if necessary in a configuration deployed in a Quality Assurance Test Lab. Scaling down of the system may have to be instrumented, but for valid availability testing, the system can only be scaled down to the point where at least one utilization of failover for all critical path items can be attempted. One tactic may be to utilize one component as a failover recipient for multiple functions either in a permanent configuration or in configurations that are reconstructed for each type of failover testing performed.

Test documentation must be maintained for all testing. In addition to the test plan and test procedures, test reports must be produced that communicate and describe the test results. Particular attention must be paid to the environment definition for the test being described as to the exact system configuration in place at that time. Time in as accurate a metric as can be recorded is another point for availability testing when failovers are introduced.

High availability testing is conducted as system attribute testing prior to deployment of a developed system, but once deployed, a system with high availability must continually undergo regression testing for high availability. This regression testing must be conducted both in a periodic calendar sense, but also when modifications to

system baselines are introduced. It is also important to run high availability testing whenever components are replaced in the system even when the part numbers are identical, due to undocumented differences in manufacturing lots that might possibly affect availability.
A dedication to regression testing for high availability will not only minimize the unpleasant experience of system failure and not failover, but also create a high probability that experience will not be undergone.

High availability visibility is definitely growing on the radar screens of senior management of companies relying on their information technology systems for revenue, which leaves out very few corporations today. By utilizing the testing layers and test avenues found in the paper, a structured approach to high availability testing can be initiated. High availability testing ensures high availability will be a continuous reality for systems today.

As a further remark, no resources were found that dealt with structured high availability testing. For high availability itself, the best reference found by this author is:

Marcus, Evan and Stern, Hal (2000). Blueprints for High Availability. New York: Wiley Computer Publishing.

QW2001 Paper 8A1

Mr. Vince Budrovich
(ParaSoft Corporation)

Increasing The Effectiveness Of Load Testing: Unit-Level Load Testing

Key Points

- Performing load testing early in the development cycle can significantly reduce development costs and prevent great frustration.
- Testing servlets as they are written is an effective way of uncovering scalability problems early on.
- Performing unit level load testing can uncover elusive errors that can get missed during application level testing.

Presentation Abstract

Load testing is often delayed until a Web application is near the final stages of development. This is a dangerous practice because the later a problem is found, the more difficult and costly it is to fix. For example, imagine that you just learned that one of your site's servlets-- a servlet that you developed months ago and that many other servlets depend upon-- cannot handle more than 10 users at once. Fixing the problem now can be anywhere from difficult to disastrous. This problem may signal a design or implementation problem (such as a poorly designed database or poorly designed objects). If so, fixing this problem could involve modifying your entire design or implementation strategy and redesigning and/or rewriting the entire application.

If you had tested this servlet immediately after it was written, you could have spotted this scalability problem immediately. As a result, you could have also spotted and reworked design and implementation flaws before they became widespread. Essentially, you could have prevented many problems that would be difficult and costly to fix.

That's why we believe you should start performing load testing at the unit level. "Performing load testing at the unit level" means that each time you create or modify a dynamic site component (such as a specific customization option, or a "wish list" functionality), you should immediately test how well it performs with different types and amounts of user traffic. By starting load testing at the beginning of the development process, you can detect and fix problems before they become widespread, at the stage where it is easiest to do so. Performing load testing from the beginning of the development cycle is one way to improve load testing's

effectiveness. You can make it even more effective by overcoming some of the challenges that prevent traditional styles of load testing from adequately exposing the scalability problems on a site:
* Determining all scalability-related program failures: Load testing should not just measure load statistics like load rate, bandwidth, etc., but also report where user traffic could cause problems such as bottlenecks, program failures, and functionality problems.
* Creating a large number of different, realistic paths: Real users do not follow fixed paths; there is always some degree of randomness inherent in their behavior. Moreover, different types, amounts, and combinations of user paths through a dynamic site can result in different problems. This means that in order to perform thorough load testing, you need to create and execute countless numbers and combinations of different, realistic user paths. Manually specifying an adequate number and variety of different paths is not only tedious, but practically impossible-- even if you have a tool that simplifies the script-writing process.
* Creating the pages to test: Before you can test a page, it must be on your sever. You need to compile the program, perform all necessary initializations, transfer the program to your site, click through the site to set the necessary state variables, then manually add inputs to invoke the page. This can be time-consuming and tedious-- especially when you do it the amount of times required to invoke and test different paths through the site.

After explaining these challenges, the speaker will discuss and demonstrate methods of overcoming them while performing unit-level load testing. Then he will explain how these same strategies can be applied to improve the effectiveness of application-level (traditional) load testing. By the end of the session, attendees will learn how load testing can be performed at the stage where it yields the best results as well as how to perform load testing in the most effective manner possible.

About the Author

As Instructional Systems Manager of ParaSoft, Vince Budrovich works closely with ParaSoft developers and customers to overlook and assure product quality and dependability. Budrovich has extensive experience working as a project manager, organizing and developing performance support materials for software products, and business systems analysis. Budrovich earned his Bachelor's Degrees in Economics and Political Science from the University of California Santa Barbara and his Master's Degree in Planning and Administration from San Francisco State University.

ParaSoft

Increasing the Effectiveness of Load Testing:
Unit Level Load Testing

ParaSoft

Presenter: Vince Budrovich

1

Fantasy Land or Tomorrow Land?

- Good News: Your Website development is done on-time and on budget
- Bad News: Your Website is very slow for most users and freezes at peak hours
- Very Bad News: You don't know what is wrong

ParaSoft

2

Fantasy Land or Tomorrow Land?

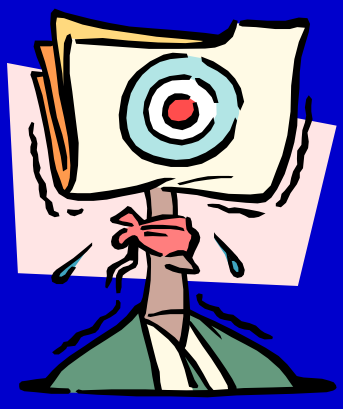
And More Bad News:

- You do not know precisely what to fix
- You do not know how much or if your application needs to be re-written
- You are going to miss the deadline and the budget target, but not know by how much.

ParaSoft

3

Fantasy Land or Tomorrow Land?



Did you perform

Unit Level

Load Testing

on every element and sub-system in your application?

ParaSoft

4

Unit Level Load Testing Overview

- Rationale: Why should you do it?
- Strategy for unit level load testing
- Web-box testing – Unit level load test for servlets & components
- Testing the database & back end
- Testing the business logic
- Testing Web servers & load balancing
- Testing bandwidth & pipes
- Customer experience: That last mile

ParaSoft

5

Rationale

Load testing - Is it a QA issue?

Yes

No

Don't know?

ParaSoft

6

Rationale

Load testing – Is it a QA issue?

- Load problems: the result of difficult design and algorithmic flaws
- Fixes: require significant re-writes… or much more
- The longer you wait, the more code you'll need to re-write

ParaSoft

7

Rationale

Load Testing – Is it a QA issue?

Example: Key servlet

- Many parts of the application work with it
- Servlet developed months ago
- Load problem: at 12+ users, it crashes
- Solutions:
  - Rewrite the servlet and more…
  - Rewrite all affected areas of the code, database interactions, certain EJBs, eliminate excess tags, etc.

ParaSoft

8

Elements to Load Test

- Each servlet and every other individual component of code
- Database and legacy systems
- Business logic interactions
- Website internally, Web servers' balance
- Connection to the Internet
- Your customer – That last mile

ParaSoft

9

Typical N-tier Website

Layers or sub-systems, from the bottom to the top:

(A) Back end: including databases & legacy systems

(B) Middleware: including servers with business logic

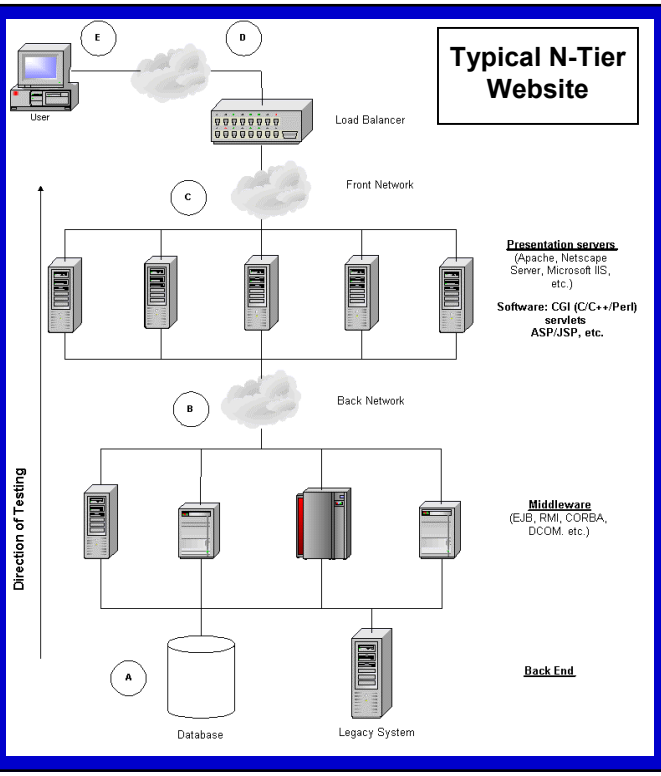
(C) Presentation Web servers

(D) Pipes connecting to Internet

(E) User or customer – That last mile

ParaSoft

10



ParaSoft

11

Strategy

- Test each element or layer
  - independently
  - immediately
- Keep testing – regression tests
- Priority – not optional
- Clean development process – emphasize problem prevention, not crisis management

ParaSoft

12

Strategy

- Expanded definition of unit level load testing
- Test each servlet or unit of code
- Test each element & sub-system
- See if each element functions adequately under load test – independent of other components

ParaSoft

13

Strategy

- Test the smallest building blocks using Web-box testing
  - Servlets or code components
- Test levels of your Website – testing each element or sub-system
  - Back end – database & legacy system
  - Business logic
  - Entire Website – balanced load among servers
  - Pipe to the Internet – bandwidth & balance
  - Customer experience
- Fix load test problems before going to the next level

ParaSoft

14

Strategy

Even if you do everything else…

- Do not forget “That last mile”
- Customer perception is key
  - Design – output page effects
  - Static analysis – enforce coding standards
- Do not delay load testing
  - The cost of a delayed fix can mean a lot of re-coding and extra work

ParaSoft

15

Web-box Testing: Servlets & Components

Why begin with the Servlet?

- It is the smallest “granularity” of unit level load testing
- Can be located on any Server in a Website
- Can impact an element if not functioning correctly
- Test it before testing the more complex elements and layers
- Fix a problem first before moving on

ParaSoft

16

Web-box Testing: Servlets & Components

- Set of techniques to deploy and test programs and related output pages
- Load test each servlet or component individually before applying elsewhere

ParaSoft

17

Web-box Testing: Steps

- Write the servlet
- Compile the servlet
- Transfer the executable servlet to the designated servers
- Invoke the servlet to activate it
- Give the servlet the correct arguments before it starts running, in order to look at the page
- Execute it and see its output page

ParaSoft

18

Web-box Testing: Steps

- Test if the page created by the servlet is correct
- Invoke it repeatedly to test the servlet under load, i.e.: to perform unit level load testing
- Use the test cases to “pound” on the servlet to emulate actual load conditions it will need to match
- For example, servlet page expected to fully display xxx times per minute

ParaSoft

19

Web-box Testing: Steps

If it does not perform as required…

- Fix algorithm in that servlet right away and test again
- Do it while it is fresh in your mind
- Test and fix before other servlets are created and tied to this servlet
- Do regression tests of entire test suite

ParaSoft

20

Web-box Testing: Benefits

- By testing every individual servlet, any load performance problem is quickly and simply solved
- Potential complications and problems are avoided

ParaSoft

21

Web-box Testing
Up to this point…

- Performed unit level load testing on every servlet
- Future load problems are not because of the individual servlets
- Proceed through the entire Website, layer by layer
  - Test every element & every sub-system
  - Load test each level of your Website application
  - Eliminate potential load problem by testing it
  - Proceed to the next layer

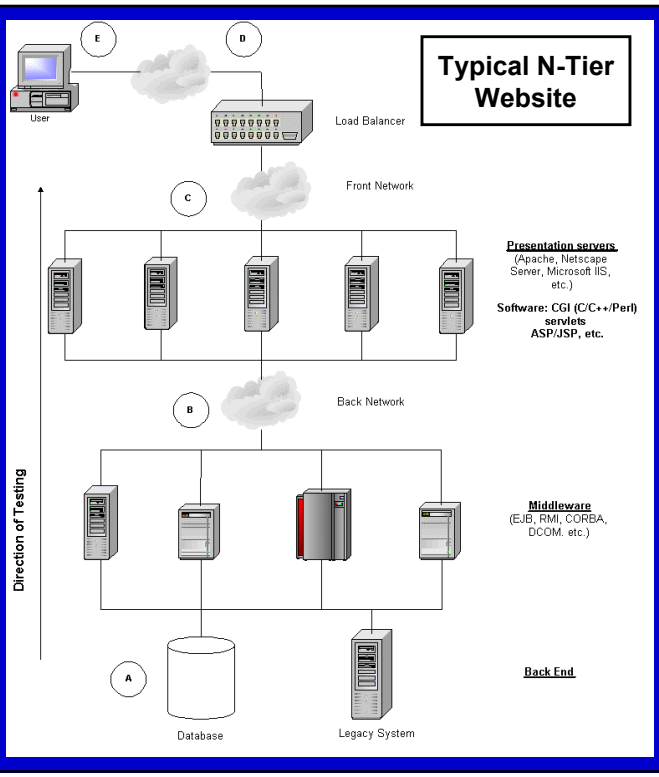
ParaSoft

22

N-Tier Website
(A) Back End

- Individual servlets - located on any server (at any level) in your Website
- Database & legacy system - located in back end of your Website

ParaSoft



ParaSoft

Back End

Database interactions:

- Emulate requests which go to the database
- “Pound” on the database to see if it can process all the transactions
- Measure responses of database

ParaSoft

25

Back End

Database Interactions:

- Create a set of “scripts” – SQL
- Scripts execute full transactions on the back end
- Repeat test thousands of times to emulate real conditions
- Measure the timing and see if it is adequate

ParaSoft

26

Back End

Legacy System Interactions:

- Test if legacy system performs adequately under load test
- Use new upcoming techniques, testing protocols - XML or EDI

ParaSoft

Back End
Up to this point…

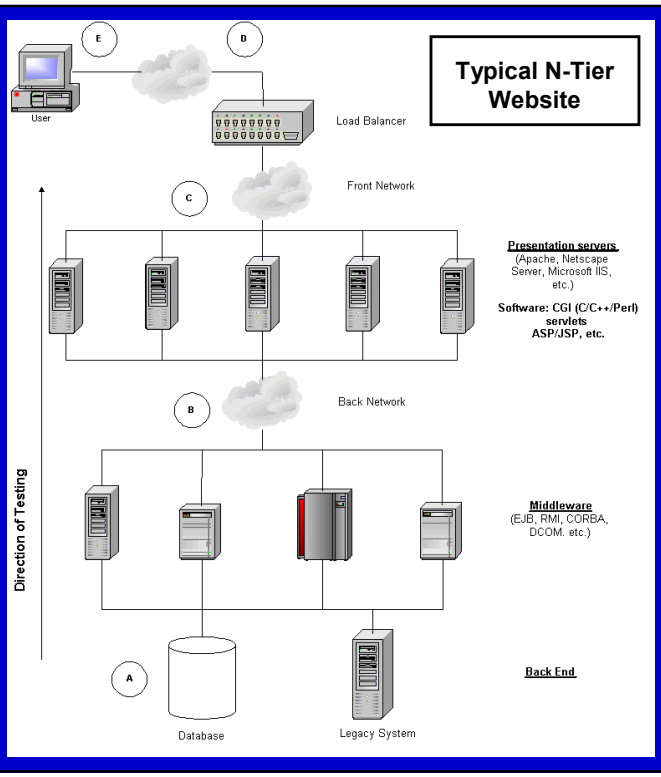
- Tested both elements in back end sub-system
  - Databases
  - Legacy systems
- Previously tested individual servlets and code components

ParaSoft

N-tier Website
(B) Business Logic

- Test the business logic - the interactions between servlets and the database or legacy system
- Located on middleware servers of your Website

ParaSoft



ParaSoft

Business Logic

- Typically written in Java or Perl
- Converts requests from customers into requests for the database or servers
- Performs logic and does calculations as specified
- How well does this software operate?
  - Is it fast enough for your Website?

ParaSoft

31

Business Logic

- Testing two layers
  - Database or legacy system
  - Business logic of individual servlets
- Problems are caused by the interactions between the two layers
- Test different business logic elements and interactions among them

ParaSoft

32

Business Logic
Up to this point…

Already Tested:

- Individual servlets
- Databases and legacy systems
- Business logic interactions

ParaSoft

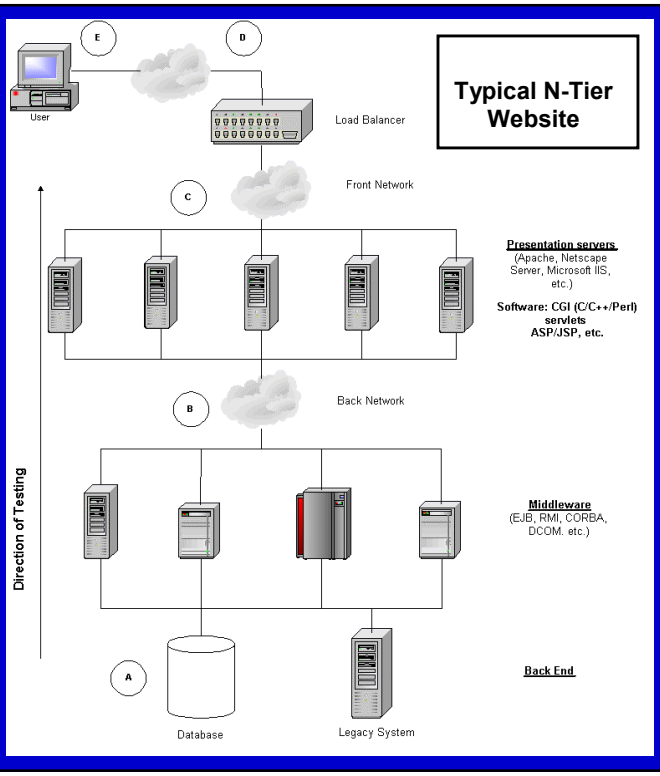
33

N-tier Website
(C) Web Servers

- Load test the sub-system of the entire Website: "Pound" on your Web servers
  - – independent of outside Internet connection
- Located in front of your Website's presentation servers on your internal network
  - – not going outside to the Internet

ParaSoft

34



ParaSoft

35

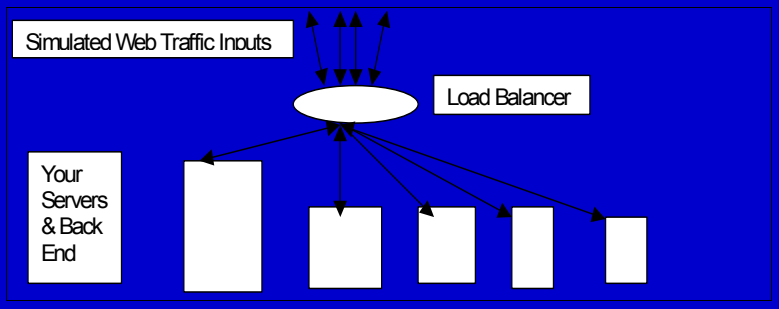
Web Servers: Load Balance

- Try to “pound” your Website with as much traffic as you can
- Do load tests that emulate your users using realistic paths
- Repeatedly execute these scenarios and load test your Web servers

ParaSoft

36

Web Servers: Load Balance



The solution of load balancing among your different Web Servers, independent of the adequacy of the pipes coming into your Website.

ParaSoft

37

Web Servers: Load Balance

Measure items such as the following:

- Load time of each page
- Number of pages loaded per second
- Simulation of x-thousand users, based on expected load
- Utilize multiple machines to emulate a large number of users

ParaSoft

38

Web Servers: Load Balance

If performance degradation is noted:

- Consider load balancing among Web servers
  - Incoming traffic
  - Outgoing traffic
- Consider additional Web servers

ParaSoft

39

Web Servers: Load Balance Up to this point…

- Elements already tested
  - Individual servlets
  - Database & legacy system
  - Business logic interactions
- Internal Website sub-system tested
- What next could affect your load?

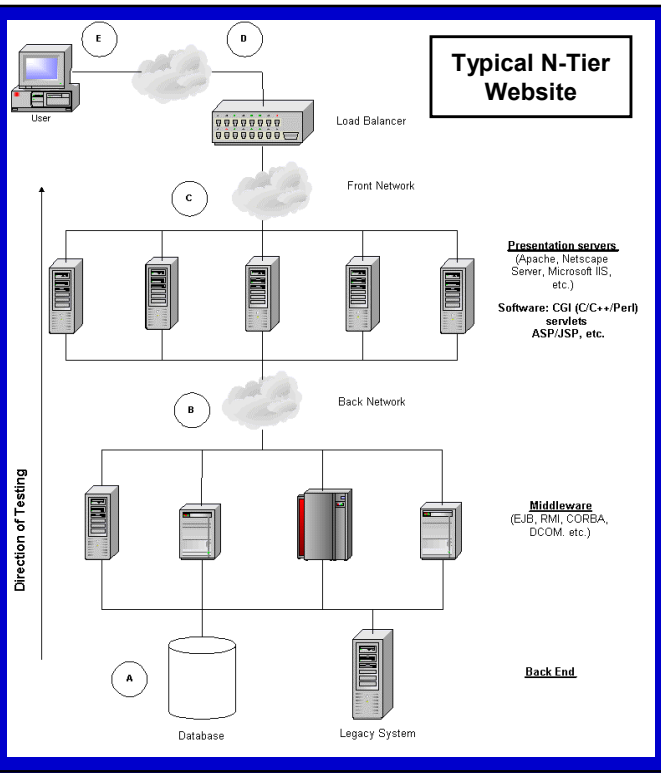
ParaSoft

40

N-tier Website
(D) Bandwidth & Pipes

- Test your connection to the Internet: Bandwidth and Pipes
- Located on your outside connection to the Internet

ParaSoft



ParaSoft

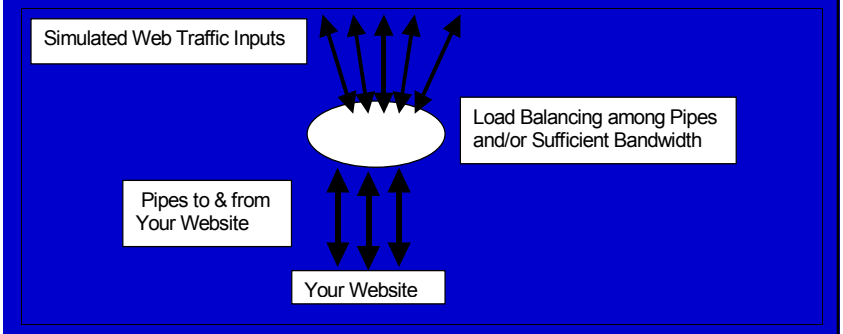
Bandwidth & Pipes

- Execute the same load test simulations you did from "inside your application"
- Solve any performance degradation caused by:
  - Outside pipes
  - Insufficient total bandwidth or
  - Unbalanced load among the pipes

ParaSoft

43

Bandwidth & Pipes



The solution of load adjustments to the quantity and configuration of pipes to your Website.

ParaSoft

44

Bandwidth & Pipes

Even if all previous load tests were adequate…

Problems with this pipe could cause your Website to perform poorly under load

ParaSoft

Bandwidth & Pipes
Up to this point…

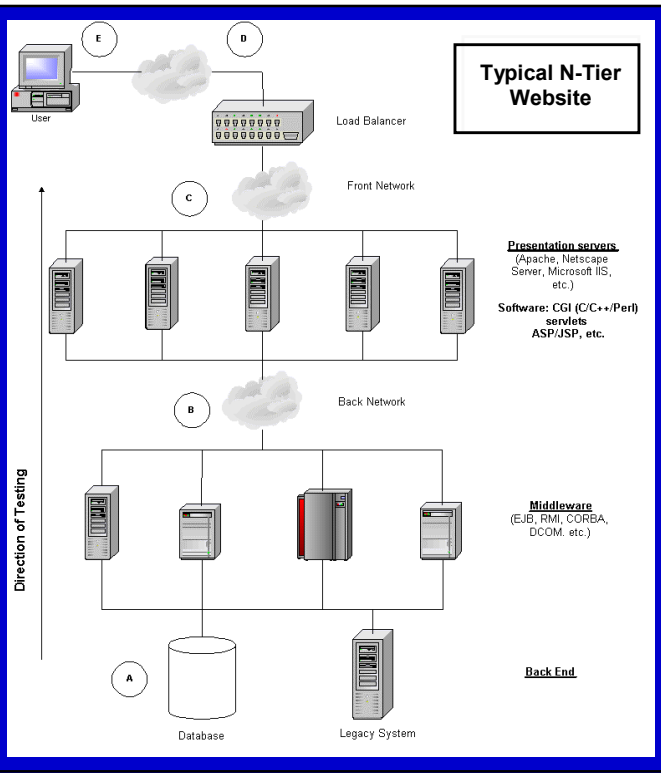
- Tested elements
  - Individual servlets
  - Database & legacy system
  - Business logic interactions
- Tested internal Website sub-system
- Tested external pipe sub-system

ParaSoft

N-tier Website
(E) That Last Mile

- Customer and user experience - That last mile
- Located at the very top left - getting feedback from all other elements and sub-systems of your Website

ParaSoft



ParaSoft

That Last Mile

- Test Your Customers' Experience
- Broad-based usability testing:
  - Test actual customer experiences
    - Different origination points
    - Different ISPs, platforms, connections, neighborhood factors, etc
- Difficult to do usability testing
  - High cost
  - High inconvenience
  - Typically not done

ParaSoft

49

That Last Mile

What is your customer's experience with your Website?

Without usability testing, how will you find out?

ParaSoft

50

That Last Mile

In addition to Usability Testing, there are two techniques:

- Testing of output page effects
- Static analysis - coding standards

ParaSoft

51

That Last Mile

Output Page Effects:

- Sluggish download pages due to
  - Too many banners
  - Too many tags
  - Too much unnecessary info
- Not load testing, but a problem of badly designed Web pages
- Customer impact is the same:
  - Slow page builds
  - Perceived slow performance

ParaSoft

52

That Last Mile

To solve output page effects, avoid the following:

- Unnecessary items on Web pages
- Dynamic generation of static data
- Unnecessary white space
- Absolute links, when relative links will work
- Unnecessary tags or empty tags

ParaSoft

53

That Last Mile

- Static Analysis is used to prevent design mistakes in the first place
- Coding standards example:
  - Automatically flag unnecessary tags (eg: empty tags) that were inadvertently introduced into your code
  - Removing these tags will speed Web page downloads
  - Important in performance speed by your user

ParaSoft

54

That Last Mile
Up to this point…

- Tested every servlet and code component
- Tested every element and sub-system - (A) through (E)
- Analyzed customer experience perceived problems
- Next: three step conclusion

ParaSoft

1. Conclusion

Your Customer or User:

- Experiences your Website as a single entity
- Does not care why your Website is sluggish or not performing acceptably
- Has other choices if your Website is disappointing for any reason

ParaSoft

2. Conclusion

- Independently load test each element and each sub-system
- Load test early and often
- Do not skip any levels
- Load testing begins with design and ends with your entire application
- Use automatic techniques to help whenever you can

ParaSoft

3. Conclusion

- **Your Website is only as fast as its slowest element**
- **Unit level load test every element**

ParaSoft

References

- White Paper, “Improving the Quality and Efficiency of Dynamic Web Site Development and Testing,” by Adam Kolawa, PhD.
- This and similar technical papers available at: www.parasoft.com

ParaSoft

Contact Info:

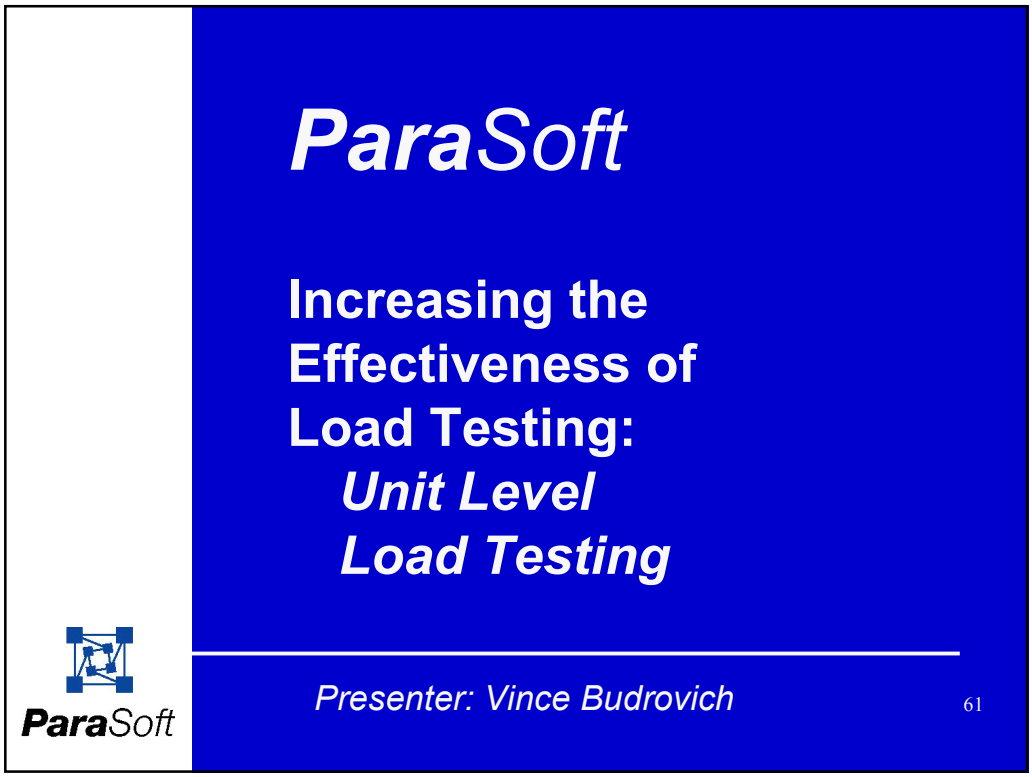
vince@parasoft.com

(626)256-3680 x1249

www.parasoft.com

ParaSoft

Presenter: Vince Budrovich


ParaSoft
Increasing the Effectiveness of Load Testing:
Unit Level Load Testing
ParaSoft
Presenter: Vince Budrovich
61

Increasing the Effectiveness of Load Testing: Unit Level Load Testing

Introduction and Rationale

Load testing is usually considered a quality assurance (QA) issue. It is a form of testing that is performed after the application or Web site is finished. But this approach often leaves the discovery of major problems until the end of a project. This in-turn pushes up development costs and delays the implementation of new functionality. One way to avoid this is to load test from the beginning of development and continue throughout the development process.

The main reason why it is so risky to delay load testing until the end of development is that load problems are typically the result of "difficult-to-fix" design and algorithmic flaws. Fixing these types of problems often requires significant rewriting, not just the flawed portion of code, but all the affected areas of the code. The longer you wait before you find and fix one of these flaws or bugs, the more code you will have to re-write.

For example, imagine that you have a servlet that was developed months ago and other servlets depend on it. Then you just learned that one of your Web site's servlets cannot handle more than a dozen users at the same time. Fixing the problem or bug now can be anywhere from difficult to disastrous. This problem may indicate a design or implementation problem, such as a poorly designed database or poorly designed objects. If so, fixing this problem could involve modifying your entire design or implementation strategy and redesigning and/or rewriting the entire application

If you had load tested this servlet immediately after it was written, you could have spotted this scalability problem earlier on. As a result, you could have also spotted and reworked design and implementation flaws before they became widespread. Essentially, you could have prevented many problems that became very difficult and costly to fix later in the process.

For another example, say you have a routine that is pulling records from the database. But instead of pulling the entire record, it is only pulling the record field by field, which is slowing the response time of the site. If this problem is not discovered until QA performs load testing, not only will you have to redesign the algorithm that pulls the records from the database, you will then need to modify the processes that interact with that algorithm. Making these modifications will take a significant amount of time and effort.

These examples illustrate why you should start load testing at the unit level. "Load testing at the unit level" means that each time you create or modify a dynamic site component, such as a specific customization option or a "wish list" functionality, you should immediately test how well it performs with different types and number of user

traffic. This also means is that as each piece or layer of your entire Web site is developed, you'll immediately load test that element or layer. For example, as soon as you develop a servlet, you'll load test that individual servlet. You'll want to keep performing regression tests so you know that new additions and work-around fixes have not inadvertently introduced any new bugs into your code.

In our analysis, we are taking a broad definition of unit testing. We are not only suggesting that you should test each individual servlet and each major sub-system or element in your Web site, but you should also load test it independently of all the other elements. The reason for this is that if you notice poor or sluggish performance at the customer end, you have no reliable way of isolating the exact cause of the problem. Also, your procedure of finding and fixing the problem can be out of control and you will have no way of predicting how long it will take to remove the problem.

One other common pitfall is that problems that load testing detects won't be apparent until the Web site is up and used. Load testing problems are not easily identifiable and other problems are much more visible to developers. For example, if a bug causes the Web site to crash, it obviously gets immediate attention. But if a bug causes the Web site to be slow to your customers during normal load usage, it will not be noticed if you have other "bigger problems." Thus, it's critical that you utilize a clean development process. If your development process is not clean, you may be so busy going from crisis to crisis to fix immediate bugs that you'll never have time in the pressurized development process to systematically proceed with load testing at all.

As stated, load problems can arise from a number of different elements or levels in the development and the operation of your Web site. Detecting and preventing load problems therefore requires taking a comprehensive point of view of your entire Web application and conducting unit level load tests on each of these elements or sub-systems. We'll then proceed, step by step, through the process of looking at each element in order:

- Web-box testing – how to load test your servlets and components
- Test database and legacy system
- Test business logic interactions
- Looking at the total load on your Web servers
- Looking at the total load on your pipes or connection to the Internet
- Your Customer's or User's experience.

Following is **Figure 1 (Typical N-Tier Web site)**, a simple graphical layout that shows several levels, including elements or sub-systems. We will proceed through examples of load testing from each of the following levels or sub-systems. The direction we will take through Figure 1 starts with load testing an individual servlet, which can be located on any server in your Web site. Then we'll proceed to explain load testing from element to element as we work our way up the page, through A, B, C, D, and E – ending at the top left corner with your User.

A. At the bottom of Figure 1 is your Back End which includes your databases and legacy systems.

B. Your middleware is one layer up, which includes, for example, servers with the business logic for one of your programs.

C. An additional layer up brings us to your presentation servers, which includes the servers that display the pages your customer is viewing.

D. Above that level are your pipes, which comprises of the total bandwidth connecting you to the Internet.

E. On the very top layer, (off to the left) is your User or Customer.

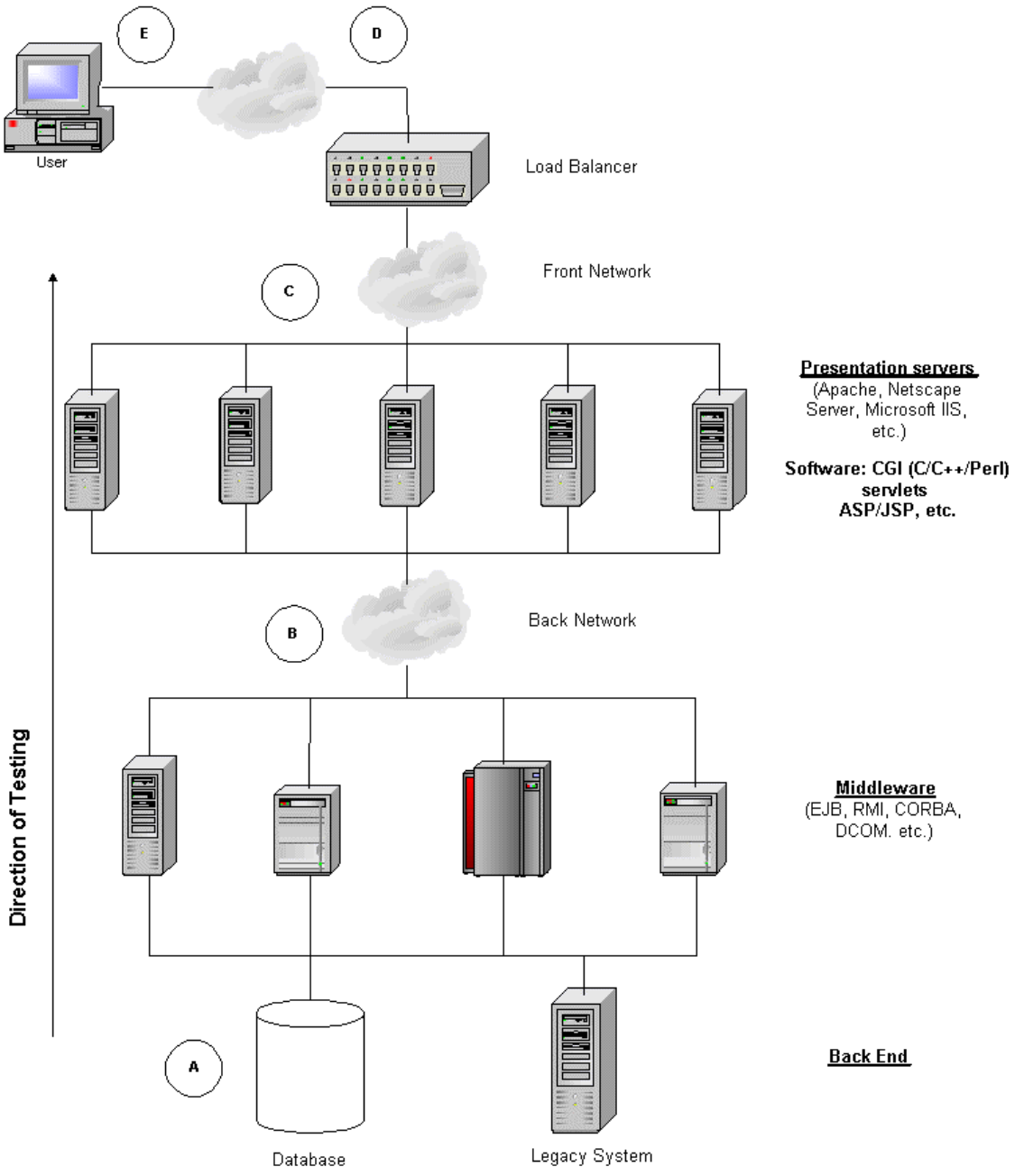


Figure 1: Typical N-tier Web site

Web-box Testing - Testing Servlets and Components

"Web-box testing" is a set of techniques that facilitate the unit testing process you perform as you deploy and test servlets and components. We'll walk through the steps, starting with writing or creating your servlet and then proceeding through the steps to conduct unit level load tests on your servlet.

There are several steps in this process. Greatly simplified, the process would include the following steps:

- Write the servlet.
- Compile the servlet.
- Transfer the executable servlet to the right place on the designated servers.
- Invoke the servlet to activate it. [Note: It is not easy to activate the servlet indpendently of everything else. Therefore, developers often will skip this step and not activate the servlet until the rest of the system is built. However, if they skip this step they are unable to individually load test this servlet.]
- Once invoked, give the servlet the correct arguments before it starts running to look at the page.
- Test to see if the servlet by itself is correct, then execute it and see its output page.
- Test if the page created by the servlet is correct.

As you can see, this process is complicated and error prone. Developers, as they work on the code, repeat it many times. Very often they try to automate it by building homemade scripts. There are tools on the market [Ref 1] which can do the same for you. In addition, the same tools can perform load testing at that level. They automate the whole process of the Web-Box Testing.

If you do not have the tools, you will have to perform the following steps to complete the Web-box testing process.

- If you can invoke it once and test that it's correct, you will write scripts (or test cases) to repeat that test and to invoke it repeatedly in order to test the servlet or component under load.

- You will pound on the servlet by doing unit level load testing (ie: by calling up that servlet again and again to emulate actual load conditions that it will need to match).

- For example, you may need to see it display x-thousand times per minute in order to fulfill your expected Web site load requirements.

If the servlet does not perform as required, you will probably have to fix the algorithm in that individual servlet. However, you only have to fix this individual servlet and not have to re-do other parts of your application that the servlet will eventually interact with. This is because you are load testing the servlet and fixing it before you create the other parts of your application.

Fix code after it has been written. It will take less time to fix an problem early on rather than two weeks and two dozen servlets later in the process. Repeat this process with each servlet and component of code that you create. In addition, you'll want to perform regression tests. Regression testing is where you repeat the entire test suite to see if any new problems emerge from code already written, tested, and corrected.

In short, by doing unit level load testing on this servlet, you can immediately and simply solve load performance problems and avoid other problems later in the development process. This is an added benefit given the compressed timeline demands typically associated with most Web site development projects.

[For more information on automated techniques of unit level load testing for dynamic Web sites, see Ref. 1]

Load Testing Database Interactions or Legacy Systems

We now know that the individual servlets that you've created have been successfully load-tested on a unit level basis. Now we will describe the testing of other parts of the Web site. We start at point **[(A) in Figure 1].**

For the purpose of this analysis, we'll assume your Web site has a database and a legacy system, and both are part of your back end. We'll first look at your database.

To test the load effectively, you really need to "pound" on the database to see if it can process all the transactions that your Web servers and the rest of your application generate. How can you pound on it? You need to emulate the behavior of the Web server in order to emulate the requests which are going to the database. Then you measure the responses in order to be sure that this database is not going to cause a bottleneck.

Note: It is very important that this be done as part of the development process, not after you've built the Web site. If you've already built the Web site, it will be too late and extremely difficult for you to fix. This is because the problem may be so complex that it may require you to redesign and rewrite the entire application.

How do you perform unit level load testing on this element?

- You create a set SQL scripts.
- These SQL scripts execute the full transactions on your back end.
- Measure the timing and see if it can be done.
- With the thousands and thousands of transactions, you see what happens.

For more information look in [Ref 2]

In addition to a database, we assume you have a legacy system and you need to find a way to simulate it in order to perform unit level load testing on its interactions. You should be able to use XML or EDI as testing protocols, which will help you load test the behavior of your legacy system. These tools are in the process of coming on the market.

Load Testing Interactions with Business Logic

Now that you've tested the database as well as legacy system interactions in your back end, you'll want to go to the next layer up on the N-tier Web site [Figure 1] and examine the business logic. **[(B) In Figure 1]**

The business logic is typically the piece of your Web site that is written in Java (or Perl). This represents the correspondence between converting the requests from your users or your customers into requests for the database or servers. It then performs the logic or does calculations as specified. For our analysis, we will assume the business logic is sitting on one of your middleware servers.

- How well does this software really operate?
- How does it respond to the requests?
- Is it fast enough for your Web site?

At this moment you have tested "two layers" – one layer is the database or legacy system and the other layer is the individual servlets. So you know that individually, each of those elements is performing in an acceptable manner. Therefore, if there is slow or other unacceptable performance at this moment, you know that it's caused by the interaction between the servlets and the back end. You are actually testing these interactions; for example, how the servlet business logic is talking to the database. Or, in another example, you might be load testing the interactions between multiple business logic elements.

Testing Your Web Servers:
Independent of (outside) Internet Connection

Up to this point, you have created a set of scripts to test your servlets. These are in effect a set of test suites. You have tested these single servlets and you know that the pages of the servlets are sound under load conditions. You have also load-tested interactions with the database and business logic. Next, you'll move one layer up on the N-tier Web site [Figure 1] and look at testing the sub-system of your entire Web site, independent of your outside connection to the Internet. **[(C) in Figure 1.]** You are technically on your internal network or LAN (Local Area Network), not going outside to the Internet.

In order to load-test this sub-system, simulate as much traffic as you can, as realistically as you can. You really need "to pound" your Web site with this traffic. Above all, you're trying to see how your Web site is going to respond to this traffic. You can use tools which will help you automate this process. [Ref 2]

Since you are inside your own location, you're not going to see any outside network or Internet bottlenecks. If you stay inside, you remove the issue of how much bandwidth is coming into your Web site. Instead, you are testing how your Web site itself is in a position to respond to all the requests that might come to it.

You're now load testing the Web site itself – not the pipes coming into it. You are testing the Web site's ability to respond to this wide variety of tests. You should conduct a set of load tests that emulate numerous individual users using realistic paths to use your Web site. These paths are executed with multiple attempts and realistic scenarios.

It's very important that testing be set-up at this level. If it works and you get the desired performance while conducting these tests, you'll be measuring items like:

- Load time of each page
- Number of pages you can load per second
- Simulate x-thousand users, based on realistic expectations for your Web site
- Utilize multiple machines which can emulate a large number of users

In short, you'll just keep "pounding" these tests on your application. If performance degradation is noted as a result of your load testing, then you need to consider load balancing among your multiple Web servers or adding servers. As the following **Figure 2** indicates, you may need to add servers or adjust the load among your Web site servers in order to get the desired performance.

This balancing might be on the input for incoming traffic to your servers or the output for outgoing traffic from your servers. Depending on the particulars of your load, you might need to balance both the input and the output.

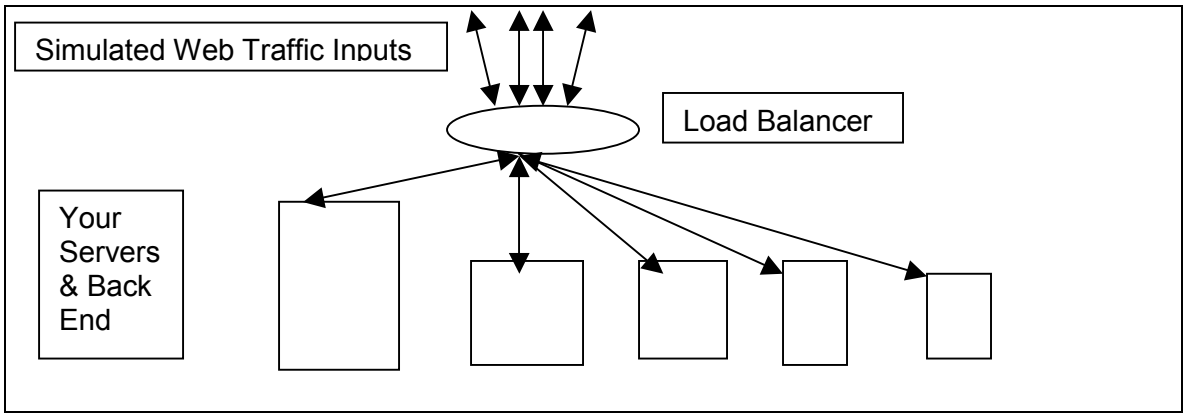


Figure 2: Load Balancing Your Servers: shows the solution of load balancing among your different Web Servers, independent of the adequacy of the pipes coming into your Web site.

Testing Your Pipes – Independent of your Servers

Once you have load tested this sub-system and made any necessary adjustments to ensure that your Web site is responding properly, then you will move one layer up on the N-Tier Web site [Figure 1] and test if your pipes coming in from the Internet are balanced or big enough. **[(D) In Figure 1.]** After you have pounded your Web site internally and made necessary adjustments to balance the load between your Web servers, you are ready for this step.

You now need to thoroughly test your pipes, such as the bandwidth to the outside Internet. You test this by running and executing the same simulations you just exercised in the previous step from "inside your application." But this time you are trying to see if there is a performance degradation that is caused by insufficient bandwidth of the outside pipes or if those outside pipes are configured incorrectly.

You now know that your Web servers are up to the task, since you have already adjusted and balanced the load among them. Now, as **Figure 3** illustrates, test the input pipes with the same tests you performed when you were testing the servers.

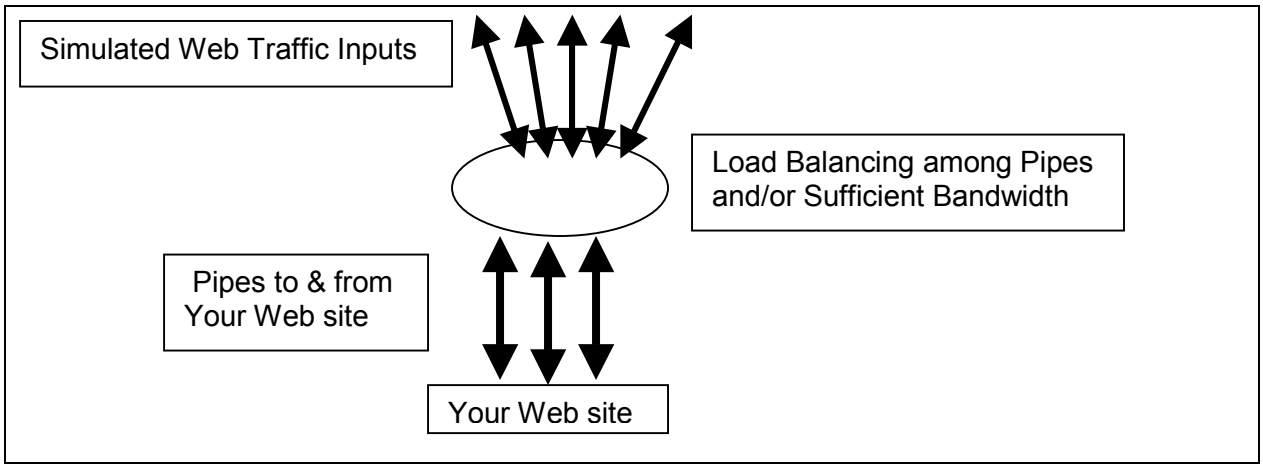


Figure 3: Your Pipe(s) & Bandwidth: shows the load adjustments to the quantity and configuration of different pipes to your application, after the application is load tested itself.

If there is significant performance degradation, then you know that you need to consider adding bandwidth or balancing the load better among the several pipes. The load problem here is not your Web site, the problem is the clogged pipe or pipes going into your Web site. Again, the solution to this element is to add to your total incoming bandwidth and/or to reconfigure your outside pipes.

Without making the necessary adjustments at this level, everything else could have tested adequately, but your customer could still have a bad experience because of this bottleneck in your connection to the Internet. Conversely, if you had not previously load tested all your other elements and sub-systems, you could not be sure that the degradation in performance was due to this particular sub-system.

Test from Customer Site: That Last Mile

Just as in the previous steps, by systematically conducting unit level load testing of each element and making any required adjustments before going to the next element, you have been able to quickly solve load degradation performance problems of your Web site. That brings us to the next level up on the N-tier Web site [Figure 1], the top level which is called the user or customer experience or "That Last Mile."**[(E) In Figure 1]**

"That Last Mile" refers to the final element in connecting your Web application with your customers. It is that final bit of phone wiring, the true processor with limited speed, the neighborhood electrical power reliability factors, and all the other issues that determine your customers' actual experience. Your user or customer is at the very top level of the chart in Figure 1 – at (E), off to the left.

To perform the tests of the actual customer experience, emulate the behaviors of your actual customers in attempting to utilize your Web site from their homes and offices.

Test their actual experiences (or emulate their experiences) from several dozen different origination points. This includes a variety of local conditions, different ISPs, different browsers, different platforms, different phone/cable/data connections, etc. Does your Web site do what it's supposed to do? Is it fast enough?

Frankly, this level of usability testing is seldom done due to the high cost and inconvenience. Of course, it should be done because it may reveal insights that may have gone undetected on all your previous tests. If performed, you can get the real test of your Web site's actual performance.

But, if you're not likely to directly test this element of your Web site due to the obvious cost and inconvenience of such tests, what else can you do? What can you do to help ensure your customers' experience with your Web site, will be a positive experience? Two additional techniques that can help you are the testing of output page effects and coding standards enforcement.

Test Output Page(s) Effects

What is your customer really seeing when they download pages from your Web site? Are they slow? Typically, they can be very slow because of your customers' limited bandwidth of the pipe or connection to the house or small business.

The slow download pages can be due to design problems. This problem could be detected early in the development process with static analysis techniques by enforcing coding standards. In this example, do your Web pages have too much "stuff"?

- Too many banners?
- Too many tags?
- Too much unnecessary "stuff" that is slowing down the page downloads?

In this final mile, is your site slow due to these download issues? The answers are related to the original design of the pages. The key to solving download sluggishness is to send a minimum amount of data, which is an issue of design, not an issue of load testing.

One of the particular things you want to pay attention to is that minimal download time is critical in order to avoid slow Web page builds. The obvious way to reduce download time is to reduce image size and page content, above all, by not including unnecessary items on the page in the first place.

In addition, a few problems that you need to avoid which are not so obvious include the following:

- Dynamic generation of static data: **Many** sites unnecessarily increase their pages' download time by performing "dynamic" generation of static data. Any time server-side technology is used to dynamically generate the same exact page, over and over again, you should create a static version of the page and, if feasible, build an infrastructure that updates the page frequently.

- Unnecessary white space: White space is often used to increase code readability. However, white space increases file size, and file size directly affects both file download time and server load. Removing excess white space typically results in a 10-50% percent reduction in file size, which results in a 10-50% reduction of download time and server load.
- Absolute links: Using relative links (e.g., /press/index.htm) instead of absolute links (e.g., http://www.parasoft.com/press/index.htm) reduces download time and server load. Also, because relative links have fewer characters than absolute links, using them also increases download speed because it reduces file size.
- Unnecessary tags: Browsers read and render every tag that they encounter, even tags that do not contribute to page presentation or functionality, such as empty tags. By eliminating the excess tags that are often added by code-generation tools or sloppy coding practices, you can significantly reduce rendering time.

Static Analysis – Coding Standards

As previously mentioned, use static analysis to flag problems before they affect your customer. Technically, you may not have a slow performing Web site, but these Web page builds may have resulted in the perceived slowness of your Web site from your customer or user.

Static analysis is the enforcement of effective coding standards throughout the development process to significantly speed up your customers' experience on your Web site. Effective coding standards enforcement can prevent you from making these mistakes in the first place.

In this case, you do not just load test this site, you should actually employ the rules contained in an automatic coding standard rule enforcement mechanism. Coding standards are language-specific "rules" that prevent errors or other practices by not introducing mistakes into your code. For example, a coding standard could automatically alert you to any unnecessary tags, such as empty tags, that were inadvertently introduced into your code. If present in sufficient quantity, these empty tags could significantly slow down your Web page downloads with your actual customer.

If you prevent some of these errors or mistakes, it will impact the performance of your Web site as perceived by your customer, and you will decrease the chance of having your Web site appear slow, even if it is not technically slow.

[For more information on automatic coding standards in your development process, see the white paper, "Automatic Unit Testing for Java Developers," by Dr. Adam Kolawa, CEO, ParaSoft Corp.]

In our example, paying attention to your output pages effects and performing static analysis by enforcing coding standards, would have led to correct design decisions early in the development cycle. This could have prevented the perceived sluggishness of your

Web site, even if you could not afford to perform usability testing from the customer level. In the Last Mile, an ounce of prevention would have been worth far more than a pound of cure.

Conclusion and References

Is the performance of your Web site meeting the expectations of your customers or users? Your customers will experience your Web site as a single item, regardless of its many complex levels. Are load problems in any level of your Web Site causing a negative customer experience?

Since it is in your best interest to find any load problems with individual elements as soon as possible in your development cycle, you should perform your load testing early and often. Do not wait until your Web site is up and running before you load test. In addition, whenever possible, you should utilize techniques that can perform the load testing automatically.

If you do not conduct unit level load testing on each of the elements in your Web application, how will you know what is the cause of the poor performance of your Web site? In addition, how will you know precisely what elements to modify to fix the problem?

The key to the successful load testing of your dynamic Web site is to verify that each layer, element, and sub-system is responding adequately and fast enough so that you know it is not the cause of a performance slowdown. After you test each element independently, you should immediately fix any load related performance problems in that element. You should be performing unit level load testing throughout your Web site development process, not just load testing the complete application when it goes live. In the same way that a chain is only as strong as its weakest link, under load conditions, your Web site is only as fast and functional as its worst performing element.

References

For any white papers, additional information referenced in this paper, or techniques that automatically perform specified tests, go to: www.parasoft.com

1. “Improving the Quality and Efficiency of Dynamic Web Site Development and Testing”

2. “Preventing & Detecting Errors in N-Tier Dynamic Web Sites”

3. “Automatic Unit Testing for Java Developers”

4. “Coding Standards in Java: Do We Need Them?”, Java Report, 3/2000, by Adam Kolawa, PhD

5. “Testing Enterprise JavaBeans” , Intl Conference for Java Development, 3/2001, by Adam Kolawa, PhD

6. “Automating the Development Process,” Software Development, 7/2000, Adam Kolawa, PhD

List of Figures:

Contact Information:

You can contact the author directly at:
Vince Budrovich, Instructional Systems Mgr.
vince@parasoft.com
(626)256-3680 x1249
ParaSoft Corporation
2031 S. Myrtle Avenue
Monrovia, CA 91016

QW2001 Paper 8A2

Mr. Scott Trappe
(Reasoning, Inc.)

Building Better Software Code: Finding Bugs You Never Knew You Had

Key Points

- Why current software quality control processes are inadequate and often miss fatal defects
- The pros and cons of different source code inspection methodologies
- Implementing automated software inspection as an outsourced service

Presentation Abstract

One of the biggest challenges facing software developers is producing high quality, defect-free code in ever-shrinking market windows. One aspect of this challenge is ensuring that the software is property tested and inspected for defects. To date, however, there have been a number of limitations to these software quality processes: they often miss dangerous defects on infrequently executed paths--these defects include null pointer dereferences, memory leaks, out-of-bounds array accesses, and uninitialized variables. Such defects usually cause the application or system to crash, or they cause data corruption. The cost of failure particularly for embedded systems can be quite high since the software often controls safety-critical equipment. Financial applications, particularly Internet applications, have a similarly high reliability requirement. In addition, in Internet applications, any defect that allows a user to crash the application or produce invalid data also presents a security vulnerability.

Formal source code inspections have long been a recognized approach to finding these kinds of defects, in addition to providing an overall quality assessment. However, the resource requirement (in terms of training, time and cost) is usually prohibitive. Tool vendors have attempted to address this issue with automation, and today there are many source code inspection tools available. Most of these tools focus on overall code quality and provide metrics for assessing quality. However, their ability to identify true defects is limited, both in terms of the number of false positives produced, and the complexity of defects that can be detected.

Recently, a new approach to inspection has emerged, automated software inspection services. These approaches use recently developed software analysis technologies including value lattices, computation analysis graphs, and theorem provers toinspect source code for specific classes of dangerous defects. These approaches can eliminate most of the false positives that characterize

pattern-oriented methods, and can detect much more subtle defects than lint-like tools whose context is limited to a single function.

In this session, we look at these different software inspection solutions and provide scenarios for when each should be used, how they operate, and what results can be expected. We also compare software inspection solutions with traditional testing, and identify the strengths and costs associated with each.

About the Author

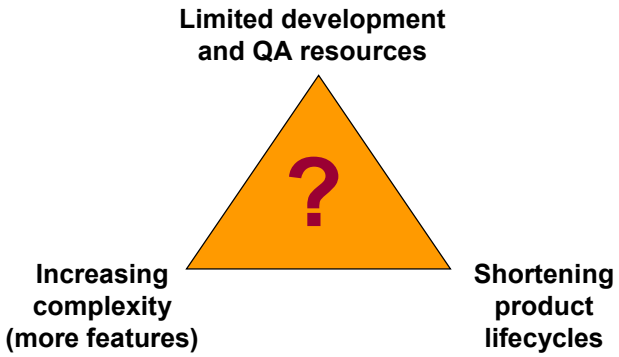
Scott Trappe is President and Chief Operating Officer for Reasoning Inc., which provides the InstantQA automated software inspection service. With more than 20 years of experience developing and marketing software development tools and services behind him, he has contributed numerous articles on the subject to software development publications and hosted presentations at industry-related events. Mr. Trappe joined Reasoning from Intrinsa, where he held a dual role of Vice President of Engineering and Marketing. Intrinsa successfully developed and marketed an automated source code inspection product similar to InstantQA. Intrinsa was acquired by Microsoft in 1999. Prior to Intrinsa, Mr. Trappe held a variety of marketing and engineering management positions with Netopia; Operations Control Systems and Tektronix. He holds an MBA from the Haas School of Business at the University of California, Berkeley and a BS in Computer Science from the University of Arizona.

Reasoning

Building Better Software: Cost-Effective Defect Detection & Elimination

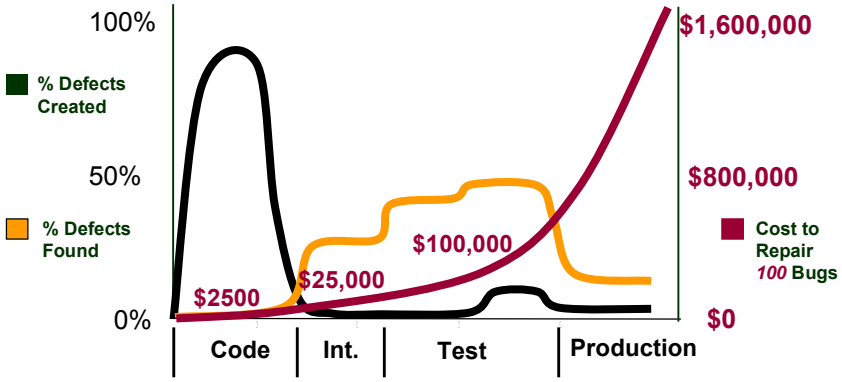
The Problem

Releasing Reliable Software on Schedule is Almost Impossible



The Problem

Finding Bugs Late is Expensive and Time-Consuming



Source: Capers Jones

Need to Find Bugs Sooner

The Problem

Testing has Become the Bottleneck

"We spend more time on testing than we do writing code. What kind of new techniques are there in testing? Very, very little."

Source: Bill Gates, Microsoft

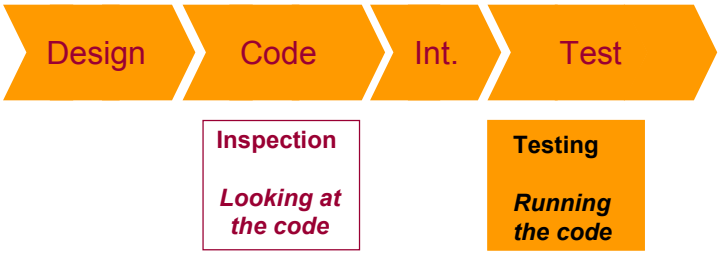
"Bug fixing is at least 30% of development effort and increases with software complexity."

Source: Capers Jones

The Need

Inspection Finds Bugs Earlier

Software Development Lifecycle



"Inspection is by far the most effective way to remove bugs."

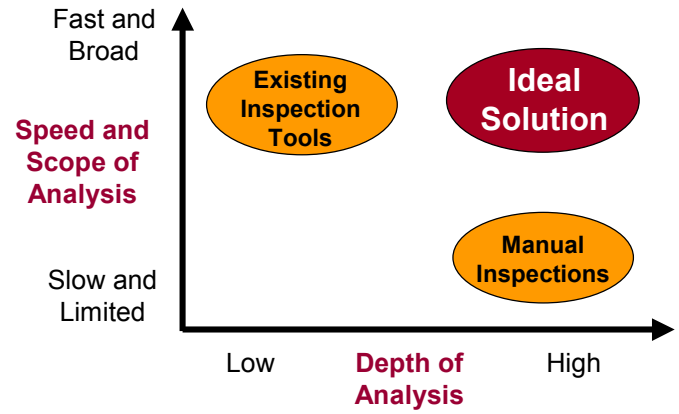
Source: Capers Jones

What Inspection Can Find

- **With inspection, we can find defects that hide on infrequently executed paths:**
  - **Null pointer dereferences**
  - **Bad array accesses**
  - **Memory leaks**
  - **Uninitialized variables**
- **With inspection, we can check for adherence to coding standards**

The Need

A Practical Way to Inspect Software



Existing Inspection Tools

- **Focused on primarily on coding standards enforcement**
  - **Code Wizard**
  - **PC Lint**
- **Only identify possible symptoms of software defects**
  - **Large number of false positives**

Manual Inspections

- **An effective but labor intensive way to find problems:**
  - High resource cost (training, time and money)
  - Lack of enforced standards
  - Can create a defensive work environment
  - Difficult to find developers willing to do it
  - Time constraints with large programs

Not practical for today's development organizations

InstantQA

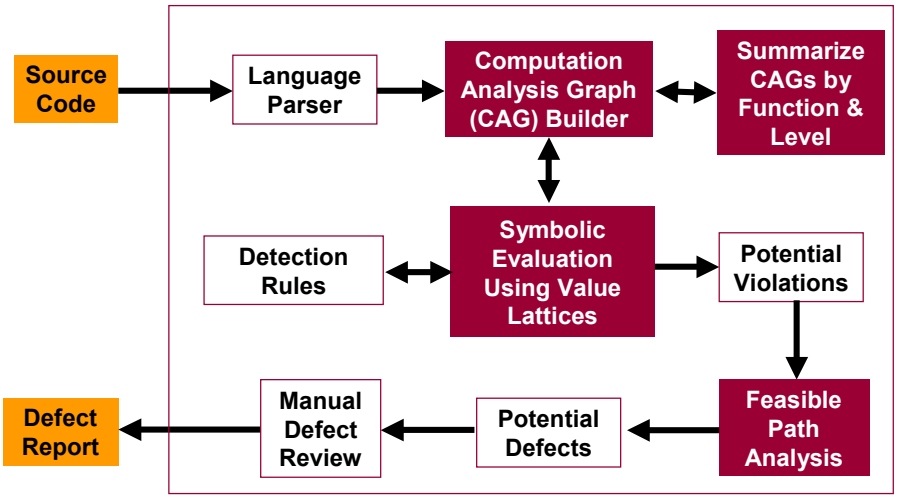
Automated Inspection Service that Finds Bugs without Testing

- Reviews 100% of code
- Finds crash-causing bugs
- Works on incomplete code
- Hardware independent
- Collects quality metrics
- No impact on in-house resources

Like CAT Scan - finds problems without surgery

InstantQA Technology

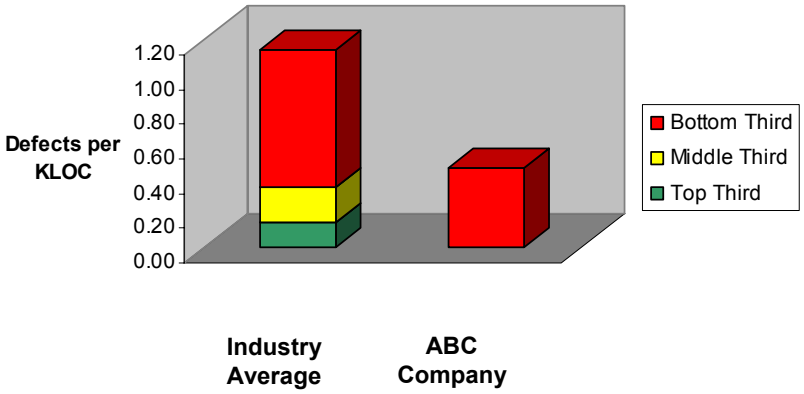
Architecture



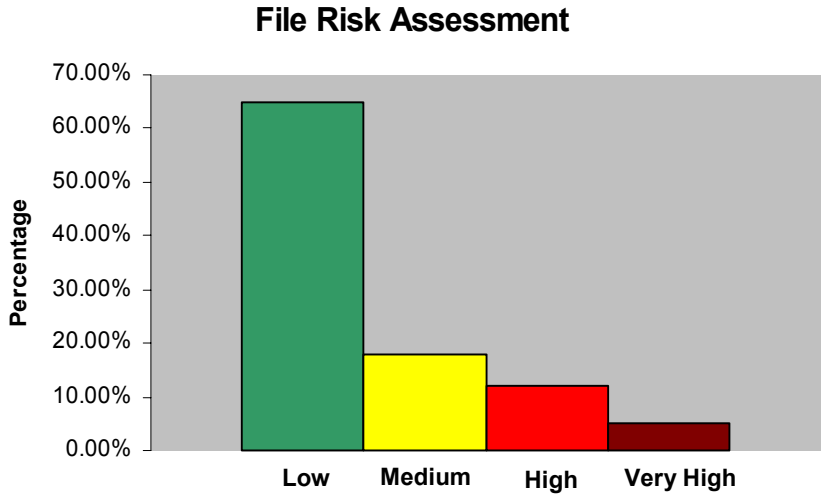
= Reasoning Innovations

InstantQA Quality Metrics

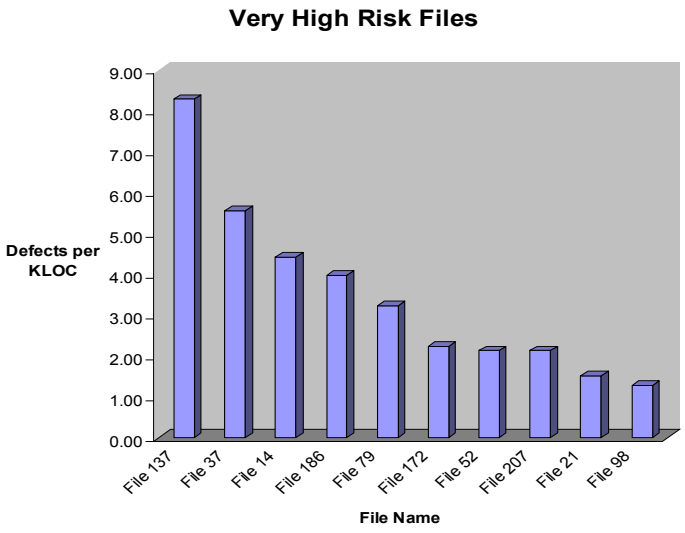
Industry Comparison



InstantQA Quality Metrics



InstantQA Quality Metrics



InstantQA Detailed Bug Report

Detailed Defect Report

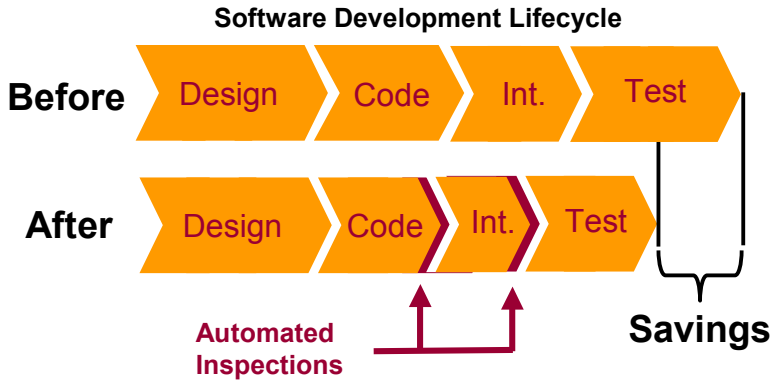
Defect Class	**Memory Leak**	Risk **Moderate**
Location	**/websrv_1.1/src/os/win32/readdir.c : 43**	
Description	**Pointers to blocks allocated by** malloc() **on lines** 34 **and** 27 **are stored in local variables** dp **and** fspec**. The memory blocks become inaccessible (still allocated, but unreachable) once** dp **and** fspec **go out of scope after line** 43**.**	
Preconditions	**The expression `(errno == ENOENT)` on line** 40 **is false and `((handle = findfirst(fspec, &(dp->fileinfo))) < 0)` on line** 39 **is true**	
Impact	**Memory leaks cause performance degradation of the application, and/or the entire system. Eventually, this may lead to a fatal out-of-memory condition.**	

Code Fragment

```
20  API_EXPORT(DIR *) opendir(const char *dir)
21  {
22      DIR *dp;
23      char *fspec;
24      int ix, handle;
26
27      fspec = malloc(strlen(dir) + 2 + 1);
28      strcpy(filespec, dir);
30      if ((ix = strlen(fspec) - 1) >= 0 && (fspec[ix] == '/)
31          fspec[ix] = '\0';
32      strcat(fspec, "/*");
33
34      dp = (DIR *)malloc(sizeof(DIR));
…
39      if ((handle = findfirst(fspec, &(dp->fileinfo))) < 0) {
40          if (errno == ENOENT)
41              dp->finished = 1;
42          else
43              return NULL;
44      }
```


Benefits of InstantQA

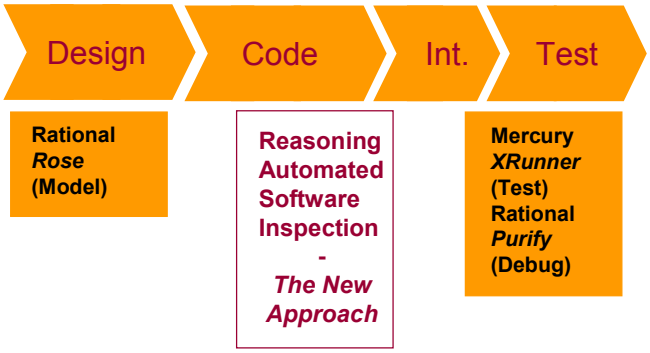
- **Reduced Time and Cost**
- **Increased Reliability**



InstantQA Completes the Cycle

Complements Design & Testing Tools

Software Development Lifecycle



Questions?

QW2001 Paper 9A1

Dr. Harmen Sthamer & Mr. Joachim Wegener
(DaimlerChrysler AG)

Evolutionary Testing Of Embedded Systems

Key Points

- Evolutionary Test automates entirely testing
- Test case design is difficult
- Powerful tool environment

Presentation Abstract

More than 90 % of all produced electronic components are used in embedded systems. Embedded systems have usually to fulfil functional as well as temporal requirements. Most of the embedded systems applied in vehicles are subject to temporal requirements. This is due to reasons of operational comfort (short reaction times of the vehicle MMI to driver commands) and due to the requirements of technical processes which are controlled within the vehicle. Examples are engine control systems, body control systems like ABS and ESP, and airbag control systems. Often these systems are also safety-relevant.

For embedded systems testing is the most important quality assurance measure. It typically consumes 50 % of the overall development effort and budget. Essential to a good test quality is the systematic design of test cases. Test case design defines the kind and scope of the test. Test case design is difficult to automate. For functional testing the generation of test cases is usually not possible since no formal specifications are applied in industrial practice. Structural testing is also difficult to automate due to the limits of symbolic executions. Furthermore, for testing the temporal behavior no specialized methods and tools exist. Therefore in most cases, test cases have to be defined manually.

A promising approach to automate test case design is the Evolutionary Test. It could be applied to testing the temporal behavior of systems as well as to structural testing. Evolutionary testing uses metaheuristic search techniques like evolutionary algorithms and simulated annealing for the generation of test cases. The input domain of the system under test represents the search space in which test data fulfilling the test objectives under consideration are searched for. The Evolutionary Test is generally applicable since it adapts itself to the system under test.

For testing temporal behavior of systems evolutionary testing searches for input situation with the longest or shortest execution times. First of all random test data are generated with which the system is to be executed. The execution times measured for each test datum evaluate the suitability of the test (fitness evaluation).

Test data with long or short execution times are selected (depending on the search for the worst case or best case execution time) and combined in order to obtain test data with even longer or shorter execution times (recombination). Following natural processes, random changes are carried out (mutation). By adding these generated test data to the already existing data a new test run is started. The test is terminated if an error in the temporal behavior is detected or a specified termination criterion has been reached. If a violation of the system's predetermined temporal limits has been detected, the test was successful and the system needs to be corrected (Wegener et al. 1997).

To automate structural testing each program structure represents a test objective for which a test datum is searched for, e.g. to achieve full branch coverage a test datum has to be found for each single branch. To guide the search to program areas which have not been executed so far the fitness functions are based on the branch predicates of the system under test (Jones et al. 1998). A test control manages all the test objectives, starts an optimization for each objective, calculates the fitness values for the generated test data on the basis of the executed program structures, and defines an efficient schedule for the testing of all the objectives. Our test environment supports among others statement testing, branch testing, condition testing and path testing. Principally, the test is terminated when all the test objectives have been considered during the test. The coverage reached and the corresponding test data are presented to the tester.

Because of the complete automation of evolutionary tests the system can be tested with a number of different input situation. Most often more than several thousand test data sets are generated and executed within only a few minutes. Prerequisites for the application of evolutionary tests are extremely few. Only an interface specification of the system under test is needed in order to guarantee the generation of valid input values.

The application of evolutionary tests in several case studies has proved successful and first industrial applications within the field of engine electronics yielded very good results. Effectiveness and efficiency of the test process can be clearly improved by Evolutionary Tests. Evolutionary Tests thus contribute to quality improvement and to the reduction of development costs. The application scope of Evolutionary Tests goes further than the work described within this paper. Additional application fields are for instance safety and robustness tests.

About the Author

Harmen Sthamer has a degree in Electronics and Communication from Polytechnic of Hannover, Germany. He has a MSc in Electronic Production Engineering and a Ph.D. in Software Technology from the University of Glamorgan, GB. Currently he is working as a scientist in the Software-Technology Laboratory of DaimlerChrysler, Research and Technology. He is currently working on systematic and evolutionary software testing methods for the verification of software-based systems. Harmen Sthamer is author or co-author of several papers and has presented on national and international conferences, e.g. IEE Conferences on

Genetic Algorithms. He is a member of the Seminal Network, UK.

Joachim Wegener has a degree in Computer Science from Technical University Berlin, Germany. He is manager of Adaptive Technologies in Software Engineering at DaimlerChrysler, Research and Technology. He was involved in the development of the classification-tree editor CTE and the test system TESSY. He is currently working on the design of software development processes for Mercedes-Benz as well as on systematic and evolutionary software testing methods for the verification of embedded systems. He is a member of SAE International, the Seminal Network and the German Computer Society Special Interest Group on Testing, Analysis and Verification.

14th International Internet & Software Quality Week 2001

Evolutionary Testing of Embedded Systems

Harmen Sthamer, André Baresel and Joachim Wegener
DaimlerChrysler AG, Research and Technology, Alt-Moabit 96a, D-10559 Berlin, Germany
Harmen.Sthamer@DaimlerChrysler.com
Andre.Baresel@DaimlerChrysler.com
Joachim.Wegener@DaimlerChrysler.com

Abstract

The development of embedded systems is an essential industrial activity. More than 90 % of all produced electronic components are used in embedded systems. Testing of embedded systems is considerably more complex than testing of conventional software systems. This is due on the one hand to the technical features of embedded systems, and on the other hand to the special requirements made on these kinds of systems: embedded systems usually have to fulfill functional as well as temporal requirements. Very often embedded systems are safety-relevant. In addition due to high costs resulting from errors occurring during the operation of embedded systems, high quality requirements apply.

Dynamic testing is the most important method for testing such quality requirements. However, test case design is difficult to automate, therefore, most test cases have to be defined manually. A promising approach to automate test case design is the Evolutionary Test. It can be applied to testing the temporal behavior of systems, to structural testing as well as to safety testing.

Effectiveness and efficiency of the test process can be clearly improved by Evolutionary Tests. This has been successfully proved in several case studies. Evolutionary Tests thus contribute to quality improvement as well as to the reduction of development costs.

0 Introduction

Testing is the most important quality assurance measure for embedded systems. It typically consumes 50 % of the overall development effort and budget. Systematic test case design is essential to a good test quality because it defines the type and scope of the test. For most test objectives, test case design is difficult to automate:

- for functional testing the generation of test cases is usually impossible because no formal specifications are applied in industrial practice,
- structural testing is also difficult to automate due to the limits of symbolic executions,
- for testing the temporal behavior of systems no specialized methods and tools exist, and also
- for testing safety constraints a generation of test cases is generally impossible.

Therefore, test cases have to be defined manually.

To increase the effectiveness and efficiency of the test and thus reduce the overall development costs for embedded systems, we require a test that is systematic and extensively automatable. While functional test case design can be automated to a large extent using new tools such as the CTE XL [Lehmann and Wegener, 2000], evolutionary testing [Wegener and Grochtmann, 1998] is a promising approach to entirely automate test case design for the aspects mentioned above. The Evolutionary Test can be applied to testing the temporal behavior of systems, it can be used to generate test cases for structural testing, and it enables the automation of safety testing. For

evolutionary testing the test case design is transformed into an optimization problem that in turn is solved with meta-heuristic search techniques, such as evolutionary algorithms and simulated annealing. The input domain of the system under test represents the search space in which test data fulfilling the test objectives under consideration is searched for. The Evolutionary Test is applicable in general because it adapts itself to the system under test.

The first chapter introduces the basic principles of the Evolutionary Test. The second chapter discusses the use of structural tests. The following two chapters illustrate the Evolutionary Test of the temporal behavior and the test of safety properties. The paper concludes with a summary of the most important results and an outlook on future work.

1 Evolutionary Testing

Evolutionary testing is characterized by the use of meta-heuristic search methods for test case generation. To achieve this the considered test aim is transformed into an optimization problem. The input domain of the test object forms the search space in which one searches for test data that fulfils the respective test aim. Due to the non-linearity of software (if-statements, loops etc.) the conversion of test problems to optimization tasks mostly results in complex, discontinuous, and non-linear search spaces. Neighborhood search methods like hill climbing are not suitable in such cases. Therefore, meta-heuristic search methods are employed, e.g. evolutionary algorithms, simulated annealing, or tabu search. In our work, evolutionary algorithms are used to generate test data because their robustness and suitability for the solution of different test tasks has already been proven in previous work, e.g. [Jones et al., 1998] and [Wegener et al., 1999].

1.1 A Brief Introduction to Evolutionary Algorithms

Evolutionary algorithms represent a class of adaptive search techniques and procedures based on the processes of natural genetics and Darwin's theory of biological evolution. They are characterized by an iterative procedure and work parallel on a number of potential solutions for a population of individuals. Permissible solution values for the variables of the optimization problem are encoded in each individual.

The fundamental concept of evolutionary algorithms is to evolve successive generations of increasingly better combinations of those parameters that significantly affect the overall performance of a design. Starting with a selection of good individuals, the evolutionary algorithm tries to achieve the optimum solution by random exchange of information between increasingly fit samples (recombination) and introduction of a probability of independent random change (mutation). The adaptation of the evolutionary algorithm is achieved by selection and reinsertion procedures based on fitness. Selection procedures control which individuals are selected for reproduction, depending on the individuals' fitness values. The reinsertion strategy determines how many and which individuals are taken from the parent and the offspring population to form the next generation.

The fitness value is a numerical value that expresses the performance of an individual with regard to the current optimum, so that different individuals can be compared. The notion of fitness is fundamental to the application of evolutionary algorithms; the degree of success in using them may depend critically on the definition of a fitness that changes neither too rapidly nor too slowly with the design parameters. The fitness function must guarantee that individuals can be differentiated according to their suitability for solving the optimization problem.

Fig. 1 provides an overview of a typical procedure for evolutionary algorithms. First, a population of guesses on the solution of a problem is initialized, usually at random. Each individual within the population is evaluated by calculating its fitness. This will usually result in a spread of

solutions ranging in fitness from very poor to good. The remainder of the algorithm is iterated until the optimum is achieved, or another stopping condition is fulfilled. Pairs of individuals are selected from the population according to the pre-defined selection strategy, and combined in some way to produce a new guess analogously to biological reproduction.

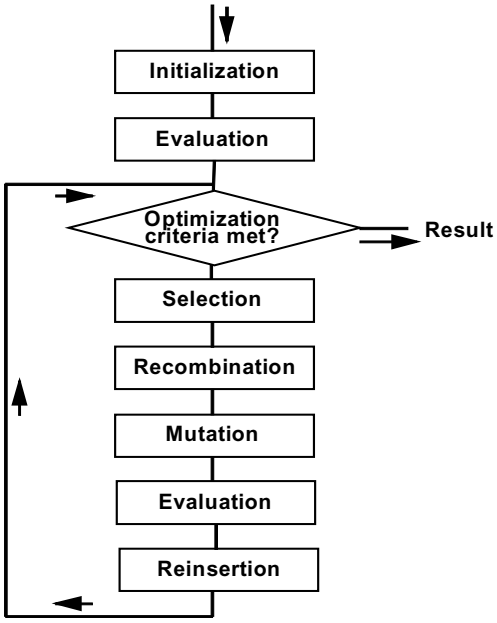


Figure 1: Evolutionary Algorithms

Combinations of algorithms are many and varied. Additionally, mutation is applied. The new individuals are evaluated for their fitness, and survivors into the next generation are chosen from parents and offspring, often according to fitness. It is important, however, to maintain diversity in the population to prevent premature convergence to a sub-optimal solution.

1.2 Application to Software Testing

In order to automate software tests with the aid of evolutionary algorithms, the test aim must itself be transformed into an optimization task. For this, a numeric representation of the test aim is necessary, from which a suitable fitness function for the evaluation of the generated test data can be derived. Depending on which test aim is pursued, different fitness functions emerge for test data evaluation. If an appropriate fitness function can be defined, then the Evolutionary Test proceeds as follows.

The initial population is usually generated at random. In principle, if test data has been obtained by a previous systematic test, this could also be used as initial population [Wegener et al., 1996]. The Evolutionary Test could thus benefit from the tester's knowledge of the system under test. Each individual of the population represents a test datum with which the test object is executed. For each test datum the execution is monitored and the fitness value is determined for the corresponding individual. Next, population members are selected with regard to their fitness and subjected to combination and mutation processes to generate new offspring. It is important to ensure that the test data generated is in the input domain of the test object. Offspring individuals are then also evaluated by executing the corresponding test data. Combining offspring and parent individuals, according to the survival procedures laid down, forms a new population. From here on, this process repeats itself, starting with selection, until the test objective is fulfilled or another given stopping condition is reached (compare Fig. 2).

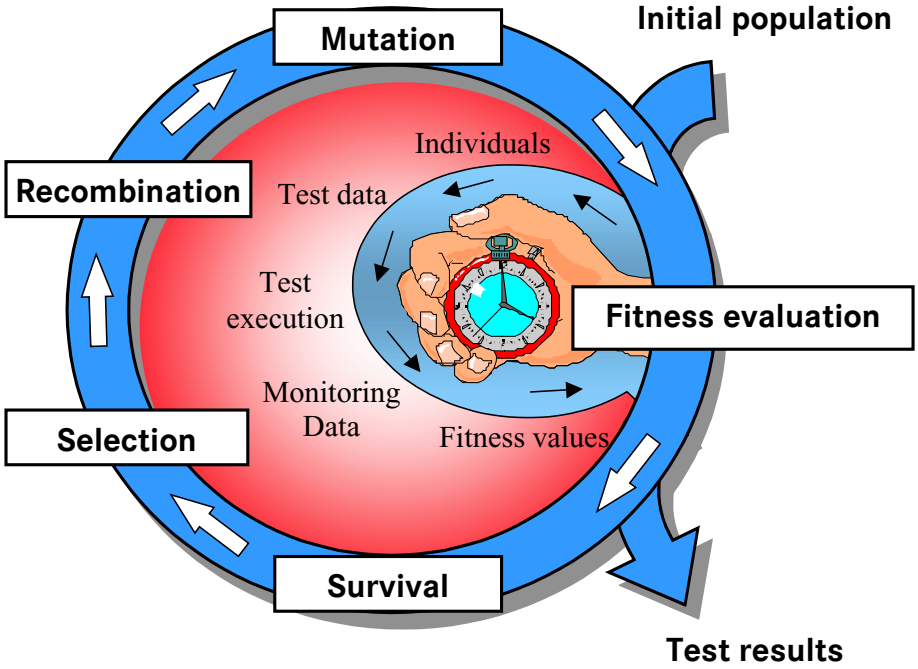


Figure 2: Evolutionary Test

2 Test Case Generation for Structural Testing

Structural testing is widespread in industrial practice and stipulated in many software-development standards. Common examples are statement, branch, and condition testing. The aim of applying evolutionary testing to structural testing is the generation of a quantity of test data, leading to the highest possible coverage of the selected structural test criterion.

Structural testing methods can be divided into four categories, depending on the control-flow graph and the required purpose of the test:

- node-oriented methods,
- path-oriented methods,
- node-path-oriented methods, and
- node-node-oriented methods.

Node-oriented methods require the execution of specific nodes in the control-flow graph. Statement testing and condition testing are the best known methods that fall into this category. Path-oriented methods require the execution of certain paths in the control-flow graph, e.g. path testing. Node-path-oriented methods require the achievement of a specific node and from this node the execution of a specific path through the control-flow graph. The branch test is the simplest example for node-path-oriented methods. LCSAJ (linear code sequence and jump) also belongs to the group of node-path-oriented methods. Node-node-oriented methods require the execution of several nodes of the control-flow graph in a pre-determined sequence without specifying a concrete path. The data-flow oriented methods all-defs, all-defuse-chains, as well as all-uses, fit into this category.

In order to apply evolutionary testing to the automation of structural testing, the test is split up into partial aims. The identification of the partial aims is based on the control-flow graph of the program under test. Each partial aim represents a program structure that needs to be executed to achieve full coverage, e.g. a statement, a branch, or a condition with its logical values. For

each partial aim an individual fitness function is formulated and a separate optimization is performed to search for a test datum executing the partial aim. The set of test data found for the partial aims then serves as the test data set for the coverage of the structure test criterion.

2.1 Fitness Functions

The fitness function definitions for the partial aims differ for the four categories of structural testing methods.

For node-oriented methods, the fitness functions of the partial aims are made up of two components: the distance and the approximation level. The distance specifies for a branching node how far away an individual is from executing the branching conditions in the desired manner (compare [Sthamer, 1996], [Jones et al., 1998], and [Tracey et al., 1998]). For example, if a branching condition $x==y$ needs to be evaluated as *True*, then the fitness function may be defined as $|x-y|$ (provided that the fitness values are minimized during the optimization) or as hamming distance. The approximation level supplies a figure for an individual that gives the number of branching nodes lying between the nodes covered by the individual and the target node ([Wegener et al., 2001], and [Baresel, 2000]). For condition tests the fitness evaluation needs to be slightly extended. The evaluation of the atomic predicates in the target nodes has to be included. The evaluation of the atomic predicates takes place in the same way as for the distance calculations in the branching conditions. For compound predicates the single distances are added and normalized.

Establishing the fitness function for path-oriented testing methods is much simpler than for node-oriented methods because the execution of a certain path through the control-flow graph forms the partial aim for the Evolutionary Test. The program path covered by an individual is compared with the program path specified as a partial aim. Thereby, the more nodes match, the higher is the fitness an individual can obtain. The fitness evaluation is supplemented by the calculation of the distances to the target path in the branching nodes in which the program path covered by the individual deviates from the target path.

The partial aims for node-path-oriented structural criteria comprise two requirements that need to be included in the evaluation of the generated individuals. The attainment of a specific node is required on the one hand, and on the other hand a path that begins with this node has to be covered. Accordingly, the fitness evaluation of the individuals has to represent both these components. The fitness function can be based on the fitness functions for node-oriented and path-oriented methods. Fitness calculations for individuals who do not reach the target node are carried out in the same manner as for the node-oriented methods. For individuals who reach the target node the mentioned fitness calculations for path-oriented methods are additionally applied in order to guide the search into the direction of the desired path.

Fitness calculations for node-node-oriented methods also take place in two stages. After the execution of the first target node, the second target node has to be covered, without a path specified through the control-flow graph. The approximation of an individual to the first target node can be evaluated in the same manner as for node-oriented methods. For all individuals executing the first target node an approximation to the second node is added. This is also calculated using the fitness function for node-oriented methods.

A detailed definition of the fitness functions can be found in [Wegener et al., 2001] and [Baresel, 2000].

2.2 Test Results

The Evolutionary Test has already been applied in various tests of real-world examples for automatic generation of test data with excellent results. For most test objects a complete coverage was achieved. Table 1 shows a selection of examined test objects from different application fields with their characteristics. One branch in the Netflow() function is infeasible. This leads to the highest possible coverage of 99.3%. For all the functions mentioned evolutionary testing performed notably better than random testing.

Test object	Lines of code	Number of branches	Maximum nesting level	Branch coverage achieved (%)
Atof()	69	57	8	100
Is_line_covered_by_rectangle()	94	24	2	100
Is_point_located_in_rectangle()	7	5	1	100
Search_field()	600	37	3	100
Netflow()	164	153	5	99,3
Complex_Flow()	46	41	4	100
Classify_Triangle()	38	38	7	100

Table 1: Complexity Measures and Branch Coverage Reached for Different Test Objects

3 Test Case Generation for Temporal Behavior Testing

Most embedded systems are subject to temporal requirements. This is due to reasons of operational comfort, e.g. short reaction times of the system to user commands, or due to requirements of technical processes that are controlled by the system. Therefore, embedded systems have to be thoroughly tested not only with regard to their functional behavior, but also in order to detect existing deficiencies in temporal behavior.

Existing test methods are unsuitable for the examination of temporal correctness. Even for an experienced tester it is virtually impossible to find the most important input situations relevant for a thorough examination of temporal behavior by analyzing and testing complex systems manually. However, evolutionary testing has already proved to be a promising approach for testing the temporal behavior of real-time and embedded systems ([Grochtmann and Wegener, 1998], [Mueller and Wegener, 1998], [Puschner and Nossal, 1998], [Wegener and Grochtmann, 1998] and [Gross et al., 2000]). When testing the temporal behavior of systems the objective is to check whether input situations exist for which the system violates its specified timing constraints. Usually, a violation occurs because outputs are produced too early or their computation takes too long. The task of the tester and therefore of the Evolutionary Test is to find input situations with especially long or short execution times in order to check whether a temporal error can be produced.

When using evolutionary testing for determining the shortest and longest execution times of test objects, the execution time is measured for every test datum. The fitness evaluation of the generated individuals is based on the execution times measured for the corresponding test data. If one searches for long execution times, individuals with long execution times obtain high fitness values. Conversely, when searching for short execution times, individuals with short execution times obtain high fitness values. Individuals with long or short execution times are selected depending on the objective of the test and combined in order to obtain test data with even longer

or shorter execution times. The test is terminated if an error in the temporal behavior is detected or a specified termination criterion has been reached. If a violation of the system's predetermined temporal limits has been detected, the test was successful and the system has to be corrected. Evolutionary testing enables a fully automated search for extreme execution times. The test especially benefits from the fact that test evaluation concerning temporal behavior is usually trivial. Contrary to logical behavior, the same timing constraints apply to large numbers of input situations.

3.1 Test Results

Previous work has shown that evolutionary testing always achieved better results than random testing (e.g. [Wegener et al., 1997] and [Wegener and Grochtmann, 1998]). The comparison with static analyses has also confirmed that the extreme execution times determined by the Evolutionary Test represent realistic estimations of the longest and shortest execution times [Mueller and Wegener, 1998]. Compared to systematic developer tests, the Evolutionary Test has also attained convincing results, as the following results illustrate. The results were achieved during the first application of evolutionary testing for the testing phase of a new engine control system for six- and eight-cylinder blocks.

The engine control system contains several tasks that have to fulfill timing constraints. Each task is a test object and has been tested for its worst-case execution time by the developers using systematic testing. The test cases for testing the temporal behavior, defined by the developers, are based on the functional specification of the system as well as on the internal structures of the tasks. For each task the developer tests achieved full branch coverage. Evolutionary testing was used to verify these results. The tests were performed on the target processor later used in the vehicles. The execution times have been determined using hardware timers of the target environment.

The results for six of the tasks (M1 to M6) are shown in Figure 3. The figure shows the longest execution times determined by the developers with systematic testing (DT) in comparison to the results achieved by evolutionary testing (ET).

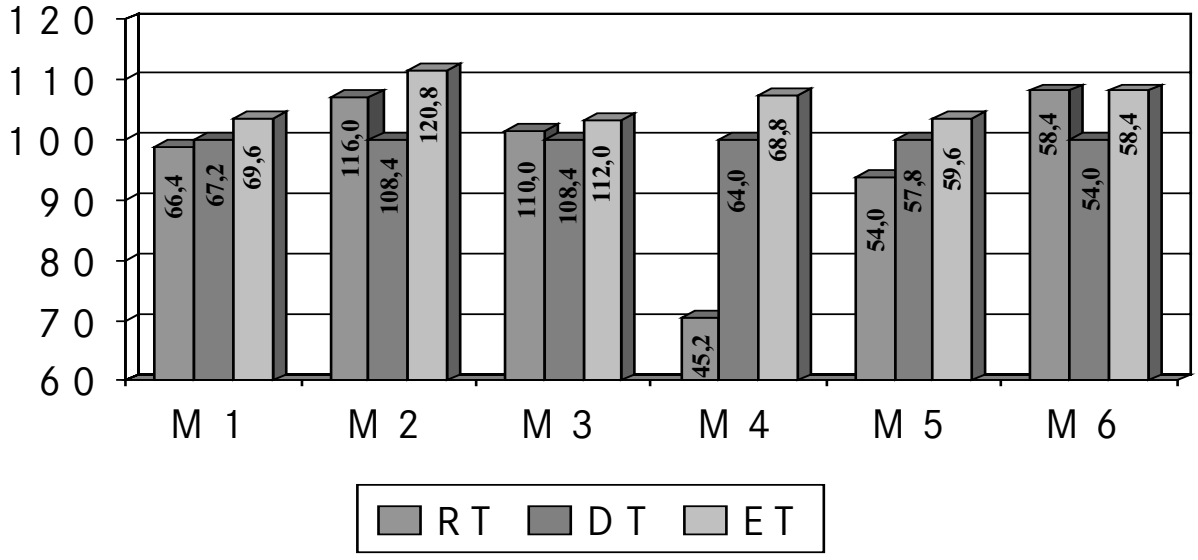


Figure 3: Results for the Engine Control System Tasks

Additionally, the results for random testing are shown (RT). The results of the developer tests are set to be 100 %. The execution times achieved, measured in µs, are shown directly in the bars. The size of the tasks varied from 39 LOC (lines of code) to 119 LOC, the number of input parameters from 9 to 32.

Comparison of the results shows that evolutionary testing found the longest execution times for all the given tasks among these three testing methods. The developer tests never reached the longest execution time. In three cases the results of the developer tests are even worse than those of the random test. For the other three tasks the results are better than those of the random test. The latter only finds the longest execution time for task M6. The longest execution time found by the random test in task M4 lies more than 35 % below the value determined by the Evolutionary Test, and 30 % below that of the developer tests.

The excellent performance of the Evolutionary Test in comparison to the developer tests shows the effectiveness of the Evolutionary Test, also in comparison to function-oriented and structure-oriented testing methods. The results are especially astonishing, because evolutionary testing treats the software as black boxes whereas the developers are familiar with function and structure of their system. An explanation might be the use of system calls of which the effects on the temporal behavior can only be rated with difficulty by the developers.

4 Test Case Generation for Safety Testing

Embedded systems are often also safety-relevant. Our work on the application of Evolutionary Tests for testing safety properties of embedded systems is just beginning. It will follow the example of [Tracey et al., 1998]. Within the context of safety analyses for embedded systems (e.g. fault-tree analysis, and software-hazard analysis) indispensable safety requirements for the system components are derived from such system behavior that has to be absolutely avoided. If a violation of the specified safety requirements is possible the system is not safe. Consequently, the aim of the test is to find input situations that lead to a violation of the safety requirements. If such an input situation can be found the system is not safe and has to be corrected.

The fitness evaluation when applying the Evolutionary Test to safety tests is similar to the fitness evaluation of structural testing. However, the fitness function is not based on the branch predicates of the program, but on the pre- and post-conditions that have been specified for the single components (e.g. [Tracey et al., 1998]). For example, if an output signal *speed* of a component is not allowed to become negative, the fitness values of the individuals can be set according to every produced output value for *speed.* Individuals who generate a small value for *speed* obtain a higher fitness value than individuals producing high values for *speed.* If the Evolutionary Test is able to find an individual who obtains a negative value for *speed*, it is proof of a violation of the safety requirements.

In order to achieve a complete automation of the safety test, we are currently working on an integration of the Evolutionary Test with Time Partition Testing [Lehmann, 2000] for the system and integration test of embedded systems. Another aspect of our work is the integration with the test environment MTest [Conrad et al., 1999] for the unit test of the software modules of control systems.

5 Summary and Future Work

The thorough test of embedded systems includes a number of demanding testing tasks. These are difficult to master on the basis of conventional function-oriented and structure-oriented testing methods. Moreover, automation is also problematic. This includes the generation of test

cases for the coverage of different structural testing criteria, the test of temporal behavior, and the test of the compliance with the specified safety requirements for a safety-relevant system.

The Evolutionary Test is a promising approach to entirely automate complex testing tasks. It enables the complete automation of test case design for structural testing, the testing of temporal behavior with regard to its exceeding or falling below the specified timing constraints, and the testing of safety properties. Evolutionary testing has already produced very good results in all these three areas of application. Due to the complete automation of the Evolutionary Test the system can be tested with a large number of different input situation, both for testing the temporal behavior and for safety tests. In most cases, more than several thousand test data sets are generated and executed within only a few minutes. If no violations of the specified constraints can be found the confidence in the correct functioning of the system will be increased to a large extent. The prerequisites for the application of Evolutionary Tests are extremely few. Only an interface specification of the system under test is needed to guarantee the generation of valid input values. For structural testing the source code of the test object is also required.

The application of the Evolutionary Test has been successfully proved in several case studies. First industrial applications within the field of engine electronics yielded very good results. Effectiveness and efficiency of the test process can be clearly improved by Evolutionary Tests. Evolutionary Tests thus contribute to quality improvement and to the reduction of development costs. The application scope of Evolutionary Tests goes further than the work described within this paper. Additional application fields are, for instance, functional [Jones et al., 1995] and robustness tests [Schultz et al., 1993].

Current work on evolutionary structural tests concentrates on the assessment of the testability of programs on the basis of statically determinable software metrics. By using appropriate information it is possible to select the best suitable evolutionary algorithms for the test, and also to start program transformations that improve the testability.

In future, it is also intended to examine more closely the combination of evolutionary testing with static analyses for testing the temporal behavior. By combining both approaches, the area in which one finds the extreme execution time of the system can be closely defined, e.g. static analyses give an upper estimate for the maximum execution time and testing gives a lower estimate for the maximum execution time. This means, developers of real-time systems would gain an efficient tool to rate exactly the minimum and maximum execution times for their systems.

In addition we are looking at investigating the application of evolutionary structural tests for testing the temporal behavior of systems. The idea is to pre-determine program paths as test aim for the evolutionary structural test which have been identified as worst-case execution time paths by means of static analyses (e.g. [Mueller, 1997], [Puschner and Vrchoticky, 1997]). If a test datum can be found that executes the path we can be sure that this is the longest execution time possible to obtain. Due to pessimistic assumptions in static analyses the path will usually not be executable. However, the pre-definition of these paths can lead to a very interesting concentration on paths with long execution times.

6 References

Baresel, A., Automatisierung von Strukturtests mit evolutionären Algorithmen (Automation of Structural Testing using Evolutionary Algorithms), Diploma Thesis, Humboldt University, Berlin, Germany, 2000.

Conrad, M., Dörr, H., Fey, I., Yap, A., Model-based Generation and Structured Representation of Test Scenarios, Proceedings of the Workshop on Software-Embedded Systems Testing, Gaithersburg, Maryland, USA, 1999.

Grochtmann, M.; Wegener, J., Evolutionary Testing of Temporal Correctness. Proceedings of the 2nd International Software Quality, Week Europe (QWE' 1998), Brussels, Belgium, November 1998.

Gross, H.-G., Jones, B. und Eyres, D., Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems, IEE Proc.-Softw., Vol. 147, No. 2, April 2000, pp. 25 – 30.

Jones, B. F., Eyres, D. E. and Sthamer, H. - H., A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing, The Computer Journal, Vol. 41, No. 2, 1998.

Jones, B.F., Sthamer, H.- -H., Yang, X., Eyres, D.E., The Automatic Generation of Software Test Data Sets using Adaptive Search Techniques. Proceedings of Software Quality Management '95, Seville, Spain, 1995, pp. 435 - - 444.

Lehmann, E., Time Partition Testing: A Method for Testing Dynamic Functional Behaviour, Proceedings of TEST2000, London, Great Britain, May 2000.

Lehmann, E.; Wegener, J., Test Case Design by Means of the CTE XL, Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Kopenhagen, Denmark, December 2000.

Mueller, F., Generalizing Timing Predictions to Set-Associative Caches, Proc. EuroMicro Workshop on Real-Time Systems, pp . 64-71, Jun 1997.

Mueller, F.; Wegener, J., A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints, Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium, Denver, USA, June 1998.

Puschner, P. and Nossal, R.: Testing the Results of Static Worst-Case Execution-Time Analysis, Proc. 19th Real-Time Systems Symposium, pp. 134-143, 1998.

Puschner, P. and Vrchoticky, A., Problems in Static Worst-Case Execution Time Analysis, Proceedingsof the 9th ITG/GI-Conference Measurement, Modeling and Evaluation of Computational and Communication Systems, 1997, pp. 18-25.

Schultz, A. C., Grefenstette, J. J. and De Jong, K. A., Test and Evaluation by Genetic Algorithms, IEEE Expert 8(5), 1993, pp. 9 – 14.

Sthamer, H.-H., The Automatic Generation of Software Test Data Using Genetic Algorithms, PhD Thesis, University of Glamorgan, Pontyprid, Wales, Great Britain, 1996.

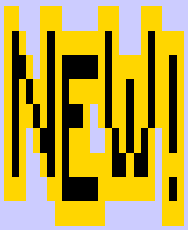
Tracey, N., Clark, J., Mander, K. and McDermid, J., An Automated Framework for Structural Test-Data Generation. Proceedings of the 13th IEEE Conference on Automated Software Engineering, Hawaii, USA 1998.

Wegener, J., Grochtmann, M., Verifying Timing Constraints of Real-Time Systems by means of Evolutionary Testing. Real-Time, Systems, vol. 15, no. 3, Kluwer Academic Publishers, 1998, pp. 275 - 298.

Wegener, J.; Baresel, A.; Sthamer, H., Evolutionary Test Environment for Automatic Structural Testing, submitted to to the Special Issue of Information and Software Technology devoted to the Application of Metaheuristic Algorithms to Problems in Software Engineering 2001.

Wegener, J.; Grochtmann, M.; Jones, B., Testing Temporal Correctness of Real-Time Systems by Means of Genetic Algorithms, Proceedings of the 10th International Software Quality Week (QW '97), San Francisco, USA, May 1997.

Wegener, J.; Pohlheim, H.; Sthamer, H., Testing the Temporal Behavior of Real-Time Tasks using Extended Evolutionary Algorithms, Proceedings of the 7th European Conference on Software Testing, Analysis and Review (EuroSTAR '1999), Barcelona, Spain, November 1999.



QW2001 Paper 9A2

Mr. Hung Q. Nguyen
(LogiGear Technology)

The Design and Implementation of a Flexible, Reusable and Maintainable Automation Framework

Key Points

- How to design a reusable test automation framework
- How to prepare for the development of an automation project
- Developmemt mistakes to avoid

Presentation Abstract

This presentation discusses a case study of how to create an automation framework designed to be a product-independent automated testing solution. "Product-independent" means that the automation solution can be easily adapted to do various tasks and accommodate changes to the application under test (AUT) without having to redo the framework. It tells the story of how one of the test automation teams at LogiGear went about the process of requirements analysis, research, designing, prototyping and implementation. This session offers a results analysis as a measurement of success. This is an experience-based technical study that can help you prepare for your next automation endeavor.

About the Author

Hung Q. Nguyen is president and CEO of LogiGear Corporation, a Silicon Valley software testing company whose mission is to help software development >organizations deliver the highest quality products possible while juggling >limited resources and schedule constraints. LogiGear offers many value-added services including >application testing, automated testing and web load/performance testing for e-business and >consumer applications. Nguyen’s company produces and markets TRACKGEAR™, a web-based defect >tracking system. LogiGear also specializes in Web application, hand-held communication device and consumer electronic product testing, and offers the software development community a >comprehensive “Practical Software Testing Training Series. ” In the past two decades, Nguyen has held leadership roles in business development, engineering, quality assurance, testing, product development, and information technology. Nguyen is the author of Testing Applications on ! the Web (Wiley) and co-author of the best-selling book, Testing Computer Software (Wiley). He also develops and teaches software testing courses for UC Berkeley and UC Santa Cruz Extension, and for LogiGear. He holds a Bachelor of Science in Quality Assurance from Cogswell Polytechnical College, and is an ASQ-Certified Quality Engineer and active senior.

The Design and Implementation of a Flexible, Reusable, and Maintainable Automation Framework

Hung Q. Nguyen
LogiGear® Corporation

Objectives

- Prepare you to build a successful and reusable automation architecture
- Share the keyword approach to creating an automation framework
- Share lessons learned in implementing a flexible architecture

Background: The Evolution

- The early days
- Developing an automation framework
- The table-driven approach
- The keyword-driven approach

The Early Days

- Collect acceptance/regression test cases to be automated
- Record and script test cases
- Improve reusability
  - Parameterize hard-coded values
  - Separate data from code by moving variables to INCLUDE files
  - Create utility functions to be shared
- Train test specialists to run scripts

The Next Wave: Creating a Framework

- Work with test specialists to understand their testing needs
- Go beyond acceptance/regression tests-- Analyzing user-scenario test cases
- Recognize the difference between task-driven and object-driven test cases

Object-Driven vs. Task-Driven

- Object-Driven
  - Click User Name text box
  - Enter your_ID
  - Click Password text box
  - Enter your_password
  - Click Login button
- Task-Driven
  - Log in using
    - User Name = your_ID
    - Password = your_password

The Next Wave: Creating a Framework

- Pre-separating data and code
  - Start by defining functions to be written
  - Variablize data and keep variables in INCLUDE files
- Pair up a test specialist and an automation engineer to improve communication and to ensure that the framework design and implementation meet the test objectives
- Train test specialists to run test scripts

The Table-Driven Approach

- Take advantage of tester's familiarity with test case creation using tables and matrices
- Accommodate localization projects
- Recognize the importance of patterns in test cases
- Enable testers to catalog test cases with Excel spreadsheets
- Enable testers to specify expected results in spreadsheets

A Table-Driven Example

A Table-Driven Example

- **for (i=1; i<= iLastDataSet; i++)**
  - **Open the dialog box.**
  - **Use the data in DataSet[i] (The first set is 1 and the last set is 12) to set the values of Match Case, Match Whole Word and Find What controls.**
  - **Click Find Next.**
  - **Verify the results.**

	PROPERTY											
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12
Match case	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON
Match whole	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Find What	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"lowercase"	"lowercase"	"lowercase"	"lowercase"	"MixedCase"	"MixedCase"	"MixedCase"	"MixedCase"

The Need for Improvement

- Business issues
- People and process issues
- Technology issues

The Business Issues

- Need to expand our service offerings and share success through our test automation expertise
- Need to have a methodology for quick deployment of test automation
- Need to build a transferable architecture
- Need a better approach to test automation job costing

The Business Issues

- Need to deliver an automation program that is practical, explainable, and trainable
- Need to be more cost effective through reusability across projects
- Need to make technology a viable business solution
- Need a tangible approach to deciding between manual testing and automated testing

The People and Process Issues

- Need to standardize test methodology-- Enabling testers and automation engineers to collaborate
- Enable testers to better specify their needs and automation engineers to better serve those needs
- Need to integrate test automation as part of the process of software testing

The People and Process Issues

- Need testers to focus on test case design, and automation engineers to focus on driver script writing
- Make data more visible and understandable from the human perspective
- Need to incorporate test case design techniques with Excel, which test specialists are already familiar

The Technology Issues

- Need to build an architecture that's tool independent as well as application independent
- Need to improve the ability to share code across projects and tools
- Need to separate control of task variables, input variables, and code
- Need to integrate action keyword into the existing data-driven model

The Technology Issues

- Want to focus the development and maintenance of test scripts on the navigation of the application under test
- Need to take advantage of Excel features to automate test case and test data creation
- Need to incorporate test case design techniques using Excel, any database, XML, or other viable data service solutions

The Integrated Solution

Integrated Testing Solutions =
[Test Specialist's Domain Expertise] +
[Manual Testing] +
[Automated Testing: Reusable Framework & Application Specific Scripts]

The Development Process

- Research possible solutions and evaluate options
- Develop requirements
- Develop the architecture
- Build the framework
- Test the framework
- Develop documentation
- Deploy the framework on a real project
- Measure performance and refine the design

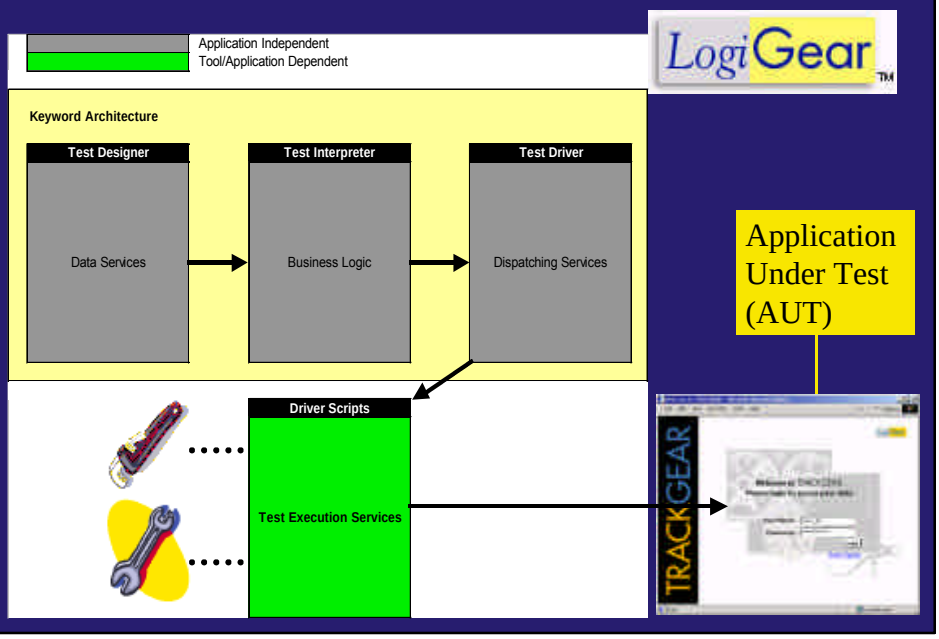
Research Possible Solutions

- Learn from past experience
- Discuss possibilities with software developers
- Talk to friends
- Read books
  - Recommend "*Software Test Automation*" by Graham and Fewster, 1999, Addison-Wesley
- Use the Internet
  - Recommend www.QACity.com, the Automated Testing page

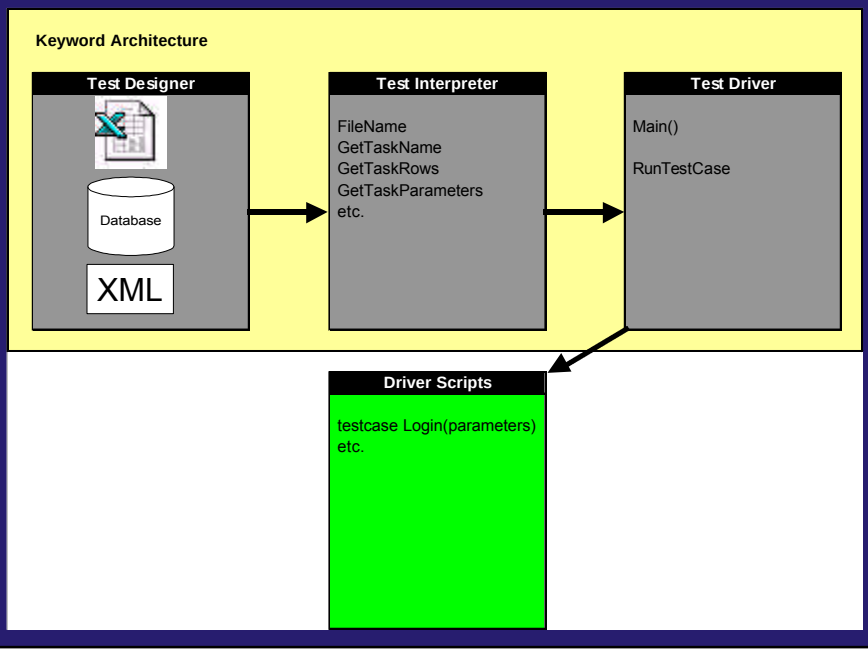
The Requirements

- Clearly state the business, people/process, and technology objectives
- Set expectations through well defined deliverables (e.g., requirement and design documents, code modules, whitepapers, training materials, etc.)
- Clearly define ways to measure success (e.g., quality of the design and code, budget, schedule, customer approval upon deployment, etc.)

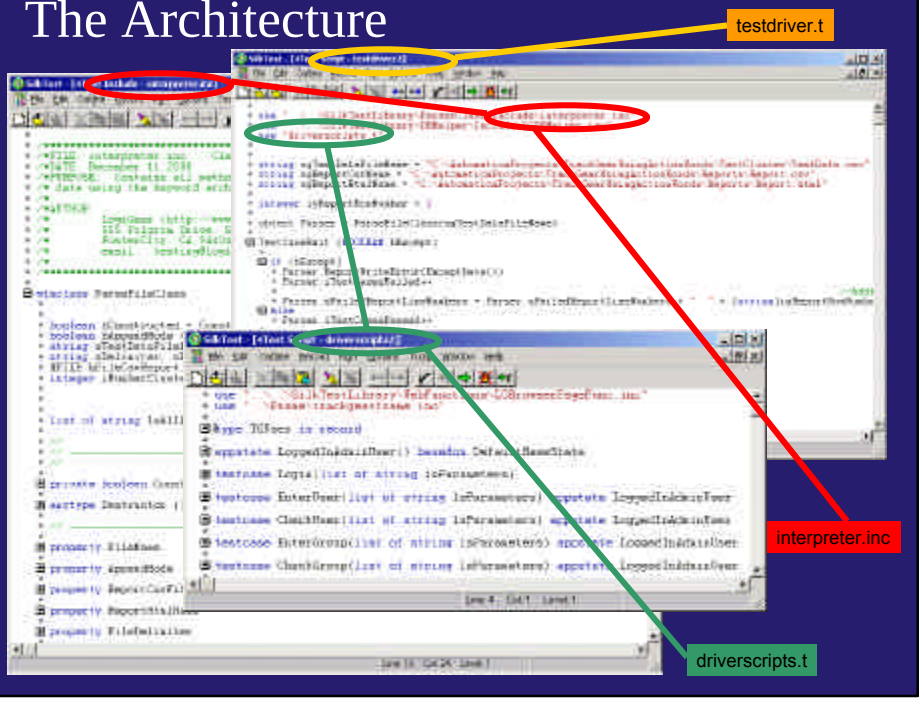
The Architecture



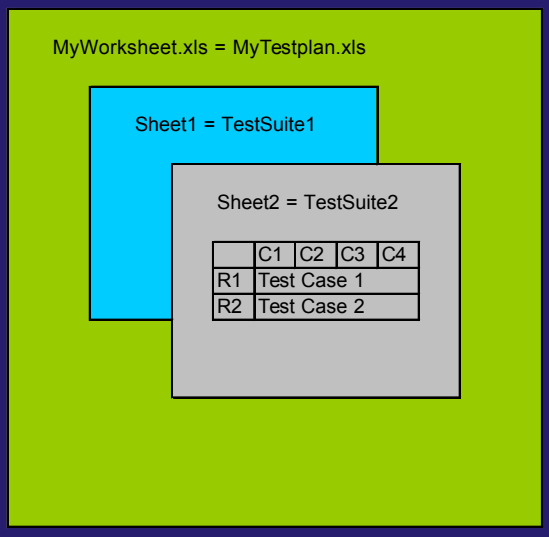
The Architecture



The Architecture



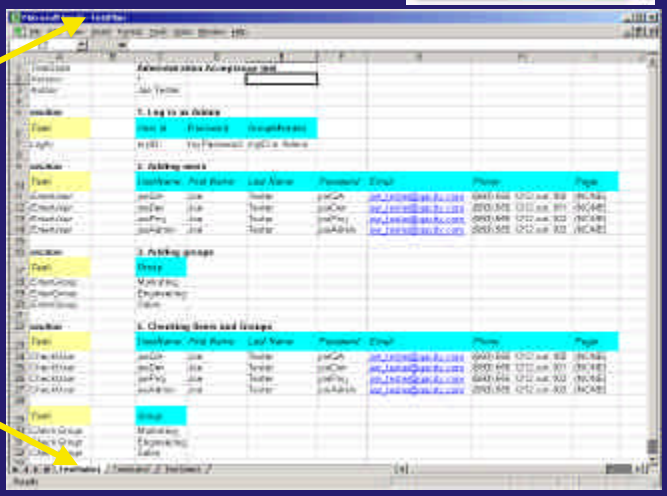
The Architecture: The Test Designer



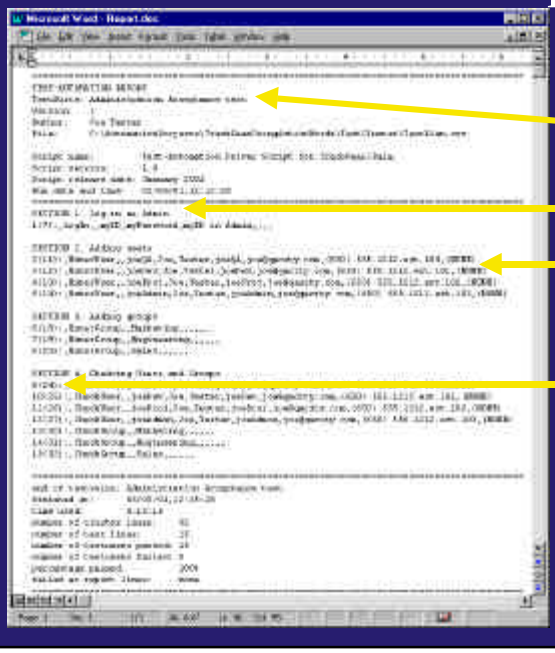
The Architecture: The Test Designer

Test Plan

Test Suite



The Architecture: The Report



Test suite

Test section

Test case

Test line & equivalent spreadsheet row number

Building the Framework

- Prototype the components
- Implement the Test Designer
- Implement the Test Interpreter
- Implement AUT specific Test Drivers
- Add the reporting function to the Test Interpreter
- Test, fix bugs, and write documentation

Lessons Learned

- Clear requirements help focus the team on the important issues.
- Leaving “Fill in the blank” sections in requirements is manageable.
- Spending time on designing and prototyping helps flush out design issues; making it more scaleable, and helping write more maintainable code.
- If the project is overly complex and the schedule is aggressive, you may need to scale back. Don’t forget to communicate changes in your plan.

Lessons Learned

- The necessary information is available! We need a way to find and analyze relevant information more quickly and effectively.
- Thoroughly research your options. Choose your designs wisely by taking business issues, people and process issues, and technology issues into consideration.
- Keep in mind that your solution might be used by one group, and maintained by another group.

Lessons Learned

- Your effort is a serious development project. Treat it as such: The key to success is good planning, scheduling and budgeting.
- Get feedback! How else can you learn?
- It won't be perfect! It's acceptable to learn from mistakes and refine the design as you go. Iteration and hard work make perfection.
- The keyword approach *works*!

Acknowledgment

Special thanks to Hans Buwalda for sharing his experience and vision on the action-word approach to creating test automation framework.

About Hung Q. Nguyen

Hung Q. Nguyen is Founder, President and CEO of LogiGear Corporation, a Silicon Valley software testing company whose mission is to help software development organizations deliver the highest quality products possible while juggling limited resources and schedule constraints. LogiGear offers many value-added services including application testing, automated testing and web load/performance testing for e-business and consumer applications. Nguyen's company produces and markets TRACKGEAR™, a web-based defect tracking system. LogiGear also specializes in Web application, hand-held communication device and consumer electronic product testing, and offers the software development community a comprehensive "Practical Software Testing Training Series." In the past two decades, Nguyen has held leadership roles in business development, engineering, quality assurance, testing, product development, and information technology. Nguyen is the author of Testing Applications on the Web (Wiley) and co-author of the best-selling book, Testing Computer Software (Wiley). He also develops and teaches software testing courses for UC Berkeley and UC Santa Cruz Extension, and for LogiGear. He holds a Bachelor of Science in Quality Assurance from Cogswell Polytechnical College, and is an ASQ-Certified Quality Engineer and active senior member of American Society for Quality.

About LogiGear® Corporation

LogiGear® Corporation is a full service software quality-engineering firm that provides testing expertise and resources to software development organizations. Some of our value-added services include application testing, automated testing, and web load/performance testing for e-business and consumer applications. LogiGear specializes in Web application, hand-held communication device, and consumer electronic product testing. LogiGear also produces and markets TRACKGEAR™, a Web-based defect-tracking solution, and offers QA Training through the Practical Software Testing Training Series.

www.logigear.com

QW2001 Paper 2W1

Mr. Adrian Cowderoy
(ProfessionalSpirit Ltd)

Quality in a Dotcom Startup -- Fact or Fiction?

Key Points

- Fighting the problems of speed, innocence and changes in objectives.
- Enhancing existing methods with ideas from elsewhere.
- Integrating quality and brand.

Presentation Abstract

This is the story of a crusade to build quality management into a start-up Internet company, from the beginning. It is a story of campaigns interrupted by deadlines, of specification techniques ruined by evolving markets, and testing interrupted by frequently changing business objectives.

It is also a story of people. Of the conflicting ideologies of engineering, creativity, community and business. And of the software engineers who received rude awakenings from each attempt to stabilise the development processes.

We came to recognise that the bigger the challenges facing quality engineers, the more ingenious we have to be to achieve our objectives.

It is not formal techniques that matter most during the early months of startup, but a number of practical guidelines. For instance:
1. Repeatedly emphasise that “quality comes first, and features come last”.
2. Define your brand values, and expand them into quantifiable quality measures.
3. Provide broadband internal communication on all kinds of information with the business and technology. Keep it to summary form or presentations.
4. Use checklists and conceptual models that help everyone see the big picture.
5. Use corporate quality procedures ONLY for critical areas like version control and planning.
6. Encourage each designer and developer to use the quality tools and techniques they know, and can apply quickly.
Especially, employ only experience people, or outsource to mature companies. (Talented people pick up the seedlings of ideas and turn them into a reality that is greater than we dared hope.)

The results are a steady introduction of quality practices, first by individuals and then by others copying them. Processes start to stabilise. We have even found that

our contractors are changing their practices to catch up with our requirements.

About the Author

Adrian Cowderoy is Managing Director of the Multimedia House of Quality Limited, a company which he established to promote quality-improvement methods for the production of websites and multimedia.

Mr Cowderoy was the General chair of ESCOM-SCOPE-99 and ESCOM-ENCRESS-98 conferences, and was Program chair for ESCOM 96 and 97 (The European Software Control and Metrics conference promotes leading-edge developments in industry and research, worldwide û see www.escom.co.uk). He is the METRICS-ESCOM Coordinator for IEEE METRICS 2001 and was on the Program committee of Metrics 98 and 99, European Quality Week 99 and COCOMO/SCM 96-99. In 1998 he was acting Conference Chair of the Electronics and Visual Arts conference in Gifu, Japan. He is a registered expert to the European Commission DGXIII.

He has provided consultancy and industrial training courses on quality management, risk management, and cost estimation to the aerospace and medical industries in the UK, Germany and Italy since 1995. He also lectures at Middlesex University (www.mdx.ac.uk) on e-commerce project management and managing Internet start-up's, and at City University, London (www.city.ac.uk), on project management for systems development.

Mr Cowderoy was project manager and technical director of MultiSpace, a 14-month million-dollar initiative sponsored by the European Commission in which 12 European organizations explored the potential to apply quality-improvement methods to multimedia and website development projects. (See www.mmhq.co.uk/multispace and www.cordis.lu/esprit.)

He was a Research fellow at City University from 1990-1998, and a Research Associate at Imperial College from 1986-1989. He was also a quality consultant and software developer at International Computers Limited, UK, from 1980-1985, where he worked on operating and networking systems for mainframes and distributed systems.

His academic qualifications include an MSc in Management Science from Imperial College, University of London in 1986, and is a member of the Association of MBA's. He received a BSc in Physics with Engineering from Queen Mary College, University of London, in 1979.

Mr. Cowderoy has published and presented extensively on multimedia quality and software cost estimation. He was joint editor of Project Control for 2000 and Beyond (Elsevier, 1998), Project Control for Software Quality (Elsevier, 1999), and Project Control: The Human Factor (Elsevier, 2000).

quality in a dotcom startup - fact or fiction?

Adrian Cowderoy
ProfessionalSpirit Ltd (UK)

These are the opinions of the author.

They do not necessarily represent official policy of ProfessionalSpirit Limited, nor do they necessarily describe situations which may or may not have occurred within that company.

coming ...

4 realities

1 challenge

10 guidelines

2 conclusions

our reality

the company	Founded in January 2000 with private investment.
the business	Mega-exchange and portal.
the technology	Uses dynamic personalised website with real-time delivery.
the people	Software development with extensive quality management and sizing experience. Commercial, accounting, web design, marcomms.

cold reality

Too many dotcoms have failed.

Big corporations are taking the market.

Size and complexity is increasing.

dotcom is a bad word

Staff are difficult to find.

Quality requirements have increased.

Funds are tight, and erratic.

market reality

the demands . . .

Internet lifecycle is 2-3 months.

- User requirements changes.
- New competition appears.
- New technologies are introduced.
- Fashions change (every 6 months).

New business plan every 2-3 months.

. . . are impossible

Product development is longer.

3+ months for specification.

3-9 months for software development.

3+ months for market launch

The pressure for changes is tremendous and quality is at risk.

ideological reality

engineers . . .

Quality management needs detail and rigor.

. . . are different

Creatives, community-builders, commercial strategists, and market makers all require simple messages and broad discretion.

expertise reality

Everyone claims to be an "expert"

- at user profiling
- at usability design
- at marketing strategy
- at identifying market opportunities
- at setting release priorities.

too many chefs

Conflict.

Danger of making the wrong choice.

progress

4 realities

1 challenge

next 10 guidelines

1 conclusion

ten guidelines

principle We need guidelines, not laws.

In a year's time,
the Internet market will be different.
Guidelines change.

scope Left hand - 5 fingers for understanding.

Right hand - 5 finger for process.

user awareness (left hand, thumb)

Different users have vastly different needs.
- Usability design becomes critical.
- Marketing strategies become complex.

All employees need to know the users.

user profiling

Identify 4-7 types of user.

Define their characteristics using a checklist.

Our checklist covers 30 issues, including
- web and computer usage.
- user background (literacy, attitudes, etc).
- relationship to the business.

market message (left hand, forefinger)

Technology can be replicated.
Brand can not.

brand-driven goals

Define your brand values
and corporate style.

Provide quantitative measures
to create the brand contract:
- software quality measures
- service level agreements for staff
- content quality measures
- system performance measures

competitiveness (left hand, middle finger)

Competition on the Internet is fierce.

Market dynamics can be difficult to learn for non-marketing staff.

differentiating features

Maintain a list of differentiating features that demonstrate competitive advantages.

Check software functional specs, business process, etc all satisfy them.

Update the list every 3 months.

business realities (left hand, fourth finger)

Dotcoms come, Dotcoms go
… the failure worry the staff.

Business models change dramatically over 12-month periods
… causing major changes in direction.

be mindful of the past

Encourage staff to read widely.

Openly discuss feasibility of new ideas.

Beware of sudden fashion changes in business models.

technology awareness (left hand, small finger)

Web software quality is very demanding.

Maintaining websites is hard work.

The new technologies look exciting
… but they can bring big problems.

build new experiences

Use workshops to explore what is possible.

Trial the latest technology in prototypes.

Evaluate cost, time and risk.

ignorance (right hand, thumb)

Lack of awareness of what's happening
is a killer.

Nobody has time to read every document.

Everyone has areas of weaknesses.

1-page summaries

Assume nothing is known.

Describe only major points.

2 pages limit.

Help people write their first summary.

staffing (right hand, forefinger)

Staff focus on their specialties.

When in a hurry,
important things get missed.

the big picture

Use checklists to cover breadth.
Use leading-edge books and websites.

Involve everyone in broad discussions.

risk (right hand, middle finger)

Anything that can go wrong, will.

As the enterprise advances you encounter
many new commercial opportunities.
Each has its own new risks.

opportunity management

Use risk management techniques:
- Maintain a log of potential opportunities, and the risks that come from them.
- Identify the biggest risks.
- Plan strategies for handling them.
- Monitor progress.

urgency (right hand, fourth finger)

There is no time to learn technologies and no time to gain experience.

experience rules ok

Recruit experienced people for key roles
- for 33% more money
you gain 100% more effect

Find quality control and quality assurance methods your team knows,
and reinforce with ***just enough*** rigour.

process (right hand, little finger)

Quality assurance managers want it all:

- corporate procedure manuals, ISO 9001, extensive measurement, process maturity, and more.

But the business needs flexibility.

don't walk before you can run

Define corporate quality procedures ONLY for critical areas like version control and planning.

- "Keep It Simple, Stupid"

10 guidelines

left hand for understanding

- User profiling
- Differentiating features
- Brand-driven goals
- Be mindful of the past
- Build new experiences

right hand for process

- The big picture
- Opportunity management
- 1-page summaries
- Experience rules ok
- Don't walk before you can run

and finally

- 4 realities
- 1 challenge
- 10 guidelines

next

- 2 conclusions

conclusion, #1

start now

Start with the quality methods that suit the leading staff and contractors.

Then build on this.

conclusion, #2

the biggest enemy of luck . . .

. . . is ignorance

Keep counting fingers.

QW2001 Paper 2W2

Mr. Todd Hsueh
(IBM)

Innovative Web Test Process & Control Tool

Key Points

- Web Test Lessons Learned
- Development-to-Testing Trilogy
- Process Control for Quality with Speed

Presentation Abstract

The "Innovative Web Test Process & Control" is a three-fold process evolved through our Web development and testing projects in the Centers for IBM e-business Innovation : : Los Angeles. This process control mechanism is currently supported by a Lotus Notes "Web Test and Incident Tracking" Database and has been submitted to the IBM Intellectual Property Review Board for invention patent and/or disclosure. The three sub processes are:

- Unit/Integration Test (UIT) Checklist Review/Approval,
- Code Release Authorization Using Pre-test, and
- Incident Tracking.

By enforcing the "UIT Checklist" process, the Creative Team (Art Director, Content Strategist, Information Designer) and the Development Team (Solution Designer, HTML/JAVA/JSP Programmers) verify their work products against requirements and design and ensure that Unit/Integration Test is properly done before System Test starts.

By enforcing the "Code Release Authorization Using Pre-test" process, the Development Team and the Test Team coordinate and synchronize their work. The new code release is delivered to the System Test environment when the Test Team is ready to receive it. The "Pre-test" ensures that the new code release is properly built with quality and truly ready for System Test.

By "Incident Tracking", all project staff has a common repository for all issues, issue status, assignments, and resolution. An issue can range from a proposal planning action item to a System Testing problem. By different groupings (Lotus Notes database views), various statistics and management status reports are instantly available.

About the Author

Todd Hsueh is a Senior Information Technology (IT) Specialist in IBM Global Services organization with 23 years of data processing experience. Since 1978, Todd has worked for various companies as a programmer, Lead System Analyst, and Project Manager. His industry experience includes manufacturing, health, DP consulting, finance, insurance, travel, media and auto business. Todd is specialized in quality assurance and managing testing process of all types of applications ranging from mainframe, client/server, to Web applications. On most projects in the last ten years, Todd worked as the Testing Lead or QA Manager and has completed all projects successfully. Todd has designed and developed several Lotus Notes/Domino applications that facilitate quality assurance process and configuration management. Todd is currently the Testing Manager for the Centers for IBM e-business Innovation : : LOS ANGELES (Center).

Innovative Web Test Process and Control Tool

QW2001, May 2001

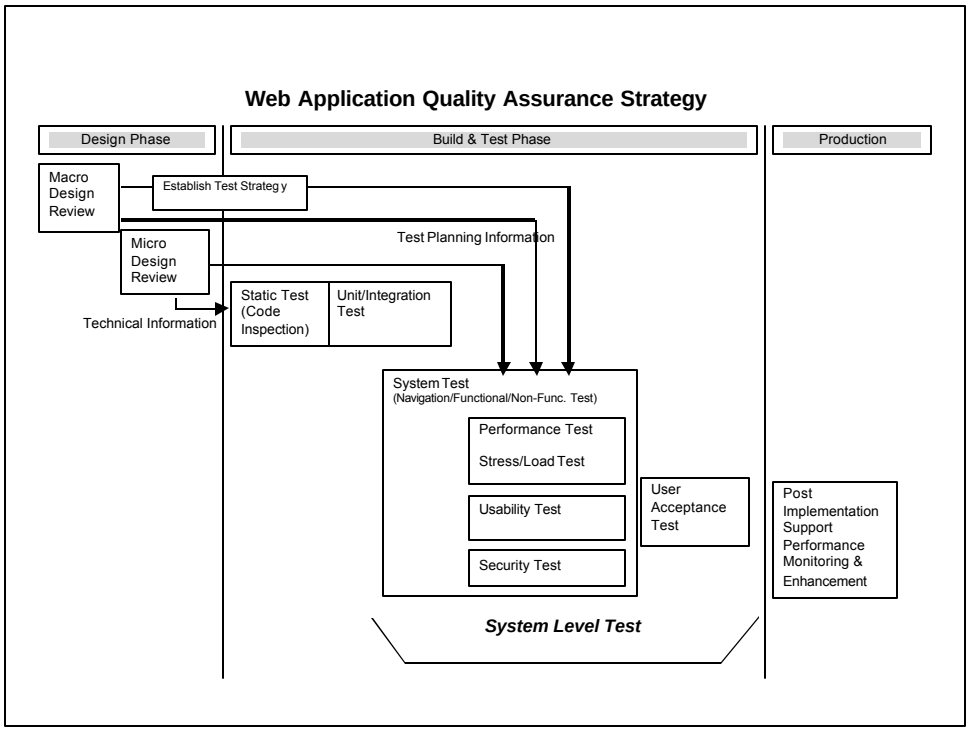
San Francisco

By Todd Hsueh, IBM

Web Testing Lessons Learned

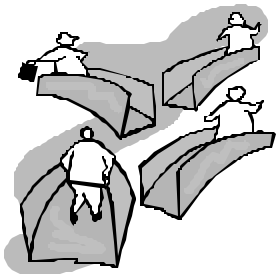
"Web changes everything ?
Web changed nothing."

"Now do you understand the importance of User Testing ?!"



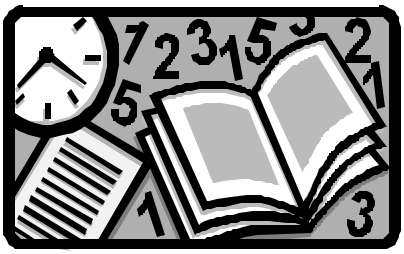
Close HUMAN Interaction

- Co/Close Location of Customer, Solution Lead, Developers, Testers
- Continuously manage expectations, no surprise
- Vendor relation



Frequent Status Checks

- Timely Information Exchange, every 4 hours !
- Share technical/testing experience
- Problem Severity & Resolution Priority setting



 5

Central Repository

- Bug/fix experience sharing cross projects
- Progress Status Report & Statistics

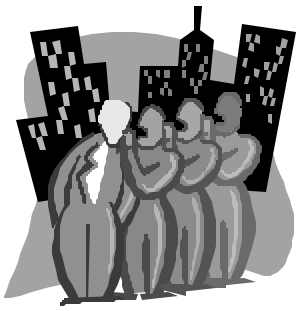
 6

Review /Walkthrough – Early QA

- Functional Requirements
- Non-functional Requirements
- Coding Standards
- Functional Templates
- Application Flow
- Use Cases
- DB Schema
- …

- Keeping all objects & documentation in sync
- Keeping all up-to-date
- Matching application behavior with business rules

Know your client's client



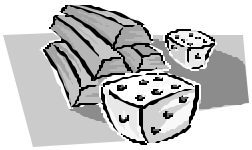
- Know their business interests
- Know their habit
- Know their environment (OS, Browser)
- Manage UAT expectations

How did a perfectly tested application fail ?

The “Impossible Best Situation”:

Development Environment = Test Environment =
User Acceptance Environment = Production Environment

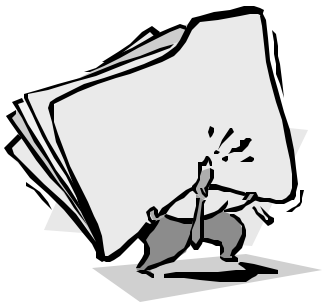
Keep them isolated !
Identify the difference and warn people !



 9

Test Scenario Maintenance

- Allow time to enhance Test Scripts
- Well-written scripts make testing efficient



 10

Load/Stress Test

- Designed for performance ?
- Schedule Load Test as soon as application is stable

 11

"Testing is hard! "

- Circulate everyone in the organization through testing at least once !
  - Now you know ….

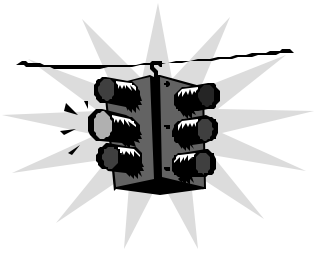
 12

Plan / Organize Testing Efforts

OS/Browser	B.C. Scenarios	Nav. Scenarios	Non-func. Scenarios
Netscape 4 / Win 95	Tester 1 - Scenario 1-5	Tester 1 – 1-100	Tester 4
IE 4 / Win 95	Tester 2 – Scenario 6-10	Tester 2 – 101 - 200	.. No need
IE 5 / Win NT	Tester 3 – Scenario 1 - 5	Tester 3 – 201 - 300	Tester3

Minimize "idle" time

- Developers, Be REAL !
- Use QA, Pre-view, Checklist, Code Release Authorization process controls

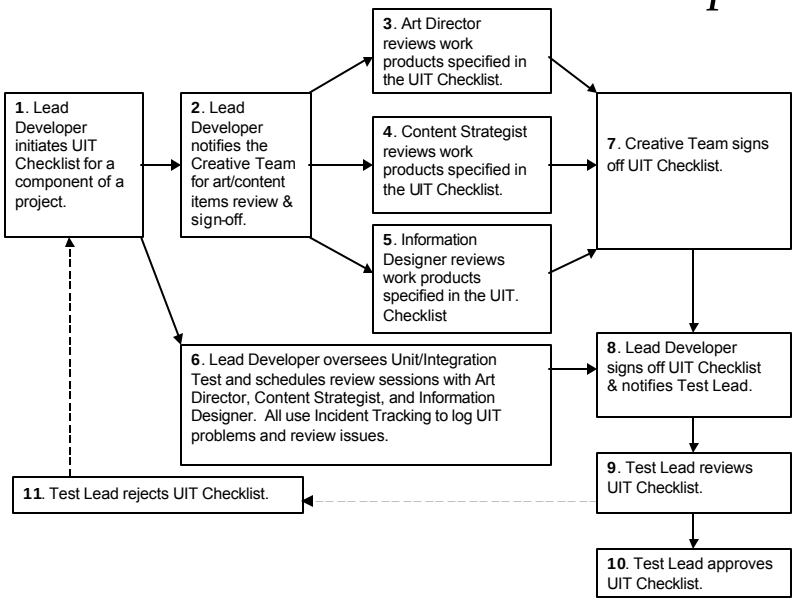


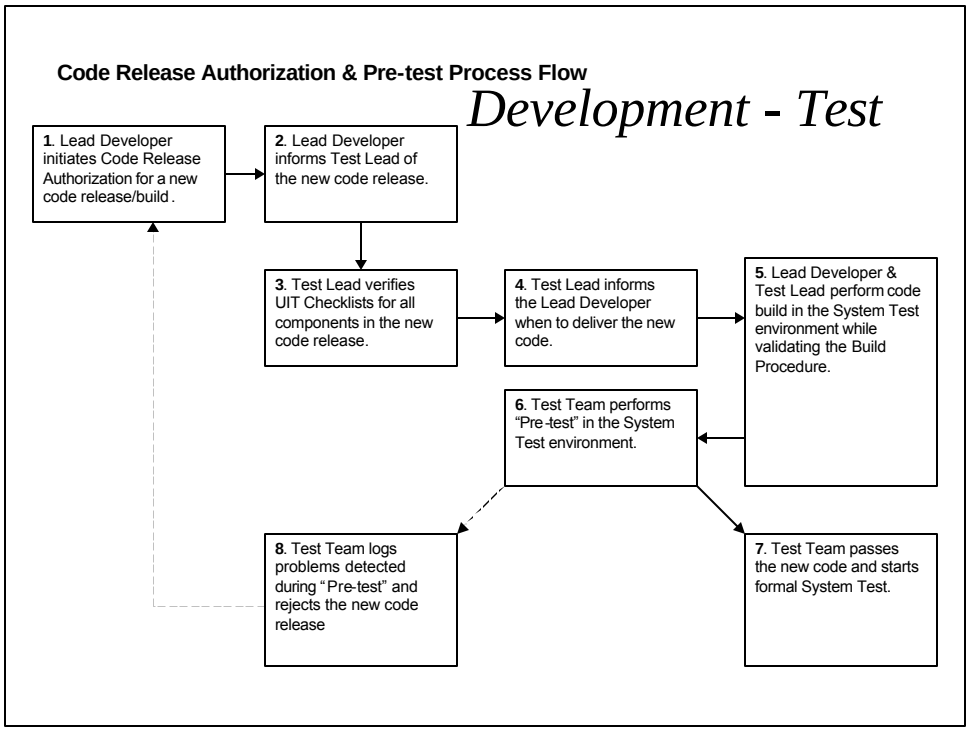
Object Management Tool & Discipline

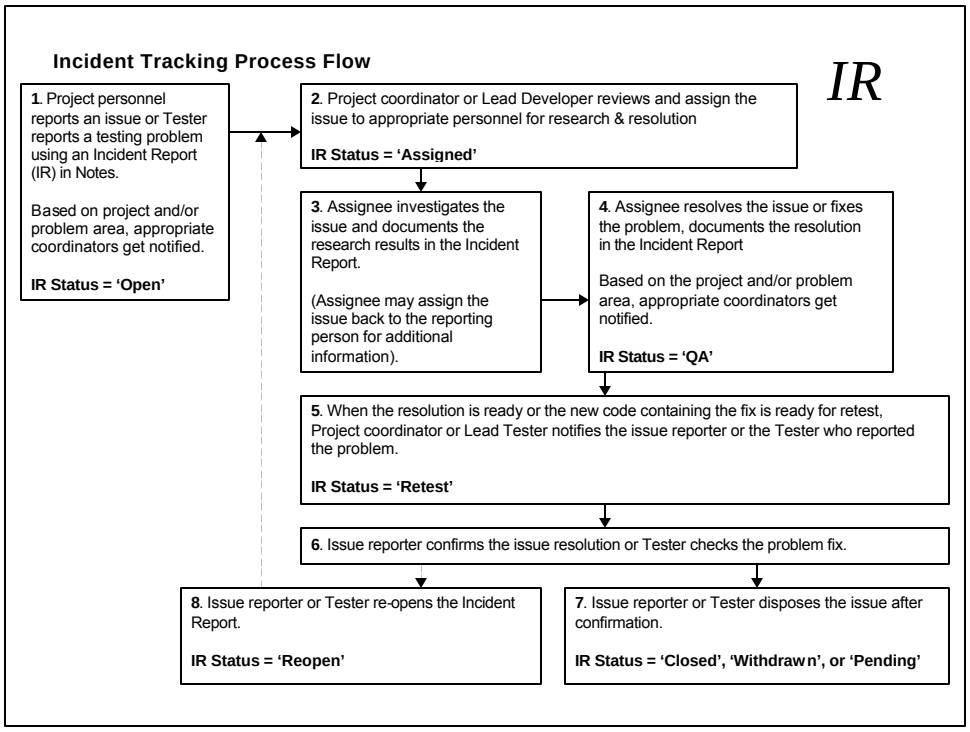
- Art
- Content
- Code

UIT Checklist Review/Approval Process Flow

Development







"Web Test & Incident Tracking" Tool Demo

- Lotus Notes Application
- Registered in the IBM WPTS and Intellectual Properties DBs

Thank you for your attention. Have a Great Conference.

QW2001 San Francisco, May 2001

Presentation (2W2):

“Innovative Web Test Process & Control Tool”

by Todd Hsueh, IBM

Web Application Development & Testing Process Control

Overview

Disclosed are a three-fold process and a Lotus Notes workflow control tool that ensures quality and efficiency in Web application development and testing projects.

The three-fold process is used by the Centers for IBM e-business Innovation : : LOS ANGELES and the workflow control mechanism is built into the Lotus Notes "Web Test and Incident Tracking" Database application.

The Challenge

Web application development and testing demand quality and speed to the market. It normally requires heavy coordination of creative artwork and technical work, and management of dynamic and static contents. Additionally, multiple work groups with cultural, work habit, and platform diversities present even more challenges to the final quality assurance (QA) group or sometimes, the Test Team who performs the final System Test to examine the functions and behavior of the Web application.

The Solution and Advantage

The answer to the above challenge is summarized in this disclosure as a three-fold process coupled with a Lotus Notes workflow control tool. The three sub processes are:

- Unit/Integration Test (UIT) Checklist Review/Approval,
- Code Release Authorization Using Pre-test, and
- Incident Tracking.

During Web application “Build” phase, the creative developers and technical developers, without much QA or tester involvement, normally perform Creative Design Review and Code Review. By enforcing the "UIT Checklist" process, QA group or testers get involved early to ensure that the creative developers (Art Director, Content Strategist, Information Designer) and the technical developers (Solution Designer, HTML/JAVA/JSP Programmers) do verify their work products against requirements and design and ensure that Unit/Integration Test is properly done before System Test starts. The workflow of notification, review, and sign-off are automated via Lotus Notes document form and email.

By enforcing the "Code Release Authorization Using Pre-test" process, the Development Team and the Test Team coordinate and synchronize their work. The new code release is delivered to the System Test environment when the Test Team is ready to receive it. Object version control and Code Build Procedures are validated with the code delivery (to the System Test environment). The "Pre-test" ensures that the new code release is properly built with quality and truly ready for System Test quickly without wasting time. The workflow of code release notification, review, and authorization are automated via Lotus Notes document form and email.

By "Incident Tracking", all project staff has a common repository and audit trail for all issues, issue status, assignments, and resolution. An issue can range from a proposal planning action item to a System Test problem. The workflow of issue notification, review, assignment, resolution, re-test, and disposition are automated via Lotus Notes document form and email. By different groupings (Lotus Notes database views), various statistics and management status reports are instantly available.

When combined, these processes synergize and ensure Web application's quality and delivery speed.

The details of each process is described below:

Process 1 - UIT Checklist Review/Approval

A Web application can be looked at as a collection of major components or Units. The "UIT Checklist" ensures that quality is built into the lower level components. Four key roles are involved in the "UIT Checklist" process. These are creative developers including Art Director, Content Strategist, and Information Designer and the lead technical developer such as Solution Lead or Development Team Lead. Every role has certain responsibilities and a portion of the checklist items to complete. In addition to having the code unit tested, by involving the Creative Team, the art/creative side of the work product is also quality assured before the code enters System Test. After verification/review is complete, each role endorses the approval of quality by a simple click of a button, which registers an electronic signature with a date-time stamp on the "UIT Checklist" in the Lotus Notes "Web Test & Incident Tracking" Database.

The Development Team Lead normally initiates the process by creating a new "UIT Checklist" by clicking an action button, which brings up a Checklist template in Lotus Notes document format. The Development Team Lead fills out the Project Name, Unit/Component Name, Code Release Number, etc. By clicking the "Notify Creative Team" button, a Lotus Notes e-mail is sent to the three key creative developers. This prompts the review and sign-off from the creative side. After validating the unit/integration test results, the Development Team Lead normally signs off last. At that time, the Test Team (Test Lead for the project) gets notified via automated Lotus Notes e-mail and the Test Team can perform quality check one final time. The Test Team may accept a "UIT Checklist" if every thing is in order. The UIT Checklist may also be rejected by the Test Lead if the subject component does not pass the unit test or the documentation is incomplete. All problems detected during Unit/Integration Test are tracked via the "Incident Tracking" function in the Lotus Notes "Web Test & Incident Tracking" database.

Process 2 - Code Release Authorization Using Pre-test

When all (or most of the) components of an application passed the "UIT Checklist" process, the Development Team Lead performs a new code build and initiates the "Code Release Authorization" process by clicking an action button that brings up a "Code

Release Authorization" form in Lotus Notes. The Development Team Lead fills out detailed information about the new code release and notifies the Test Lead via e-mail by clicking an action button. The Test Lead, upon receiving the notification, will check the completeness of the new code release by verifying the "UIT Checklists" and the "Code Release Authorization" form, and reply to the Development Team Lead when the Test Team is ready to receive the new code release in the System Test environment. When the time comes for the new code delivery, the Development Team Lead and the Test Lead will perform a new "build" in the System Test environment following the "Build Procedure". This way, the new code is properly migrated into the System Test environment and the "Build Procedure" is also validated.

After the new code is implemented in the System Test environment, the Test Team will perform a "Pre-test" using a pre-selected subset of test scenarios. This "Pre-test" certifies the readiness of the new code for the formal System Test. If the new code passes the "Pre-test", System Test will start. Otherwise, the new code release will be rejected and the "Pre-test" problems are tracked via the "Incident Tracking" function in the Lotus Notes "Web Test & Incident Tracking" database. Again, the new code release acceptance and rejection processes are also handled by action buttons and automated Lotus Notes e-mails.

Besides code release technical information, also included in the "Code Release Authorization" are Functions included/excluded, Problem Fixed, and Platform/Browser Versions applicable to the new code. This information enables the Test Team to better plan for the System Test. The "Pre-test" prevents false start of System Test and major time loss.

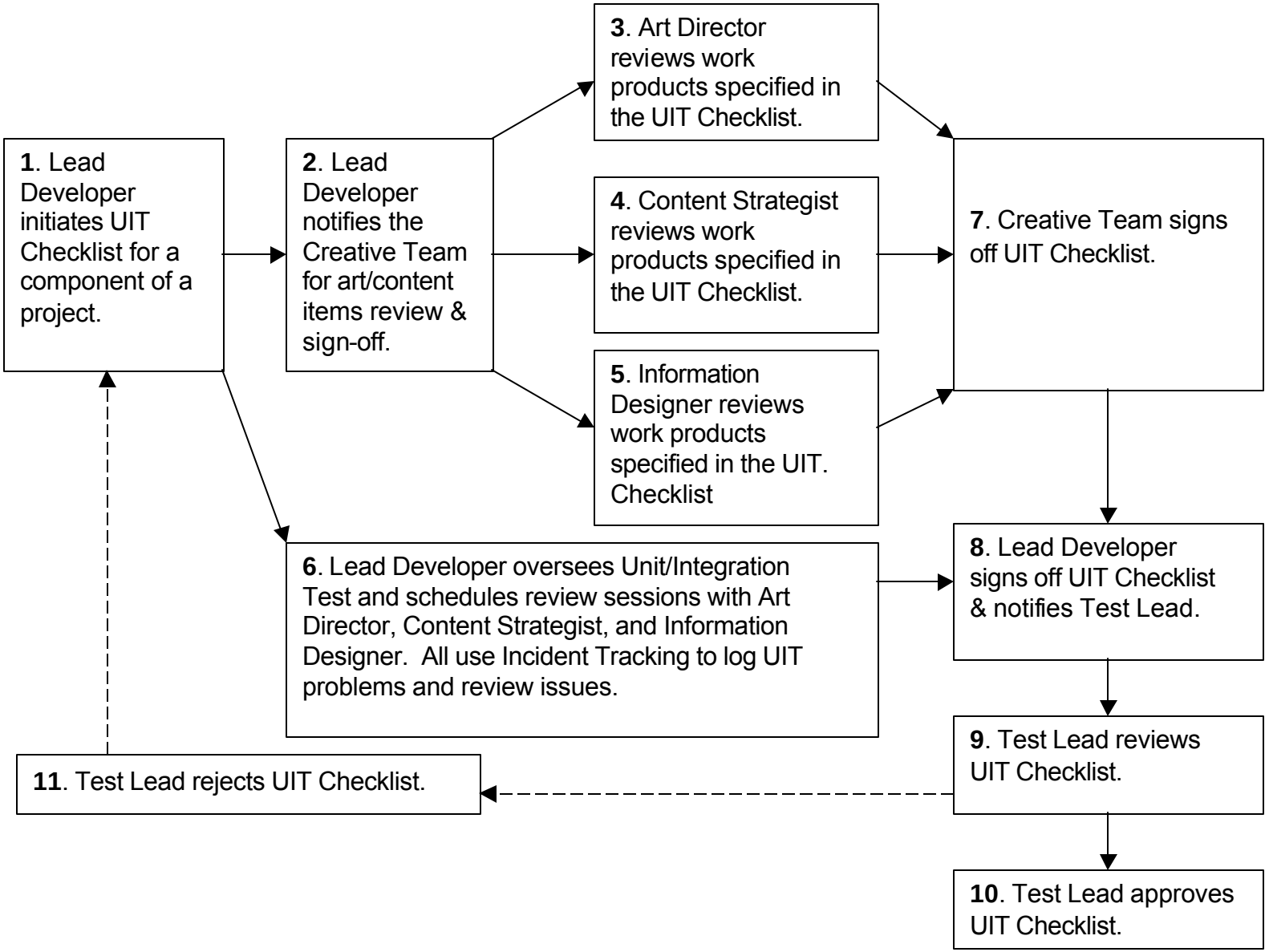
Process 3 - Incident Tracking

For best project practice, any issue that requires action and resolution needs to be tracked. Incident tracking function facilitates the logging, assignment, resolution, and verification of all kinds of issues. An issue can be reported by anyone associated with a project. By properly setting up the database, the appropriate project staff can be notified when a new issue is reported. Then the issue can be assigned to other personnel to research and/or resolve. When the issue is resolved, the assignee can inform the assigner and the original reporter. When the resolution is verified, the issue can be closed. All these interactions and notifications are automated via action buttons and automated Lotus Notes e-mail.

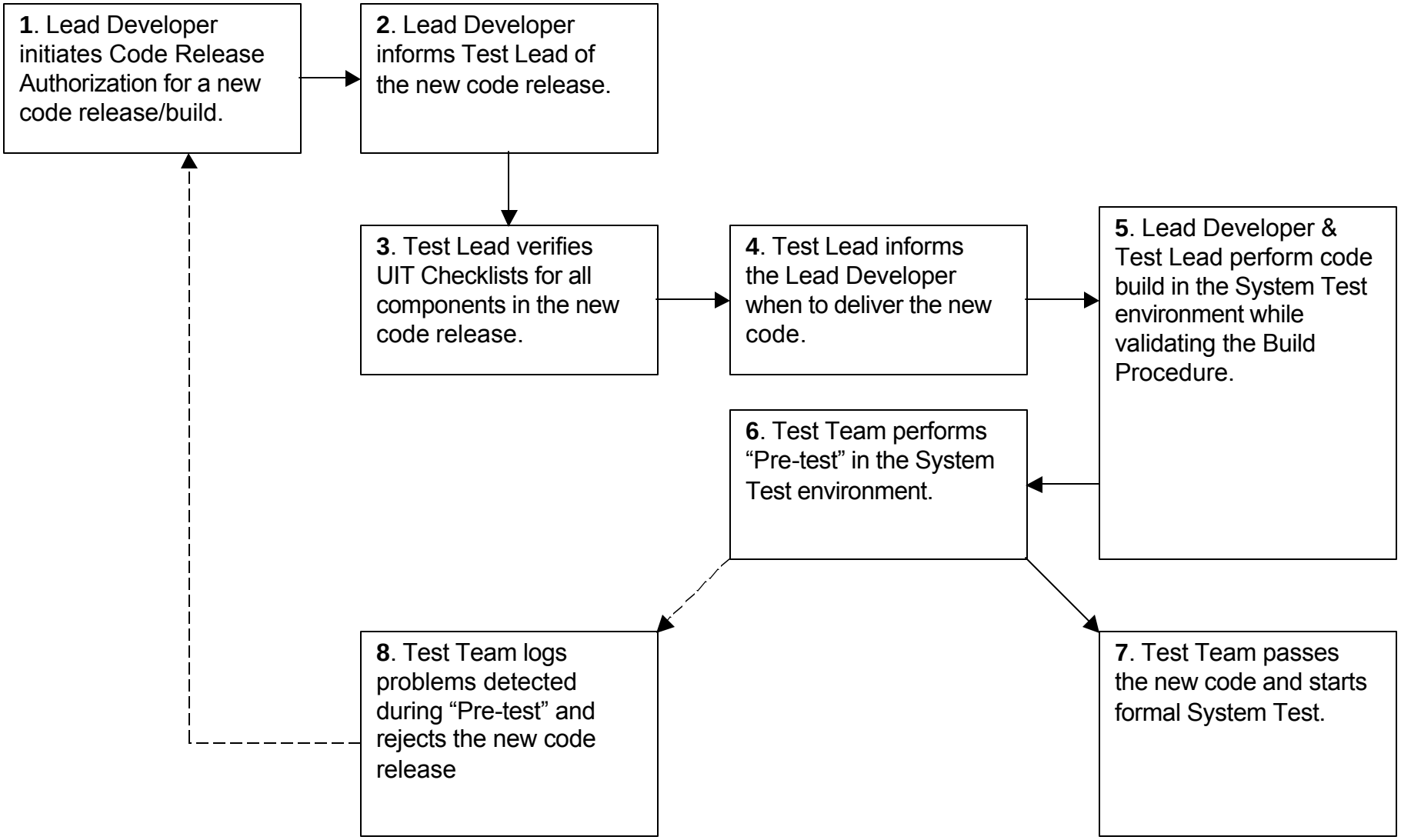
By Lotus Notes "View" structure, multiple projects can share the same database. Various project staff with different focus can create instant up-to-date management reports and project statistics easily. For testing efforts, the problem assignments/resolution cycle is shortened using the "Incident Tracking". Additionally, technical experience in terms of problem resolutions can be shared by all developers cross projects through this central repository.

The following flow charts illustrate the three processes described in this disclosure.

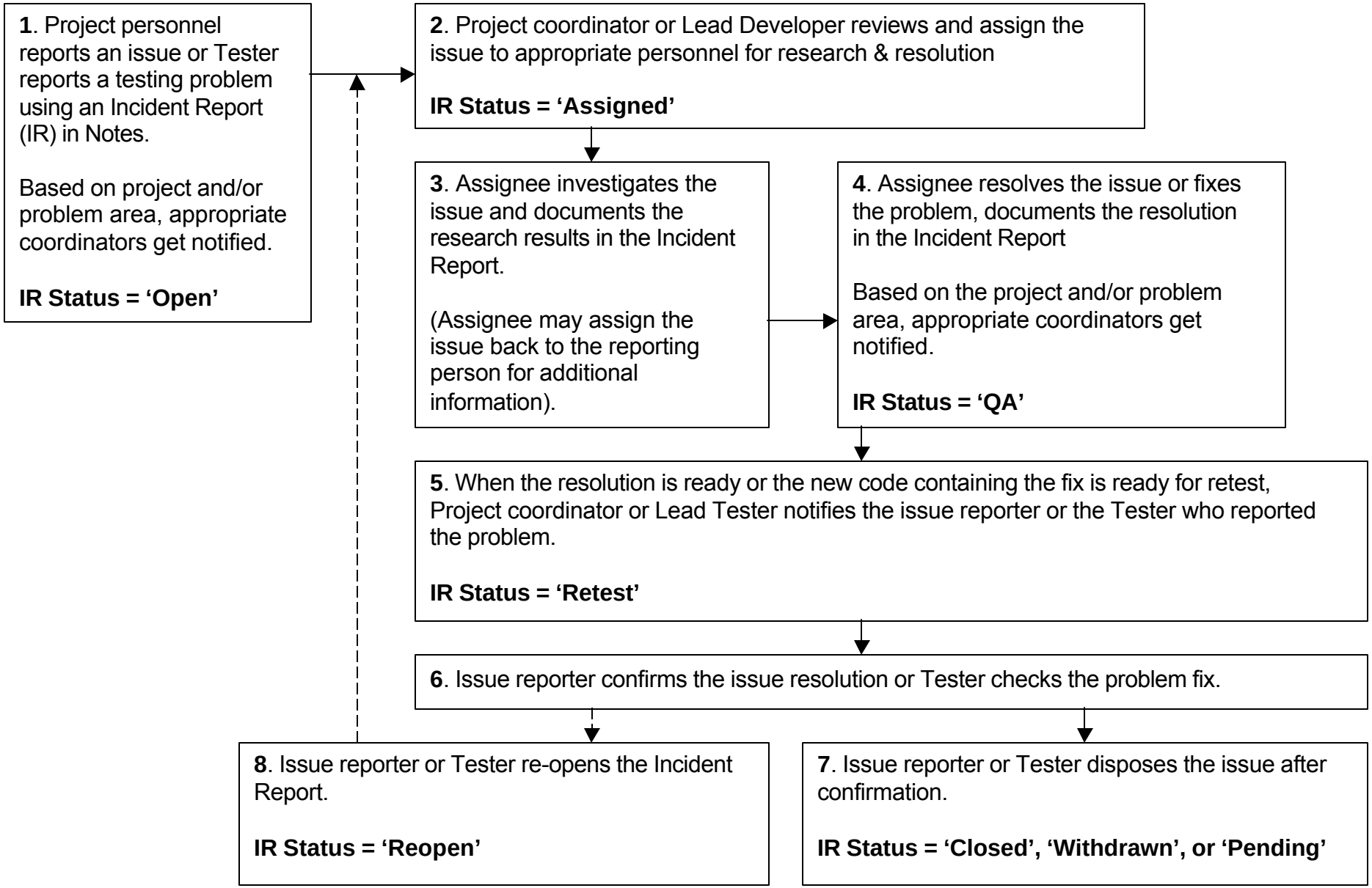
UIT Checklist Review/Approval Process Flow



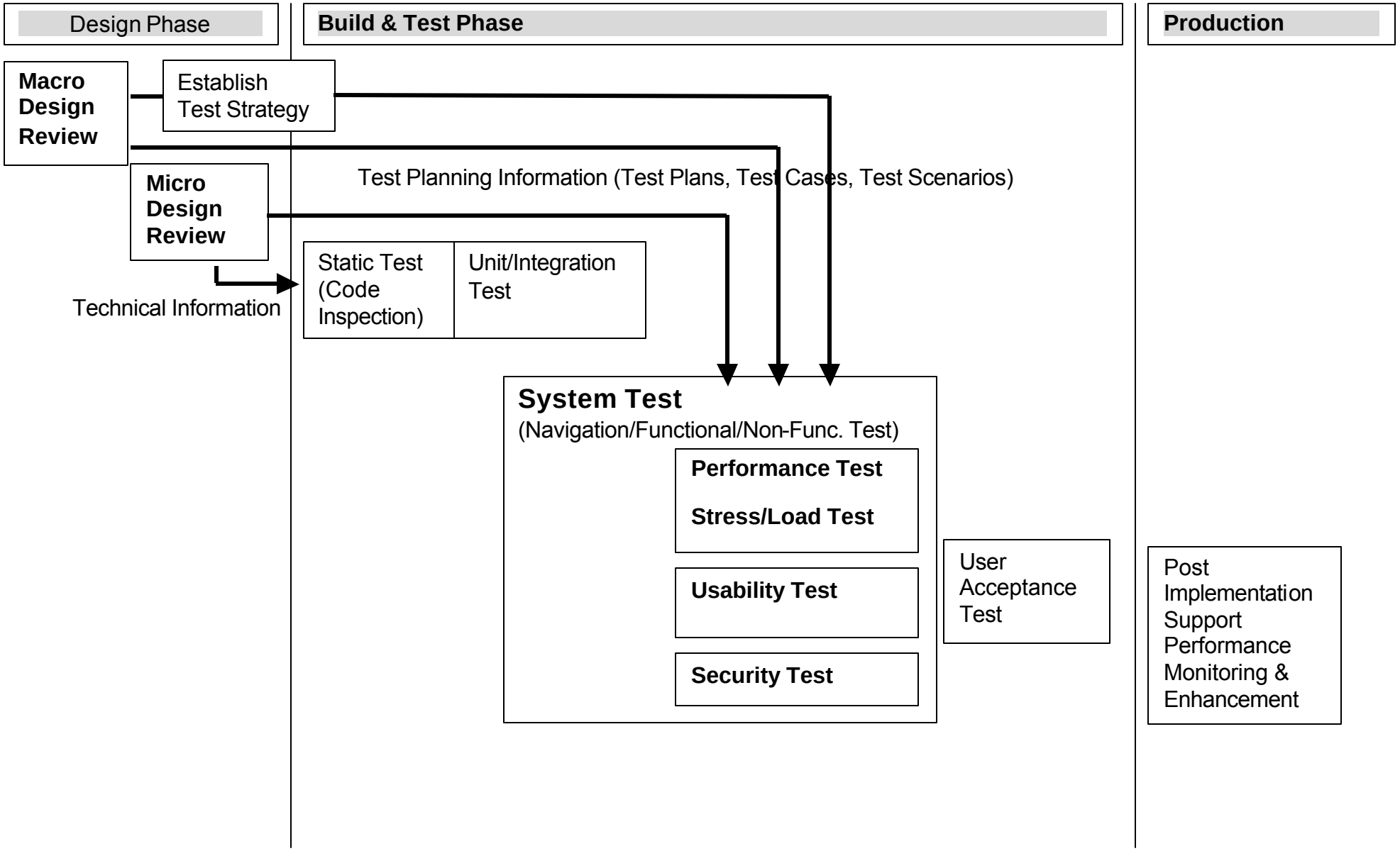
Code Release Authorization & Pre-test Process Flow



Incident Tracking Process Flow



Web Application Quality Assurance Strategy



QW2001 Paper 3W1

Ms. Nancy Landau
(Alltel Technology Services)

Performance Testing Applications In Internet Time

Key Points

- Standardizing test processes
- Profiling applications and user workflows
- Working with diverse development teams

Presentation Abstract

How can a test team simplify the transition from performance testing two-tier client/server applications to testing complex, multi-tier web applications combined with an exponential growth in testing needs? In this presentation, Nancy Landau presents case studies that address changes made in performance testing methods to handle compressed delivery schedules, new architectures and technologies, and changing customer expectations. The experiences focus on performance testing, but the strategies apply to all test efforts.

About the Author

Nancy Landau has 15 years of experience in quality assurance and financial services. She has been involved in design, development, deployment, test, and support of large-scale client/server solutions for the mortgage banking industry. She is the lead client/server performance test analyst for the Residential Lending Division of ALLTEL, a Fortune 500 company.

Performance Testing Applications in Internet Time

Nancy Landau

ALLTEL INFORMATION SERVICES

Objectives

- Review performance testing basics
- Describe fundamentals
- Explain success factors
- Review examples

Audience: Web developer, performance engineer, stress test / QA project manager

Terms & Concepts

- Application Under Test (AUT): The software application(s) being tested.
- System Under Test (SUT): The hardware & operating environment(s) being tested.
- Virtual User: Software process that simulates real user interactions with the AUT.
- Process/Workflow: A user function within the AUT.
- Scenario: A set of workflows defined for a set of virtual users to execute.
- Transaction: A subsection of the measured workflow; more granular user events for which response time will be measured.
- Bottleneck: A load point at which the SUT/AUT suffers significant degradation.
- Breakpoint: A load point at which the SUT/AUT suffers degradation to the point of malfunction.
- Scalability: The relative ability or inability of the AUT/SUT to produce consistent measurements regardless of size of workload.

Why Performance Test?

- Internet applications bring performance issues direct to your users
- Slow response times and errors have a direct cost

Test Development

- Automated testing IS software development
- Lifecycle mirrors product development
- Use iterative test development
- Emphasize planning stages
- Plan for reuse

Plan - System Usage

- Get system usage information
  - Identify workflows
  - Define typical user profiles
  - Define transactions and expected results

- Examine typical and peak workloads
- Define access methods - connection speeds, etc.
- Trace use cases to components and hardware

Case Study #1

- Web application for data analysis reports
- Outsourced development
- Big difference in estimated capacity
- Different model workflows
  - Report sizes
  - User actions
  - Cached data
- Lesson: understand usage patterns

Plan - AUT Architecture

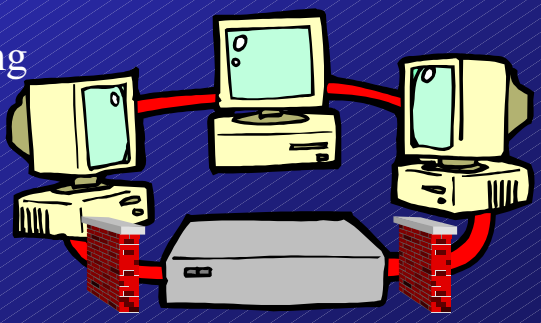
- Perform architectural walkthroughs
- Understand security methods
- Review 3rd party components
- Understand queues
- Identify caching models
- Examine session management
- Verify tool compatibility with AUT

Case Study #2

- Web-based customer service application
- 3rd party component for host connectivity
  - Bottlenecked only on multi-processor servers
  - Vendor provided recompiled component
- Delphi controls, 3270 Active-X
- GUI test tool supported both Delphi and 3270
- BUT not together!
- Lessons: know your components and your tools!

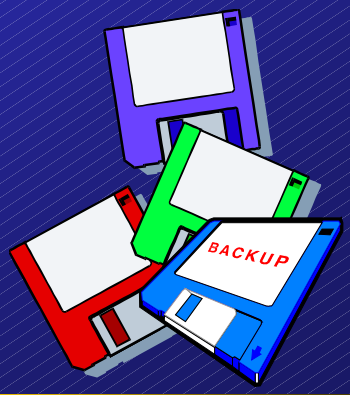
Plan - SUT Architecture

- Review physical infrastructure
- Examine firewalls
- Review connectivity
- Identify load balancing
- Review encryption



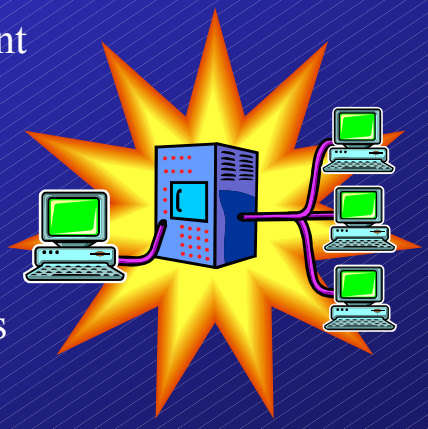
Plan - Test Data

- Define representative set of data
- Define appropriate volume of stored data
- Develop test database
- Plan backup and restoration
- Establish data verification points



Plan - Test Environment

- Obtain dedicated environment
- Typify production
  - Hardware
  - Networks
  - Databases
- Perform manual dry run tests
  - Identify concurrency risks
  - Confirm application behavior

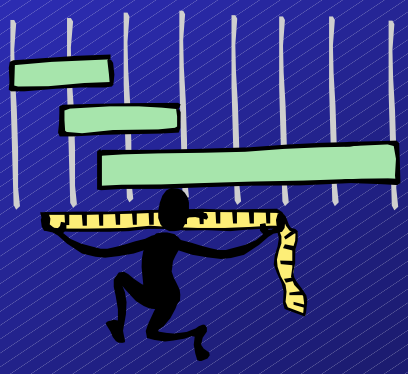


Case Study #3

- Web-based ad-hoc reporting tool
- Development environment limited
- Preliminary test in near-production environment exhibited concurrency issue with just one user
- Lesson: use near-production environment

Plan - Test Metrics

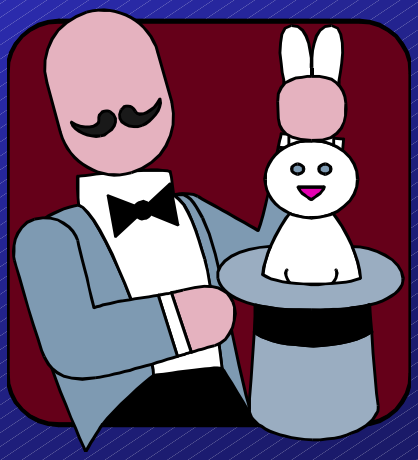
- Response times
- Session abandonment
- Server utilization
- Network load



Correlate test metrics to production monitoring

Plan - Toolkit

- Automated test tools
  - virtual users
  - GUI users
- Monitoring methods
  - Server performance
  - Network load
  - Client / virtual user driver
- Logs and log parsers
- Synchronize the measurements!



Create Virtual Users

- Record user actions
- Define wait / think times
- Add transactions
- Add verification checks
- Parameterize data

Create Scenarios

- Establish mix of virtual users
- Ensure a varied, representative workflow
- Establish monitoring points

Perform Dry Runs

- “Test the test”
- Use full logging
- Identify unexpected conditions
- Review instrumentation and monitors
- Validate parameterized data
- Revise test scripts

Case Study #4

- Web-based customer service application
- Web server interpreted response from application server as success
- Application server returned errors in the response
- Examined return data to identify “true” success
- Lesson: HTTP 200 is not always success! Validate responses against expected results

Perform Tests

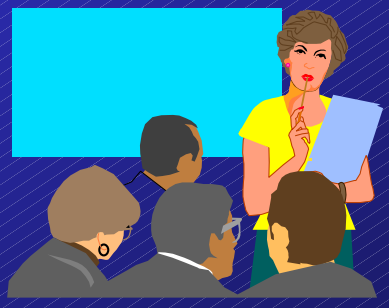
- Reduce logging to production levels
- Ramp-up virtual users
- LAN versus WAN tests
- Identify and troubleshoot bottlenecks
- Track all changes

Analyze Results

- User response times
  - Averages
  - 75th percentile or higher
- Memory utilization
- CPU utilization
- Server configuration
- Database and SQL tuning
- Code tuning

Report the Results

- Report components
  - Executive summary
  - Report body
  - Appendices
- Identify the audience
- Mirror the test plan
- Acknowledge contributions



Rinse and Repeat

- Emphasize reuse
- Develop plan templates
- Create report templates
- Develop application matrix
- Develop reusable test components
- Define standard measurements
- Define and share toolkit

Conclusion

- Automated testing is development
- Planning is essential
- Plan for reuse
- Profile the users
- Learn the technologies
- Understand the infrastructure

Questions?

Thank You!

Nancy Landau

Nancy.Landau@ALLTEL.com

QW2001 Paper 3W2

Mr. Steve Splaine
(Splaine & Associates)

Modeling The Real World For Load Testing Web Sites

Key Points

- Concurrent users - why do they make a difference
- Can you/should you play with "Think Times"?
- The effect of cookies, SSL and client-side connections on load generators

Presentation Abstract

Requesting your Web site's home page 100 times per minute is not going to give you a very accurate idea of how your Web site is actually going to perform in the real world. Explore the variables that you should consider when designing a Web load or stress test, including user activities, security, user access speeds, and geographic locations.

About the Author

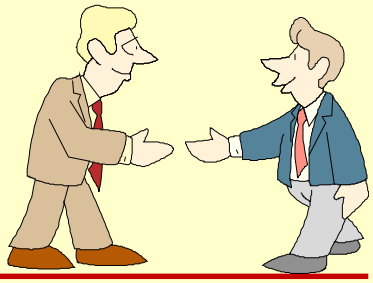
Steve Splaine is a chartered software engineer with over 20 years experience in developing software systems: Web/Internet, Client/Server, Mainframe, and PCs. He is an experienced project manager, tester, developer, and presenter, who has consulted with over 100 companies in North America and Europe. In addition, Steven is the lead author of the recently published software-testing book "The Web Testing Handbook".

Modeling the Real World for Load Testing Web Sites

Presented by
Steve Splaine

Steve@Splaine.net

Presentation Overview



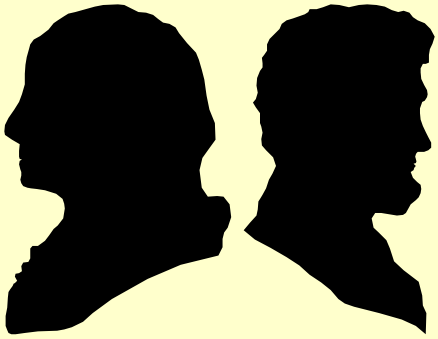
Measuring the Load

- Hits per day
- Page views per day
- Unique visitors per day

- Transactions per second
- MB per second
- # of concurrent users
- # of session initiations per hour

User Profiles

In order to better simulate the real world, not only should the load be estimated, but also the user profiles of the users that make up the load.



User Activities

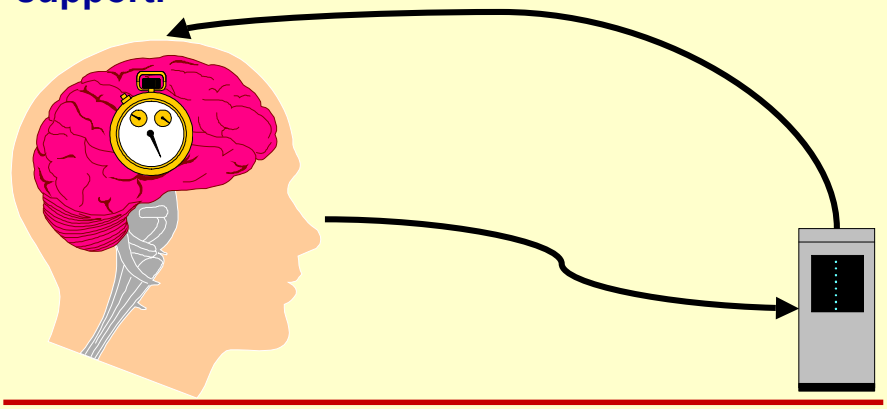
Some transactions occur more frequently than others and should therefore make up a larger proportion of the test data/scripts used for performance testing.

Browsing

Buying

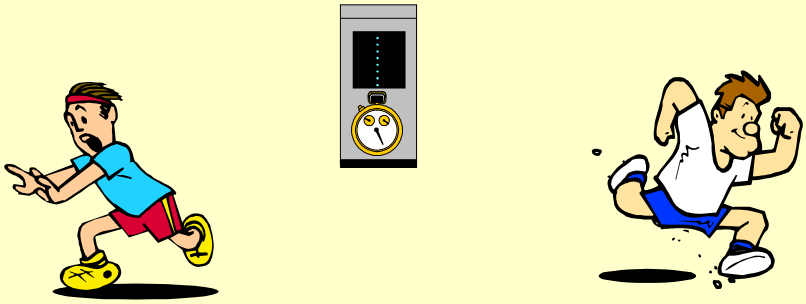
Think Times

The time it takes a client (or virtual client) to respond to a Web site has a significant impact on the number of clients a Web site can support.



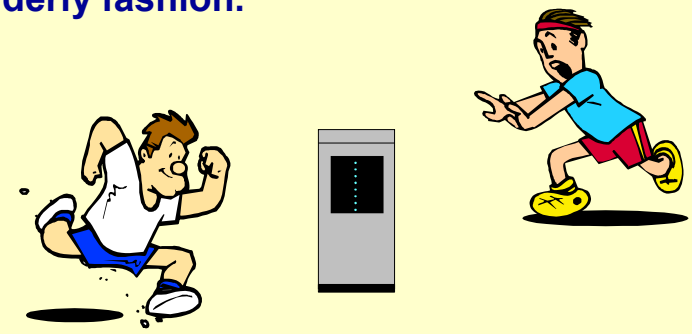
Site Abandonment

The time it takes a Web site to respond to a client has a significant impact on whether the client will request a subsequent page.



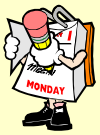
Site Arrival Rates

In the real world, visitors typically arrive at a Web site in a random distribution, not in an orderly fashion.



Usage Patterns

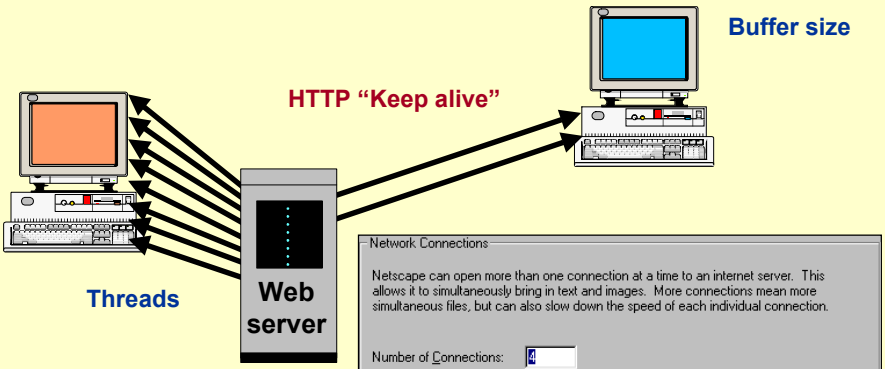
Unlike typical mainframe or client/server applications, Web sites often experience huge swings in usage.



Hits by Day of the week(1000's)		
Monday: 131	16%	
Tuesday: 153	18%	
Wednesday: 145	17%	
Thursday: 141	17%	
Friday: 131	16%	
Saturday: 75	9%	
Sunday: 68	8%	

Hits by hour of day																							
2%	3%	2%	2%	2%	2%	3%	2%	3%	3%	7%	6%	7%	7%	6%	6%	6%	4%	7%	4%	5%	4%	5%	3%
12	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11
Morning (AM)												Night (PM)											

Client Network Connection Options

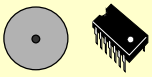


Client Preferences

<ALT>

Graphics on/off?

JavaScript/
VBScript on/off?



Java/ActiveX
on/off?

Cache sizes

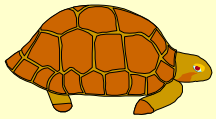
Language selection

Cookies on/off?

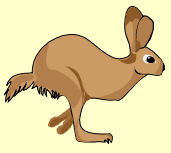
Encryption settings

Client Internet Access Speeds

Sample transmission methods:

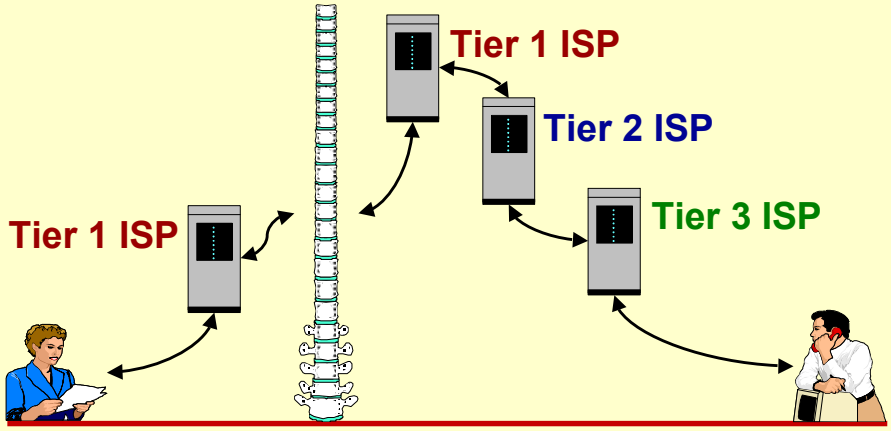


- Dial up
  - Land line/Cellular/Satellite
  - Analog/Digital
  - 14.4Kbps, 28.8Kbps, 33.3Kbps, 56.6Kbps
- ISDN, ADSL
  - 1 Line/2 Lines
  - 128Kbps to 1.5Mbps (T1)
- LAN/Cable modem
  - Dedicated/Shared
  - 1.5Mbps (T1) to 45Mbps (T3) +
- Frame relay
  - 56Kbps to 45Mbps (T3) +
- Optical
  - 52Mbps (OC1) to 34 Gbps (OC768) +



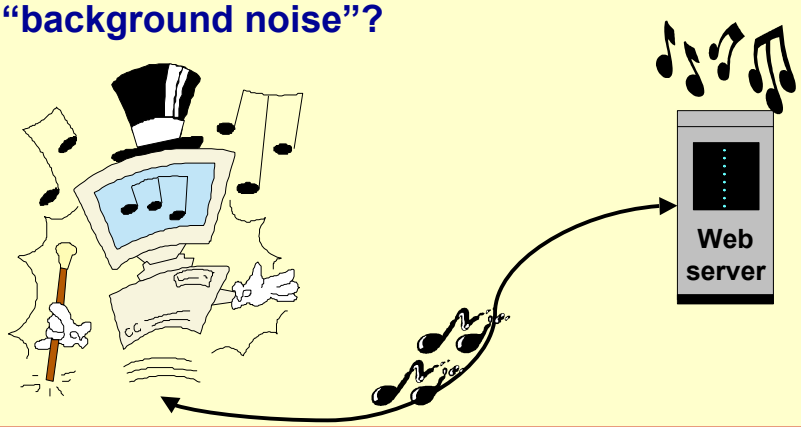
ISP Tiers

The number of hops (or Tiers) a packet of data has to make before reaching the Internet Backbone will affect a client's response time.



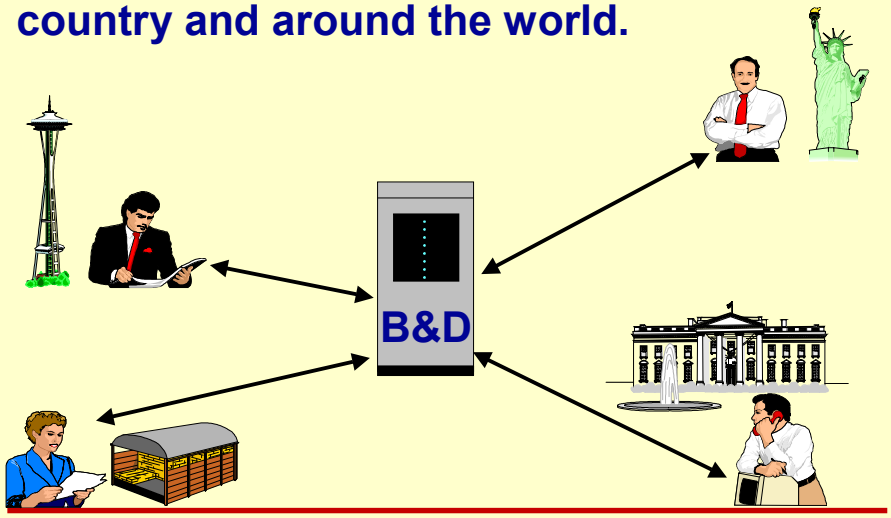
Background Noise

What additional activities need to be considered to accurately reflect the performance degradation caused by network and application "background noise"?



User Geographic Locations

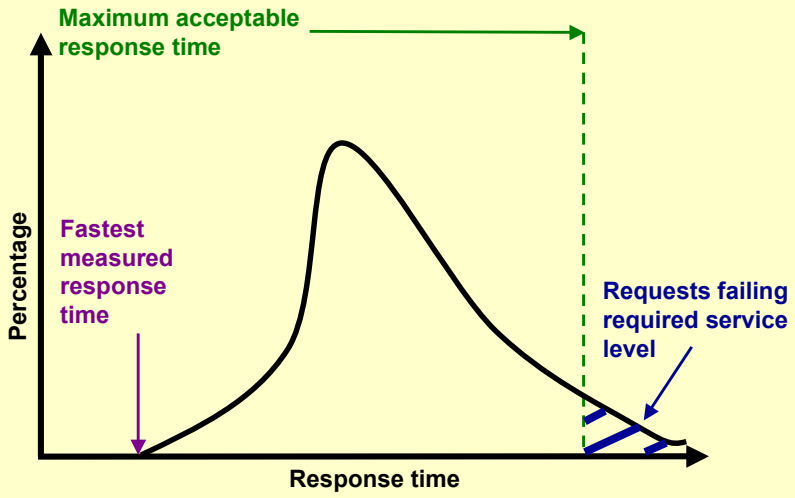
Response times vary around the country and around the world.



Getting the Right Mix

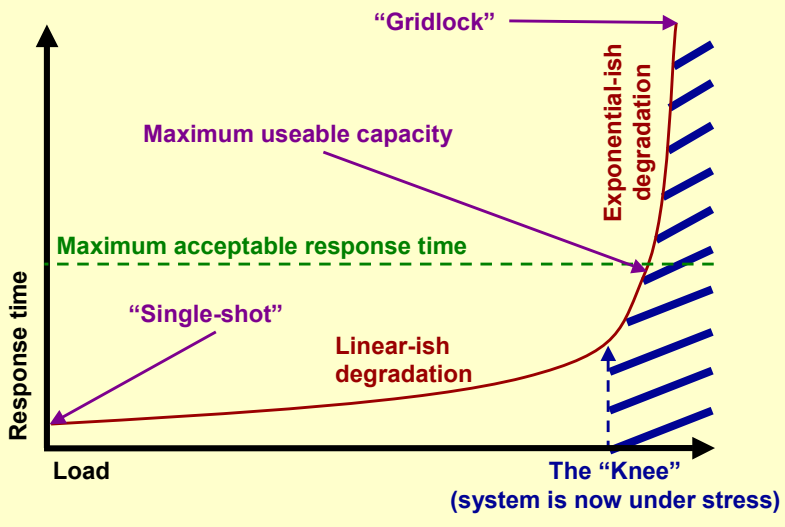
The parameters that can be taken into account, the more accurate the model of the real world.

Example Response Distribution Graph

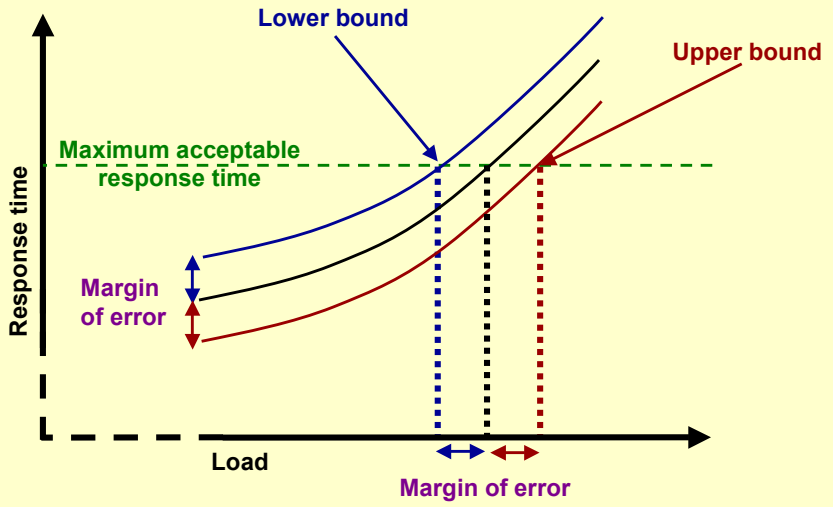


Note: Load remains constant

Example Response Time Graph



Margin of Error



Presentation Wrap Up

- Estimate load size
- Design user profiles
- Generate test data to support user profiles
- Run 3 sets of tests
  - Best guess
  - Pessimistic scenario
  - Optimistic scenario


QW2001 Paper 4W1

Mr. Nikhil Nilakantan, Mr. Ibrahim K. El-Far
(Florida Tech)

Why Is API Testing Different

Key Points

- Understand the challenges of API testing
- Study realistic usage scenarios
- Build API test automation

Presentation Abstract

This presentation describes the challenges unique to the testing of Application Programming Interfaces (APIs). We isolate and explain three problems/difficulties commonly encountered while testing APIs.

These problems are:
1)Parameter Selection
2)Parameter Combination
3)Call Sequencing

We then provide solutions to these challenges and through examples show how to build effective API test automation.

About the Author

Nikhil Nilakantan is a graduate of Florida Tech where he received his B.S. in Computer Science. He was a researcher at the Center for Software Engineering Research at Florida Tech for 3 years. In the past he has worked at Microsoft Corporation as a test engineer. He also occasionaly teaches classes on software testing methodology. His fields of interest/research are software reliability, model-based testing, intelligent test automation and improving test management processes. He is currently a Quality Assurance Engineer with Hewlett Packard Corporation in Cupertino, CA.

Ibrahim K. El-Far is a doctoral student in computer science under James A. Whittaker at the Florida Institute of Technology, Melbourne, Florida. He has been working with model-based testing techniques for over four years at the Center for Software Engineering Research at FIT. His interests are in investigating new software models, test automation and tools, adequacy criteria, and software testing education. In 2000, El-Far received an IBM CAS Fellowship supporting his

research in software testing.

Why is API Testing Different?

Nikhil Nilakantan
Hewlett Packard Corp.
(nikhil_nilakantan@hp.com)

Ibrahim K. El-Far
Florida Institute of Technology
(ielfar@acm.org)

Agenda

- Introduction
- APIs Are Different
- APIs Are Complex
- Testing APIs
- Three Nuts to Crack
- Additional Issues
- Exploring an API
- Testing an API
- Automation
- Automation Demo
- Future Work & Conclusion
- Q & A

May 30th, 2001

Introduction

- Many styles of testing
- Many types of applications
- APIs constitute a large part of these applications
- No Silver Bullets

APIs Are Different

- Generally misunderstood
- Different for many reasons:
  - Invisible to the human user
  - Require a knowledge of inner workings
  - Require considerable programming skills
  - Tend to be high in complexity

APIs Are Complex

- The Word Object 9.0 Model contains:
  - Around 200 classes
  - Over 200 enumerated types
  - Most classes have at least 10 methods
- The Word 9.0 Document Class contains:
  - 125 fields, 3 events & 63 methods

  Not counting the parameters or events for each method we have 125 X 63 = 7875 combinations
  - Many parameters are complex types or class objects

May 30th, 2001

Testing APIs

- Testing involves designing of sequences to satisfy test objectives
  - Requires isolation of specific parameters
  - Requires a mechanism to evaluate return values

How would you test a call with 31 parameters?

Function **CreateLetterContent** (DateFormat As String, IncludeHeaderFooter As Boolean, PageDesign As String, LetterStyle As WdLetterStyle, Letterhead As Boolean, LetterheadLocation As WdLetterheadLocation, LetterheadSize As Single, RecipientName As String, RecipientAddress As String, Salutation As String, SalutationType As WdSalutationType, RecipientReference As String, MailingInstructions As String, AttentionLine As String, Subject As String, CCList As String, ReturnAddress As String, SenderName As String, Closing As String, SenderCompany As String, SenderJobTitle As String, SenderInitials As String, EnclosureNumber As Long, [InfoBlock], [RecipientCode], [RecipientGender], [ReturnAddressShortForm], [SenderCity], [SenderCode], [SenderGender], [SenderReference]) As LetterContent

May 30th, 2001

Three Nuts to Crack

- Parameter selection
  - Selecting "interesting" values
  - Exercising boundary conditions
- Parameter combination
  - Exercising stored data & computation
  - Separately legal values maybe illegal when used together
- Call sequencing
  - Almost impossible to test all combinations

Additional Issues

- Inadequate domain knowledge
- Poor documentation
- Unavailability of source code
- Time constraints

Exploring an API

- Things to do BEFORE you begin testing:
  - Review documentation
  - Map the interface
  - Review source code
  - Isolate (and deliver) the following artifacts:
    - Common calls, parameters
    - Valid/Invalid parameters & return values
    - Realistic usage scenarios
    - Utilized resources

Testing The API

- When is automation desirable?
  - Do you have enough information about the system under test?
  - How much effort is required to write the automation?
  - How expensive is the automation to maintain?
  - How effective is the automation going to be?
- Kinds of automation
  - Capture-Replay automation
  - Monkeys
  - Intelligent test automation

Building Automation

Good automation should be:

- Modular
- Flexible (easy to change)
- Scalable (easy to extend)
- Understandable

May 30th, 2001

Automation Too Expensive?

If automation is too expensive – Go back to basics!

- Assess risky API calls, parameters & values
- Perform boundary analysis on parameters, values
- Stress boundary conditions
- Use combination tests

May 30th, 2001

Automation Demo

May 30th, 2001

Conclusions

- APIs are different
- APIs are complex
- API testing is NOT an easy problem
- Work on API testing is in its fledgling stages

May 30th, 2001

Future Work

- Defining API test adequacy criteria
- Using model-based approach(es) to resolve:
  - Parameter selection problem
  - Parameter combination problem
- Automated API exploration tools
- Comparing effectiveness of testing methodologies with respect to APIs

Questions?
Comments?
Suggestions?

Nikhil Nilakantan
nikhil_nilakantan@hp.com

Ibrahim K. El-Far
ielfar@acm.org

Presented at the 14th International Internet and Software Quality Week (QW2001), San Francisco, California, USA on May 30, 2001.

Why is API Testing Different?

Nikhil Nilakantan , *Hewlett Packard*
Ibrahim K. El-Far, *Florida Institute of Technology*

Abstract

A large majority of the software tested today is in the form of application programmable interfaces or APIs. APIs are clearly distinct in many aspects from other software interfaces. Thus, the software engineering community is presented with some unique testing problems. Some of these problems are a natural outcome of the characteristics of APIs. For example, APIs cannot be visually explored like graphical user interfaces, and they do not necessarily leave a persistent effect on the operating environment such as file system interfaces. APIs are also generally more complex and, for the most part, rich in functionality. Some issues that testers need to deal with everyday like poor documentation and inadequate personnel skill manifest themselves severely with APIs. We concentrate on problems that are challenging to solve even in the most ideal of situations. This paper isolates and addresses three such challenges: parameter selection, parameter combination, and call sequencing. Some heuristic versatile approaches are suggested as basic steps that are helpful in meeting these challenges. Recommendations on how to build API test automation and a presentation of the issues surrounding automation conclude this paper.

Keywords

Programmable interfaces, API testing, call sequencing, test automation

1. Introduction

1.1 Two Lessons in Testing

There is an unmistakable rise in the demand for quality in today's software industry. Government agencies and corporate customers are keen on demanding high quality especially when such factors as safety, security, or performance are critical to success. The quality requirement is not restricted to this variety of customers anymore. The last few years have witnessed the popularization of the Internet and the exposure of the world of software to the masses. Traditional as well as up-and-coming vendors are competing for shares of this new market, and public satisfaction with their software is key to their prosperity. E-commerce constitutes an outstanding example of this.

Nikhil Nilakantan is a software quality engineer at Hewlett Packard, Cupertino, California, USA (nikhil_nilakantan@hp.com); Ibrahim K. El-Far is a doctoral student in computer science at the Florida Institute of Technology, Melbourne, Florida, USA (ielfar@acm.org).

Nowadays, little distinguishes one product from another that is developed for the same purpose and market. The feature demand and supply are typically the same. The cost-to-consumer and time-to-market constraints influence all competitors alike. One of the very few things that set a vendor apart is the quality of their product, and, in this paper, we are interested in only two aspects. These are how well a product conforms to its specifications and how well it meets the expectations of its customers. In order to gain confidence in these facets of software quality, present-day industry is inclined to rely on testing.

Software engineering has an abundance of testing styles. Over the past two or so decades, a proliferation of paradigms, techniques, and case studies have appeared in the relevant literature. Many statements can be made about that body of knowledge, but two basic lessons come up repeatedly.

There are no silver bullets in software testing. This has been suggested and proven by computing theory a few decades ago. It is impossible for testing to reveal all faults in an application [3]. No style of testing can be said to be unconditionally superior over another. No style is guaranteed to exercise an application under test in every way possible since that, too, is unachievable [7]. However, we can say that different styles exercise software in (sometimes radically) different ways. Consequently, following different lines of attack on the software can build confidence in the quality assurance process and the quality of the released product that is proportional to how well the latter withstands attacks.

How well different styles do depends on that application type and properties and on test objectives. This is less articulated in the literature. Each application presents challenges unique to its genre, properties, operating environment, or project conditions. Imagine the disparity in testing graphical user interfaces and file systems; in testing applications implemented in C++ and those implemented in Java; in testing in the Windows and UNIX operating environments; and in testing a well-specified system and a similar system with little supporting documentation. Therefore, there is a need for studies on what collections of testing techniques are useful for particular situations, software genres, and objectives.

1.2 Testing Programmable Interfaces

Application programmable interfaces or APIs are the constituents of most large applications and some major operating systems such as Microsoft Windows. APIs drive everything from graph drawing packages, to speech engines, to web-based airline reservation systems, to computer security components. Many applications can also be viewed and treated as APIs from a testing perspective. Compilers are a good example, where program statements can be regarded as API calls. Despite their apparent significance, little work has been done to isolate and study the problems surrounding the test of APIs.

The abundance of programmable interfaces seems to be sufficient justification to warrant such a study. However, what makes APIs a different class of systems from a testing

perspective? Are there any problems that surface only when working with APIs? Are there problems that manifest themselves more severely when working with these interfaces? Unfortunately, APIs are often looked upon in the same light as other common software interfaces [14] such as graphical user interfaces or GUIs. This is perhaps caused by a lack of understanding or awareness of their distinctive attributes.

Programmable interfaces are invisible to the human user. Even though API calls may result in humanly visible output, they cannot be directly made by humans, and their return values are certainly not immediately observable. This renders caused-effect analysis and the like very hard to perform. Most importantly, this makes the utilization of exploratory techniques [6] cumbersome without the aid of custom-built tools.

Testing APIs requires a thorough knowledge of its inner workings. API calls often interact with the operating environment both affecting it and being influenced by it; for instance the operating system or OS kernel and memory handling modules. API calls often cause a cascade of other calls in the same API, calls to other software, and calls to the OS. Developing an understanding of the inner workings serves to describe and detect the state of the operating environment when a routine is called and in diagnosing failures caused by call sequences.

Testing APIs requires considerable programming skills. API tests are generally in the form of sequences of calls, namely, programs. Even when tools are available to automate some code generation, each tester must possess expertise in the programming language(s) that are targeted by the API. In addition, when the source code is available for review and scrutiny, the tester is at a substantial disadvantage without knowledge of the programming language(s) and tool(s) with which the API is implemented.

Programmable interfaces, in general, are functionally rich and complex in nature, supplying its users with many routines, types, classes, and constants. The complexity alone makes testing them a thorny task, which can never be fully appreciated without an example or going through the testing experience.

Take for example the Microsoft Word object model [9]. This API has a little fewer than 200 classes and well over 200 enumerated types. Few of the classes have less than 10 members to consider. Many of these classes are much more complicated than that. For example, the Document class has 125 fields, 3 events, and 63 methods. Members of a class are often classes in the same model, adding the complexity of the member to that of the object. The rather simple-looking Column class has 5 methods and 15 fields 7 of which are instantiations of non-trivial classes.

As to the complexity of individual calls, the story gets grimmer. Consider the function CreateLetterContent of the Document class (next page), which has 31 parameters. Eight of these parameters are of the notoriously treacherous Variant type. Sixteen are String, one is Long (signed 32-bit integer), one is Boolean, and one is Single. Finally, three are of different enumerated types that can take three to four values each.

```
Function CreateLetterContent (DateFormat As String, IncludeHeaderFooter
As Boolean, PageDesign As String, LetterStyle As WdLetterStyle,
Letterhead As Boolean, LetterheadLocation As WdLetterheadLocation,
LetterheadSize As Single, RecipientName As String, RecipientAddress As
String, Salutation As String, SalutationType As WdSalutationType,
RecipientReference As String, MailingInstructions As String,
AttentionLine As String, Subject As String, CCList As String,
ReturnAddress As String, SenderName As String, Closing As String,
SenderCompany As String, SenderJobTitle As String, SenderInitials As
String, EnclosureNumber As Long, [InfoBlock], [RecipientCode],
[RecipientGender], [ReturnAddressShortForm], [SenderCity], [SenderCode],
[SenderGender], [SenderReference]) As LetterContent
```

Figure 1: The CreateLetterContent function

At this point, some questions pose themselves. Designing tests is essentially designing sequences of API calls that have a potential of satisfying the test objectives. This in turn boils down to designing each call with specific parameters and to building a mechanism for handling and evaluating return values. This translates in the case of CreateLetterContent to making thirty-one decisions: which value should a parameter take? In some cases, the number of choices is small such as in Boolean and enumerated types, but even those can be rough to handle; WdTextureIndex and WdTableFormat enumerated types have dozens of values each. What about integers, floats, and strings? What about variants, which can take any user-defined type? What about parameters that are complicated objects? How should those be populated?

The values of all the parameters need to be determined in order to have a syntactically valid call. What values make sense together? What recipe of parameters will make the call exercise the API's functionality in a desired manner? What combination will cause a failure, a bad return value, or an anomaly in the operating environment?

Suppose those questions are addressed in the design of individual calls. Now consider the sequence of calls, and the same questions will repeat themselves. With the number of calls in the order of thousands in the Word object model, the number of possible sequences, even with reasonable limits on sequence length, is unmanageable. Which sequences are the best candidates for selection?

Those are fundamental questions of testing in general. They are particularly hard to answer, as the magnitude of the corresponding problems is extraordinary in the case of APIs. In this work, we concentrate on three of the problems that we just pointed out:

- **Parameter selection** – choosing values of individual parameters
- **Parameter combination** – picking out interesting parameter combinations for calls with multiple parameters
- **Call sequencing** – deciding the order in which to make the calls to force the API to exhibit its functionality

1.3 Outside the Boundaries of Technology

There are many difficulties that can arise from project settings, personnel skills, and that are not necessarily related to the nature of APIs or the technologies employed in testing them. Here are some issues that may impede productivity and progress.

- *Inadequate domain knowledge* – Testers may not be well trained in using the API. Considering the difficulty in exploring these interfaces, they would have to spend some substantial time to learn about the API. This problem can be partially solved with involving the testers from day one starting with design and throughout the process.
- *Poor documentation* – Without the proper documentation that clarifies the purpose of calls, their parameter types and legal values, their return values, the calls it makes to other functions, and usage scenarios, designing tests can be a nightmare.
- *Unavailability of source code* – This may hinder efforts to diagnose anomalous behavior. In addition, understanding how certain functions are implemented may reveal some vulnerability that can be exploited during test.
- *Time constraints* – Thorough testing of APIs is time consuming and requires a learning overhead and resources to develop tools and design tests. Keeping up with deadlines and ship dates may become a nightmare.

1.4 About this Work

This work amounts to a small but first step toward studying API testing concerns. We give a general discussion of problems and remedies, but by no means have we started out with the intention of a comprehensive study. Our recommendations are based on our experience in testing some programmable interfaces. However, the data we have collected over a few months is limited and inconclusive, and the door is wide open for improvement.

In the remainder of this paper, we explore in more detail the three problems defined in section 1.2. In the following section, several suggestions that require little tool support are presented and discussed. A complete section is dedicated to automation: when it makes sense, what kinds of automation there are, and how to build one. Alternative styles that do not require automation are proposed in addition to automation or a cheap effective alternative.

2. Nuts That Are Hard to Crack

The first step to effectively test any interface is to identify and study its points of entry. In the case of GUIs, these are such items as menus, button, check boxes, and combo lists. For APIs, the points of entry are the provided routines and their input parameters. Subsequently, a chief task is to analyze the points of entry as well as significant output items. In particular, we look at the valid and invalid values of the parameters and return values. At this point, it is desirable to also perform boundary analysis [11] on all the variables in concern. A third step is to understand the purpose of the routines, the contexts they are expected to be used in, and the situations for which no behavior is known a priori. Once all this information is gathered, understood, and preferably

documented, parameter selections and combinations need to be designed, and different call sequences need to be explored.

2.1 Input/Parameter Selection

Selecting "interesting" values for input parameters tends to be a little more difficult for APIs than other interfaces since call arguments often constitute complex data structures. Therefore, not only are testers required to understand the calls, but also all the constants and data types used by the interface. A good example taken from [12] is presented in figure 2.

In figure 2, the call **GetCommState** has two input parameters. The first is straightforward to examine. The second is a pointer to a C structure with 28 fields. Selecting values for the two parameters is essentially choosing 29 actual values, and that is not simple. In order for the values to be valid test cases, the choices must vary with the context in which the call is made. For each of the variables, boundary, critical, and other risky values must be identified.

There is more to selection than that. Frequently, a value gains significance based on its internal usage as opposed to what is superficially visible. In such situations, category partitioning from a purely black box perspective becomes troublesome. Ideally, API testers should have access to source code, allowing for better analysis of problematic input values than is achievable by reading documentation and functional specifications.

2.2 Parameter Combinations

The number of possible combinations of parameters for each call is typically large. Even if only the boundary values have been selected, the number of combinations, while relatively diminished, may still be prohibitively large. In the case of the problem in figure 2, even with only two values selected for each parameter, there would be hundreds of millions of applicable combinations.

Parameter combinations are extremely important for exercising stored data and computation. In API calls, two independently valid values might cause a fault when used together. Therefore, a routine called with two parameters requires selection of values for one based on the value chosen for the other. Often the response of a routine to certain data combinations is incorrectly programmed due to the underlying complex logic.

Parameter combination is further complicated by the function overloading capabilities of many modern programming languages. It is important to isolate the differences between such functions and take into account that their use is context driven.

```
typedef struct _DCB {
    DWORD DCBlength;          /* sizeof(DCB)                     */
    DWORD BaudRate;           /* Baudrate at which running       */
    DWORD fBinary: 1;         /* Binary Mode (skip EOF check)    */
    DWORD fParity: 1;         /* Enable parity checking          */
    DWORD fOutxCtsFlow:1;     /* CTS handshaking on output       */
    DWORD fOutxDsrFlow:1;     /* DSR handshaking on output       */
    DWORD fDtrControl:2;      /* DTR Flow control                */
    DWORD fDsrSensitivity:1;  /* DSR Sensitivity              */
    DWORD fTXContinueOnXoff: 1;     /* Continue TX when Xoff sent */
    DWORD fOutX: 1;           /* Enable output X-ON/X-OFF        */
    DWORD fInX: 1;            /* Enable input X-ON/X-OFF         */
    DWORD fErrorChar: 1;      /* Enable Err Replacement          */
    DWORD fNull: 1;           /* Enable Null stripping           */
    DWORD fRtsControl:2;      /* Rts Flow control                */
    DWORD fAbortOnError:1;    /* Abort all reads and writes on Error */
    DWORD fDummy2:17;         /* Reserved                        */
    WORD wReserved;           /* Not currently used              */
    WORD XonLim;              /* Transmit X-ON threshold         */
    WORD XoffLim;             /* Transmit X-OFF threshold        */
    BYTE ByteSize;            /* Number of bits/byte, 4-8        */
    BYTE Parity;              /* 0-4=None,Odd,Even,Mark,Space    */
    BYTE StopBits;            /* 0,1,2 = 1, 1.5, 2               */
    char XonChar;             /* Tx and Rx X-ON character        */
    char XoffChar;            /* Tx and Rx X-OFF character       */
    char ErrorChar;           /* Error replacement char          */
    char EofChar;             /* End of Input character          */
    char EvtChar;             /* Received Event character        */
    WORD wReserved1;          /* Fill for now.                   */
} DCB, *LPDCB;[1]

Function Prototype:
WINBASEAPI BOOL WINAPI GetCommState(HANDLE hFile, LPDCB lpDCB);
```

Figure 2: A problem in parameter combination

2.3 Call Sequencing

When combinations of possible arguments to each individual call are unmanageable, the number of possible call sequences is infinite. Parameter selection and combination issues further complicate the problem call-sequencing problem. Faults caused by improper call sequences tend to give rise to some of the most dangerous problems in software. Most security vulnerabilities are caused by the execution of some such seemingly improbable sequences. An example of such a fault is taken from [2] and illustrated in the following table.

Time	Code Snippet	Exploit
1	if (access (“filename”, W_OK)!=0) exit (-1)	
2		rm filename
3		ln –s/etc/passwd filename
4	if ((fd=open(“filename”, O_WRONLY))-1) exit(-1)	
5	write(fd,”junk\n”,5);	

Table 1: Code snippet

Table 1 shows how the code snippet can cause a race condition, leaving it wide open for an exploit. It is important to realize that neither the **access()** nor the **open()** calls would independently cause a fault but used together can bring out a rather nasty security vulnerability.

3. Possible Solutions

The underlying thought to approaching and meeting each of the challenges can be expressed in the following points:

- *Review* and scrutinize the available documentation looking for the following:
  - ✓ List of calls, parameters and return values (DELIVERABLE)
  - ✓ List of valid/invalid parameters (DELIVERABLE)
  - ✓ Example usage (DELIVERABLE)
  - ✓ List realistic and common usage scenarios (DELIVERABLE)
- Map the interface (Partition/Categorize functionality) – Note: These criteria are subjective
  - ✓ Related objects
  - ✓ Related calls
  - ✓ Data types
  - ✓ Draw a map of the application interface (DELIVERABLE) – This is visually helpful
  - ✓ Pinpoint resources such as OS (DELIVERABLE)
- Review source code (if available)
  - ✓ Internal data structures
  - ✓ Shared data
  - ✓ Pointers
  - ✓ Binding (e.g. whether a DLL is statically or dynamically linked)
- Design and develop test automation if feasible

3.1 Reviewing Documentation

Reviewing accompanying documentation for the API being tested is an extremely useful exercise. Documentation is the only artifact of the software design process that truly reflects the original intent of the designer and is therefore, invaluable to the tester.

According to Whittaker [13], good documentation should reflect the following aspects of the design phase:

- All the tasks that the user wants to perform are represented in either transactions or sequences
- All input sequences that represent important input conditions and combinations are specified
- All the inputs to the software are fully specified, including the interface definitions that define how the inputs will be received. Thus, all hardware and device interfaces are specified as well as any graphical user interfaces for human users
- All the outputs that the software must generate are fully specified, including formats of reports

- All requirements that are subject to change are clearly marked

Unfortunately, API documentation, just like for other software, tends to be incomplete. Despite the lack of comprehensive documentation, it is still possible to extract important information from what little is provided. In particular, Whittaker's abstraction methods suggested for the "discovery phase" in [13] can be applied in gathering essential facts from the documentation:

- Highlight important terms. A good practice is to insert identifiers so that it is easy to refer back to important parts of the requirements
- Isolate transactions to find inputs
- Identify users
- Formulate an input list
- Analyze inputs through sequence analysis. In particular analyze each input or class of inputs to determine the conditions under which it generates results.
- Based on the last action, synthesize an output list

Figure 3, from [10], describes the LoadLibrary function call that is a part of the Microsoft Windows operating system kernel (kernel32.dll). While the document is not complete based on the requirements above, it does offer valuable information about how the LoadLibrary call works. The first artifact of value is the function definition along with inputs, return parameters and data types. The remarks section contains information on the purpose of the function and a list of functions that are commonly used with it (GetProcAddress, FindResource, LoadResource, and CreateProcess). Useful information such as the instability of the function if called from DllMain is included. Based on this data it is possible to create at least a preliminary list of transactions, inputs, users, sequences and outputs. As more information is gathered about other DLL calls in the kernel, the lists can be cross-referenced to draw and a more accurate picture of the API.

3.2 Mapping the Interface

Humans think visually. Mapping the programmatic interface into a diagram can be very helpful. The diagram not only helps the tester understand the various interactions with the interface but doubles as a quick reference later. A supplementary document to the interface diagram is the list of all the calls for a particular interface. Looking at this diagram along with the calls can be a very powerful tool to understand application behavior in detail.

This function maps the specified .DLL file into the address space of the calling process.

HINSTANCE LoadLibrary(LPCTSTR *lpLibFileName***);**

Parameters

lpLibFileName
Pointer to a null-terminated string that names the .DLL file. The name specified is the filename of the module and is not related to the name stored in the library module itself, as specified by the **LIBRARY** keyword in the module-definition (.DEF) file.
If the string specifies a path but the file does not exist in the specified directory, the function fails. When specifying a path, be sure to use backslashes (\), not forward slashes (/).
If the string does not specify a path, the function uses a standard search strategy to find the file. See the **Remarks** for more information.

Return Values

A handle to the module indicates success. NULL indicates failure. To get extended error information, call **GetLastError**.

Remarks

LoadLibrary can be used to map a DLL module and return a handle that can be used in **GetProcAddress** to get the address of a DLL function. **LoadLibrary** can also be used to map other executable modules. For example, the function can specify an .exe file to get a handle that can be used in **FindResource** or **LoadResource**. Do not use **LoadLibrary** to run a .exe file, use the **CreateProcess** function.
If the module is a DLL not already mapped for the calling process, the system calls the DLL's **DllMain** function with the DLL_PROCESS_ATTACH value. In Windows CE, a DLL is loaded once, but then it is mapped into each processes address space when a process implicitly or explicitly loads the library with the **LoadLibrary** function.
When Windows CE loads a DLL, all path information is ignored when determining if the DLL is already loaded. This means that a DLL with the same name but a different path can only be loaded once. In addition, a module ending with the extension ".CPL" is treated as if the extension if ".DLL".
It is not safe to call **LoadLibrary** from **DllMain**.
Module handles are not global or inheritable. A call to **LoadLibrary** by one process does not produce a handle that another process can use—for example, in calling **GetProcAddress**. The other process must make its own call to **LoadLibrary** for the module before calling **GetProcAddress**.
Two different modules cannot have the same filename, given that the extensions are different. These effectively have the same "module" name. For example, if **LoadLibrary** is made on "Sample.cpl", the operating system will not load Sample.cpl, but instead will again load Sample.dll. A similar limitation exists for modules with the same name but residing in different directories. For example, if **LoadLibrary** is called on "\\Windows\Sample.dll", and then **LoadLibrary** is called on "\\MyDir\Sample.dll", "\\Windows\Sample.dll" will simply be reloaded.
If no filename extension is specified in the *lpLibFileName* parameter, the default library extension .DLL is appended. However, the filename string can include a trailing point character (.) to indicate that the module name has no extension.

Figure 3: Documentation for the Windows Kernel LoadLibrary call

3.3 Reviewing Source Code

Reviewing source code is very valuable to a tester, especially when there is limited documentation available. Code listings give detailed information at a far greater depth than high-level documentation. However, the draw back is that this requires studying hundreds if not thousands of lines of code.

Referring again to the code listing in figure 2, we see that it gives a great deal of information that it's specification may not have shown. For example, it is unlikely that the high-level documentation for the function call GetCommState stated that the DCB structure is made up of 28 parameters. The code points out that the DCB structure can be referred to in multiple ways (DCB, *LPDCB). Looking at variable names helps relate high-level information (like the purpose of the GetCommState call) to lower-level structural and data type specific information. This is a good point to make a list of the data types used. This information is valuable when deciding on specific test cases later. Knowing the limits of specific data types helps choose parameter values that are likely to break software functionality through buffer overruns.

3.4 Isolating Common Calls & Parameters

Isolating common calls and their parameters is essential to testing programmable interfaces. Not knowing which calls are used more than others in an API is equivalent to not being aware of the commonly used user controls in a graphical interface.

For any software it is more important that the most commonly used functionality work better than features that are not used as often. Pareto's principle [5] (also called the 80/20 rule) suggests that 80 percent of users use 20 percent of a system. From a practical perspective, it makes more sense to concentrate on testing the most commonly used 20 percent of an application.

While it is not always straightforward to isolate the most often used calls without conducting usage studies, it is simple to prepare a short list of possible candidates based on the function of the software under test. For example, if the application being tested is an email API (such as the Microsoft Outlook object model), it is fair to assume that certain calls such as the compose command will be used more often than others.

Figure 4, from [8], contains a code listing that shows the use of the Microsoft Office 9.0 object library through Visual Basic to compose and then send an email message. Studying the code we can see that some of the commonly used calls for the Outlook mail object are CreateItem(), Subject(), Body() and Send(). These calls are very likely to be used on a regular basis for email messages.

Also, some commonly used parameters are: an email string for the Recipients() method, the heading for the Subject() method, and the message string for the Body() method. In this way, it is possible to make educated guesses and pinpoint commonly used functions in an API.

Source code analysis makes internal data structures transparent. These data structures are normally invisible from a black box perspective due to use of information hiding and other object oriented programming techniques.

3.5 Isolating Valid/Invalid Parameter and Return Values

After making a list of commonly used calls, parameters, and return values, determine possible valid and invalid inputs. These values are a little more difficult to determine than most other interfaces because they often involve complex data structures. However, using a combination of information from the design specification and source code analysis performed before, it is possible to isolate a set of valid and invalid call parameters and return values. Any one of many published methods can be used to partition the inputs.

```
Sub NewMailMessage()
    Dim ol As New Outlook.Application
    Dim ns As Outlook.NameSpace
     Dim newMail As Outlook.MailItem

    'Return a reference to the MAPI layer.
    Set ns = ol.GetNamespace("MAPI")

    'Create a new mail message item.
    Set newMail = ol.CreateItem(olMailItem)
    With newMail
        'Add the subject of the mail message.
        .Subject = "Training Information for October 1997"
        'Create some body text.
        .Body = "Here is the training information you requested:" & vbCrLf

        'Add a recipient and test to make sure that the
        'address is valid using the Resolve method.
        With .Recipients.Add("mindym@imginc.com")
            .Type = olTo
            If Not .Resolve Then
                MsgBox "Unable to resolve address.", vbInformation
                Exit Sub
            End If
        End With

        'Attach a file as a link with an icon.
        With .Attachments.Add _
            ("\\Training\training.xls", olByReference)
            .DisplayName = "Training info"
        End With

        'Send the mail message.
        .Send
    End With

    'Release memory.
    Set ol = Nothing
    Set ns = Nothing
    Set newMail = Nothing
End Sub
```

Figure 4: Creating and Sending an MS Outlook email Message [8]

A list of valid parameters is required to verify that the interface actually performs the tasks that it was designed for. While there is no method that ensures this behavior will be tested completely, using inputs that return quantifiable and verifiable results is the next best thing. Equivalence classes further increase the chances that behavior for a large set of input parameters will be tested and verified. Conversely, a list of invalid parameters is required to confirm that these inputs are not accepted by the application interface. This list can also be used to test for data type limits and boundaries.

3.6 Isolating Realistic Usage Scenarios

Testing of realistic scenarios is necessary to force interaction between individual features (or function calls). Usage of software in the real world exercises multiple features at the same time. For example, if a user were creating a Microsoft Word document, it is more than likely that he or she would simultaneously be using the basic text-editing feature, the formatting feature, the spelling and grammar checker, and the print preview features. Additionally, the user might have tables, charts and pictures embedded in the document. Combinations of these different features represent realistic usage scenarios. In the programmable interface world, usage scenarios exist in the form of multiple function calls in different sequences, or similar sequences that use different parameter values.

Usage scenarios can be determined in many ways. If the initial design method used was similar to the Rational 4+1 model, the usage scenarios would already be available as a formal requirements analysis and design document. If a UML based model of the design was created using visual tools such as Rational Rose, it becomes simpler to refer to the use case and the sequence execution diagrams to make a detailed list of usage scenarios.

Usage scenarios can be developed by studying the problem domain of the software under test. Understanding what tasks the software was built to perform is a good way to derive use cases. For example, if the software under test is a ftp program, an obvious usage scenario would be transferring a file from a server to a local machine. High-level usage scenarios can be refined into many lower-level scenarios. The previous scenario could be repeated with different file formats (binary, ASCII etc.)

3.7 Pinpointing Used Resources

Another important, if somewhat difficult aspect of testing programmable interfaces, is pinpointing used resources. All software applications rely on the operating system for resources such as memory allocation, disk space, networking etc. The availability of these resources is integral to the proper working of a software application.
Most applications inherently trust the operating system to deliver any resources that they need. However, non-availability of these resources is often the cause of software failure. Most testers are unaware of interactions between the software under test and the operating system. Those aware of these interactions, find it very difficult to test them due to their being invisible except to the best system programmers. It is important to isolate and differentiate between the various system resources provided to the application interface under test and understand how each resource affects the software.

Tools are now being developed that will soon make it possible to test system resources and simulate resource depletion or failure in a controlled manner. Tools such as Hostile Environment Application Tester (H.E.A.T) developed at the Center for Software Engineering Research (CSER) at Florida Tech allow the testing of applications on the system interface level. H.E.A.T is capable of intercepting system resources like memory allocation (LocalAlloc, malloc, GlobalAlloc), networking , disk space, COM interfaces to name just a few.

3.8 The Exploratory Stage

Sections 3.1 through 3.8 make up the exploratory stage of the API testing process. The procedures mentioned in this stage are general guidelines to follow while constructing test cases. Ideally, following stages 3.1 through 3.8 will bring to light a lot of information on the application under test. However, testers may selectively choose only those stages relevant to the project at hand.

4. What About Automation?

In addition to reaping valuable information on possible ways to exercise an API's boundaries and test its functionality, applying the techniques described in the previous section will typically reveal a fair amount of failures. Artifacts produced by the exploratory stage can be mapped into the test matrix for the API. After designing test cases according to these guidelines to satisfy particular objectives, automating the testing effort needs to be considered as means of boosting productivity.

4.1 When Does Automation Make Sense?

It is not clear whether automating a test suite is always a good idea, and, between advocate and opponent views, there is no clear answer. Whether automation is feasible and beneficial depends entirely on the project at hand. However, there are general rules of thumb that can be used to make such a decision.

- **Is there enough information about the system?**
  There can be no warrant for automation that finds little or no bugs. However, in order for an automation to find bugs, it is essential that enough information about the system under test be available to recognize a failure when one occurs. While it may be relatively easy for automation to discover system crashes with little or no information, it is hard to recognize subtler behavioral faults without knowing the expected response before hand.

  Further, it is necessary to gauge whether automation of your test suite is realistically possible. For example, will the automation be able to properly synchronize calls if the system under test is a real time API?

- **How much effort is involved and is it worthwhile?**
  Writing test automation takes time and effort, but is all the trouble worthwhile? The sources invested in automation efforts can alternately be employed in running several more manual tests. While the tests that need to be run repeatedly such as smoke tests

(also called Build Verification Tests or BVTs) have a legitimate case for automation, the case for other test scenarios is not as clear-cut.

Another matter of concern is whether the test automation will keep finding new bugs each time it is run, or whether it will encounter what Beizer calls the pesticide paradox [1]. The pesticide paradox can be avoided by developing intelligent test automation that can vary test sequences based on a behavior model of some kind. However, designing and developing this kind of automation requires extra thought and planning.

- **How expensive is it to maintain your automation?**
  The cost of maintaining hard coded test automation scripts can be prohibitive. Most test automation requires some sort of maintenance every now and then. If the application under test changes (due to altered requirements or for any other reason) it is necessary to reflect the changes in the automation to retain its effectiveness. For example, if a routine is originally developed to accept three parameters and is later changed to only accept two parameters, the automated tests will crash unless they are changed accordingly. This change will become especially expensive if the original author of the test script is no longer around. The new tester would have to dig through and understand the test code before the change could be made. Think of the untold man-hours that could have been spent running manual tests instead.

- **How effective and efficient is the automation?**
  Unfortunately, this question cannot be accurately answered ahead of making the decision to automate. There are almost no studies that address the issues of whether a technique finds bugs and whether it is efficient in doing so. The only way to get an accurate answer is to know about all the bugs before hand – making testing a moot point! Realistically, the results are obtained at the end of the projects and should be used in some manner to estimate whether automation will meet your bug count / quality expectations in the future.

4.2 Kinds of Automation

Test automation is of many different kinds. Technically, if a test case or scenario is run through a script or program it is an automated test – even if the test by itself is completely useless. Test automation can range from extremely dumb to extremely intelligent. The level of intelligence your automation has is entirely up to the tester. Here is a list of common kinds of test automation.

- **Capture-Replay Automation**
  This is the most common and basic kind of test automation possible. Capture-Replay automation involves creating a test script that mimics a test sequence defined by the tester. This can be achieved by either manually writing a test script, or having one generated by the use of a Capture-Replay tool (such as the record feature in Rational Visual Test). This kind of automation is repetitive, especially prone to the pesticide paradox and is usually hard to maintain.

- **Monkeys**
  Monkey testing involves writing "dumb" automation that randomly applies inputs to the application under test. The Monkey itself has no understanding of the inputs nor does it apply them in a specific order. For example, a monkey testing a GUI would be able to recognize controls on the screen such as buttons, combo boxes and edit boxes. It would then exercise these controls in a random manner not knowing if a specific input is even applicable at any given time. While Monkey testing is popular in some circles, it is very repetitive, hard to steer (random) and tends to find only the "low hanging fruit." Monkey automation tends to be more forgiving to changes in the test application as it is unaware of input context.

- **Intelligent Test Automation**
  A model of some kind always drives intelligent test automation. The model used could be formal (finite state machine, Markov chain, grammar), or informally based on certain characteristics of the system under test. Formal models such as state machines can be extremely powerful as a large body of literature in computer science and mathematics supports them. The drawback of such models however, is the need for a great deal of thought and planning to construct them. Formal models are difficult to build without details on how the target system is supposed to work.

  Little work has been done on using formal model-based testing methods for large APIs (an example of this work can be found in [4]). However, it is possible to build a general model based on certain aspects of the system in a less formal manner. For example, a model can be a composition of the most common scenarios used in an API. This would be simple to develop, as the information is already present in the artifacts from the exploratory stage.

  Intelligent automation tends to be a lot more flexible to change than normal automation, as the model is independent of the automation code. In addition, it is possible to run multiple test sequences and generate many different test cases without changing the automation code.

4.3 Building Automation

It is important to put a lot of thought into the design of test automation before it is developed. While it is not possible to entirely solve the problems with test automation mentioned above, it is important to minimize them through a little planning. Here are some characteristics of well-designed automation.

- **Modular**
  A modular design keeps the automation friendly to change. Moreover, modularity helps increase understandability of the test suite, which is valuable to future owners of the test scripts.
- **Flexible (easy to change)**
  A test script must be flexible enough to support change in the system under test. This change must be possible with minimal effort. Modularity in design works towards the separation of higher-level actions, and test sequences from lower level system inputs.

For example, a high-level action defined as, "start chat session with <server name>" does not have to be changed even if the underlying call to the chat application changes. The change can simply be made at another layer in the automation that defines the atomic call to the chat program itself.

- **Scalable (easy to extend)**
  Good automation design always considers the possibility of extending the functionality of the test automation to test other areas of the application. Scalability is also increased with modularity.
- **Understandable**
  It is integral that the automation be designed and developed to be understandable. Automation can be made more understandable in two ways – documentation and modularity. Detailed documentation of test scripts seldom exists due to time constraints and test schedules. However, separation of different aspects of the automation into distinct easily understandable entities is very useful towards alleviating this problem.

5. Conclusions

API testing is not an easy problem to solve and this paper missed many important issues. Questions need to be answered on test adequacy criteria (when to stop testing). Current model-based testing approaches do not solve the parameter selection/combination issues (even if they do solve the sequencing problem to an extent). Automated methods need to be developed to explore APIs under test. One approach could be the gathering of information through parsing of documentation. Some tools are already available that gather call information from objects (the object browser in Microsoft Visual Basic for example) but are extremely limited. The various testing methods need comparison for their effectiveness with respect to APIs.

Work on testing of Application Programming Interfaces is still in its fledgling stages. It is only recently that the testing community has begun to acknowledge the difference in challenges between testing APIs and other types of software. This paper has been an attempt to isolate and address three of these fundamental differences.

Acknowledgements

This work was partially supported by Microsoft, Inc. James A. Whittaker and Nadim Rabbani contributed some useful suggestions.

References

[1] Beizer, Boris. Software Testing Techniques. Second edition, International Thompson Publishers: 1990.

[2] Bowen, Thomas F. and Segal, Mark E. Remediation of application specific security vulnerabilities at runtime. *IEEE Software,* 17(5): 62, September/October 2000.

[3] Dijkstra, E.W. Structural programming. In *Software Engineering Techniques*, pages 84-88. Buxton and Randell, 1969.

[4] Jorgensen, Alan, and Whittaker, James A. An API testing method. In the Proceedings of the Software Testing Analysis & Review Conference (STAREAST 2000), May 2000.

[5] Juran, J.M, Gryna, F.M. Jr. and Bingham, F.M. *Quality Control Handbook.* Third edition, McGraw Hill, New York, 1979.
[6] Kaner, Cem, Falk, Jack, Nguyen, Hung Quoc. *Testing Computer Software*. Second edition, Wiley: April 1999.
[7] Kaner, Cem. The impossibility of complete testing. *Software QA*, vol. 4, 1997. Obtainable from http://www.kaner.com/imposs.htm, March 2001.
[8] Martin, Mindy. Automating Microsoft Outlook 98. MSDN Online. http://msdn.microsoft.com/library/techart/msdn_movs105.htm, March 2001.
[9] Microsoft, Inc. *Microsoft Developer Network (MSDN) Online*. URL: http://www.msdn.microsoft.com/, March 2001.
[10] Microsoft, Inc. MSDN Online. LoadLibrary http://msdn.microsoft.com/library/psdk/winbase/dll_1o8p.htm, March 2001.
[11] Richardson, D.J., and Clarke, L.A. Partition analysis: a method combining testing and verification. *IEEE Transactions on Software Engineering*, 11(12): 1477-1490, December 1985.
[12] Whittaker, James and Atkin, Steve. Software engineering is not enough. In review, *IEEE Software*, 2001.
[13] Whittaker, James. *Introduction to Software Engineering*, p28-32. SES Press 2000.
[14] Whittaker, James. Software's invisible users. To appear in *IEEE Software*, 2001.

QW2001 Paper 4W2

Mr. Phil Hollows
(RadView Software)

Best Practices in Web Performance Testing

Key Points

- Benefits of Component Testing before QA
- How to Manage Scalability and Verification Resources
- Reduce cost and speed the process with lifecycle verification

Presentation Abstract

For a business based on the Web, there is no substitute for the "e-assurance," or confidence in their Web site's performance and scalability achieved through a rapid developing, testing and deploying cycle. It must discover and resolve any problems or issues as early as possible and at each point in the development cycle to stay competitive. Its testing solutions must, therefore, be not only available to the development workgroup as a whole throughout the lifecycle but also be reliable enough to accurately and efficiently prepare for the 85 million users expected to be buying online in 2003 (according to Jupiter Communications). Having a combination of Internet speed, or "e-acceleration," and "e-assurance," or confidence in Web performance is fundamental. The best solutions to expedite e-business success combine comprehensive verification of the performance, scalability and integrity of their Web systems into a single, standard process that the entire team can access.

About the Author

Phil Hollows joined RadView in October 1999, bringing with him over 12 years of experience in software engineering and management consulting. Phil previously worked as the Technical Director for Kronos, Incorporated, where he was responsible for technical strategy and implementation for the Timekeeper Systems Division. Prior to that, Phil was the Managing Consultant in New York for the executive information system consultancy Metapraxis, Inc. Phil had previously worked for Metapraxis in the UK as both a developer and consultant, moving to the US in 1993. Phil holds an MA in Physics from The Queen's College, Oxford, England, and a Certified Diploma in Accounting and Finance from the UK's ACCA.

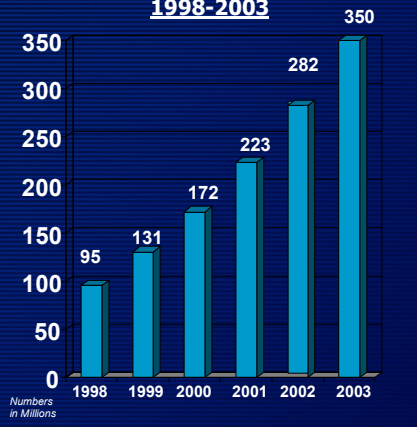
Best Practices in Web Performance Testing

Phil Hollows
VP Technology, RadView Software

RADVIEW

Background: Internet Growth

Worldwide Wired Internet Users
1998-2003



Source: eStats, 1999

Internet Growth & Acceptance

- E-business has become a widely accepted and maturing business model; a standard practice for transacting daily business
- Global wired Internet users will climb to 130 Million in 1999
- The number of Global Internet users is expected to Reach 350 Million by 2003
- The number of worldwide wired Internet Users in 2003 will still only represent about 5.7% of the world's total population
- Performance testing is not optional any more!

RADVIEW

2 / 2

Best Practices Overview

#1 – Start Now!

#2 – Test Realistically

#3 – Test Early, Test Often

#4 – Increase Collaborative Testing

#5– Beware the Capture / Playback Trap

#6– Maximize your RoI

#7– Understand the Production Environment

#8– Be Ready for Wireless and Broadband

Start Now!

- Competitors are one click away
  - You must be sure your site is fast enough
  - You must ensure that it is working correctly
- Starting small is better than not at all
  - Buy a smaller license and rent peak load testing
- Address software development practices
  - Performance and scalability testing will likely emphasize software engineering shortcomings
  - Take a big picture approach to quality
  - Building automated tests is a great way to find bugs
- Reap the rewards
  - Build better, faster applications more reliably
  - Reduce the risks of high profile failure

Test Realistically

- Emulate different user actions in a single test
  - Connection speed
  - Path through the site
  - Browser
- Use available history to model agendas
- Exceed best practices
  - No downloads longer than eight seconds
  - Test at three to four times historic peak loads
  - Verify application integrity as well as performance
  - Don't extrapolate!
- Define your key goals
- Build tests that can discover the causes of failure
  - Track system and server metrics
  - Compare your results to previous sessions

Test Early, Test Often

- The earlier a problem is found the faster and cheaper it is to address…
  - …yet many leave scalability testing to the end
- Embed scalability testing in development
  - Scalability test components prior to integration
  - Increase confidence early in the cycle
  - Improve cross-functional collaboration
  - Employ a single project-wide test solution
- Test well
  - Consider functional testing as a single user scalability test
  - Always verify correctness under load
  - Design tests to be robust to GUI and language changes
- Final load testing should be a true assurance exercise

Beware the Capture/Playback Trap

- Capture/playback commonly used far too late in development/testing cycle -- Capture/playback can be used effectively:
  - Excellent during usability testing
  - Quick and dirty, or one-time use scenarios
  - To 'seed' the creation of higher-level scripts
- Design higher-level, modular scripts for reusability
- Input data should not be hardcoded in scripts (read data from file, database, or spreadsheet instead)
- Expected results should also be kept in files readable by automated verification processes

Greg Arsenault – Vice President of Technology, AnyDay.com

Increase Collaborative Testing

- Place performance at the heart of your project
  - Simply providing functionality is no loner enough
  - Ensure performance release criteria are defined
  - Treat functional testing as a single user scalability test
  - Share your successes inside and outside the group
- Use life cycle testing approaches
  - Ship better components to testers
  - Reduce costs and cycle times
  - Improve communications and cooperation
- Solutions will make it easier to enable early and responsive testing

Maximize your RoI

- Industry ASP for 600 users is $49,000 (Newport Group)
- Share automation resources efficiently
  - Don't limit testing capabilities to the QA function
  - Increase return by encouraging life cycle usage
  - Deploy a solution that is economic across multiple users
- Cost of ownership is not just the software license
  - Efficient solutions reduce hardware needed to simulate users
  - Reduces TCO by lowering overall infrastructure investment
- Embed automation into the software process
  - Make performance a requirement, not an afterthought
  - Reduce regression risks
  - Discover more defects earlier in the production cycle

Understand the Production Environment

- Baseline your system for consistent metrics
  - Monitor your system after deployment
  - Validate your models against actual usage
  - Understand other applications already deployed
  - Be aware of other planned deployments
- Extend testing to post-deployment environments
  - Use vendor or xSP services for peak load testing
- Soak test your applications prior to deployment
  - Your application will be up continuously
  - Verify system longevity and durability
- Establish a plan for how to handle excessive load
  - When capacity is exceeded, how will it be handled?

New Technologies: The Internet and Wireless

- The Population of Internet Users will significantly expand with the growth of the wireless Internet.
- Gartner Group predicts that the mobile phone will be the most common Internet access device in the world with the total number of installed mobile phones exceeding one billion some time after 2003.
- IDC forecasts that the number of wireless device users with access to inbound and outbound information services and Internet messaging will increase a whopping 728% from 7.4 million in 1999 to 61.5 million by 2003 in the United States alone.
- In fact, some industry forecasters believe that application development initiatives for wireless applications will overshadow classic PC-browser-based application development initiatives within the next 3 to 4 years.

R A D V I E W

11 / 11

The Performance Affect of Wireless and Broadband

- Wireless usage will outpace fixed access
- What's the impact on an application?
  - More users connecting more often
  - Different traffic and resource consumption
  - 2.5G and 3G high speed networks coming
  - Significant usability challenges
- Broadband: letting multimedia loose
  - From flash banners to streaming video
  - Increases competition for computing and network resources
- Affects all aspects of application quality
  - Greater demands on applications already deployed
  - Testing for performance gets harder
  - What are your users expectations with multimedia?

R A D V I E W

12 / 12

QW2001 Paper 6W1

Mr. Mark Johnson
(Cadence Design Systems)

How Are You Going To Test All Those Configurations?

Key Points

- Testing configurations is hard, especially in the web-world
- You can get your configuration testing under control
- It is possible to cover many configurations with limited resources and time

Presentation Abstract

With all the combinations of products and environments (OSes, web browsers, etc.) configuration testing is even more important in the web-world. This paper presents how, when faced with added complexity from the web, we improved our configuration testing coverage with limited resources and time.

About the Author

Mark Johnson has over 25 years experience in software and hardware development and test. He has had many roles over those years and has come to realize he is most interested in helping individuals and teams be more productive and effective in their work.

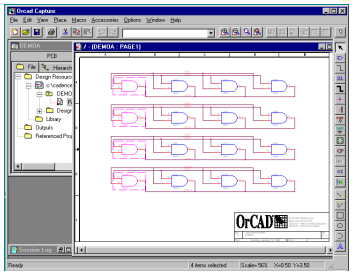
How Are You Going to Test All Those Configurations?

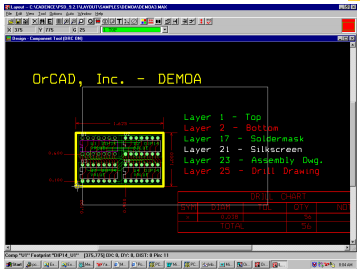
Mark Johnson
Cadence Design Systems

Agenda

- Background
- The Problem
- Source to CD Everyday
- Clean OSes by Imaging
- Planning and Managing Configurations
- Putting it All Together

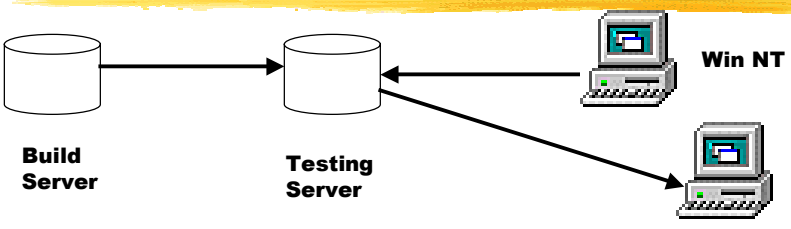
Ancient History





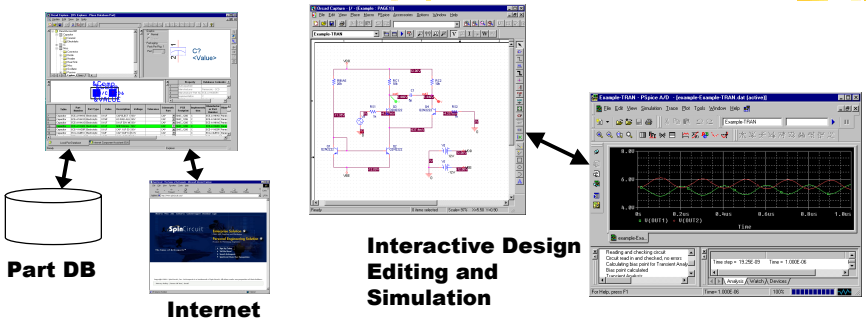
- The early days at OrCAD
- EDA software for the Windows PC market
- Small set of products, loose interfaces
- Products released separate from each other

The Old Testing Environment



- Nightly build of checked-in source
- Copy to server for test group use
- Testers assigned OS to use for release cycle
- Installation and Configuration test at the end

The World Starts to Change



- Add products needing tighter integration
  - Database and Internet access
  - Fast experiment/simulate cycle

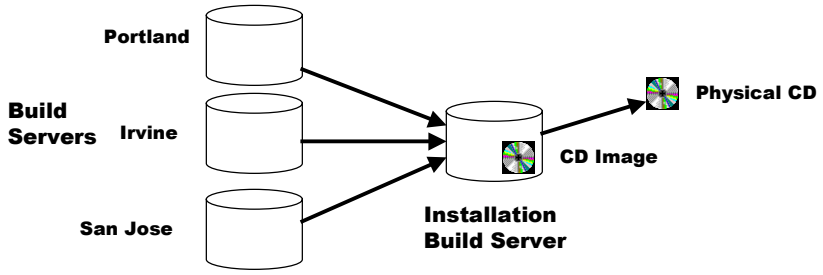
Our Testing Strategy

- Some testing on each configuration is good
- Don't need to do full testing on each configuration
- We are looking for installation/licensing problems with different configurations

Testing the Way a Customer Sees the Software

- Combine product releases onto one CD
  - With tighter interfaces, product changes need to be synchronized
- Decide to move to testing in a customer install environment
  - Find installation and configuration problems earlier
  - Testing needs matching versions of applications

Source to CD Everyday

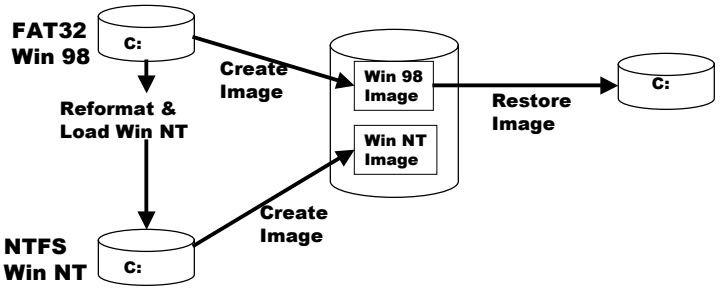


- Nightly product build produces executable files from latest checked in source
- Nightly installation build produces installable CD image

Knowing Where You are Starting From

- Concern about installing day after day to same system
  - registry entries accumulate
  - files that creep on or get lost
- Do periodic clean system setup
  - Reformat system, reinstall OS
  - Takes 6-8 hours
  - Only do every month or two

Clean OSes by Imaging



- Makes copy of hard drive at the track/sector level
- Fast restore puts you back to state when image was made

Problems with Imaging

- Hardware dependent (drivers) so aren't able to share
- Windows NT and 2000
  - Administrative rights, adding users to a system
  - Need a network admin to get system on network
- Cadence network: something happens periodically to cause images to stop connecting

More Changes

- Going to small teams
  - 1 or 2 testers per small team
  - 6-12 week release cycles
- Going to web updates
  - Test with previous web updates
  - Test with web updates to related products
  - Test with different web browsers

Getting to the Configurations

- Get 2/3 of the way through testing with only 1/3 of the configurations covered
- Habit - tend to do most comfortable thing
- Knowing what to change today
- Tracking configurations used so far

Planning and Managing the Configurations

- Make it part of test planning
- Identify the variables that are important to you
- Select the combinations you want to test
- Assign the configurations to testers
- Track progress testing the configurations

Table of Configurations

CD Testing Configurations

	Locking				Server/Client			OS				Browser		Products to Install												
CFG	NIC	Hasp	Rain bow	Inst	Svr	Cl	Oth	95	98	NT	2K	Net sc	IE	Cap	eCap	Demo	Cap+ AP	CIS	PS AD	PSP	PSA DB	Opt	Lay EE	Lay +	Lay
1	x										x		x	x									x		
2				x							x	x		x											
3		x								x		x	x				x			x					
4	x									x			x					x	x				x		x
5		x								x		x		x						x			x		
6			x			x	x	x					x					x				x				
7	x								x			x						x				x	x			
8		x			x						x	x					x		x				x		x	
9			x							x			x					x		x					x	
10				x						x			x	x									x			
11		x									x		x					x								
12			x								x		x					x	x				x	x		
13	x									x		x	x					x	x				x	x		
14			x				x				x	x				x										
15	x												x					x	x							x
16		x				x		x				x			x											
17			x		x				x				x				x			x			x		x	
18	x										x		x	x					x				x	x		
19		x									x	x			x											
20	x									x		x					x			x	x				x	

Configurations are not that Easy

- Restoring basic clean OS is easy
- Adding other things adds time
- Installing latest CD is easy
- Adding various web updates adds time
- Reality was 2 configuration changes per week

Results

- This has worked well for us
  - Finding installation and configuration problems early
  - Few installation or configuration problems are escaping
- 7 web updates and 3 CD releases over 1-1/2 year, while coordinating with 5 web updates to closely integrated products

The Future

- We are now moving to web-enabled applications
- User's system (client)
  - OSes
  - Web browsers
  - Add-on items like Java environment
  - Downloaded 3rd party plugins
- Create a set of standard client images

The Future

- Need to try multiple server configurations
- Create a set of baseline server images
  - Limited hardware availability, longer setup time
  - Clean OS images starting point for configurations
  - Add web server and database software, make image of each
  - May add standard test data sets and make more images
- With images, can switch configurations easily

Recommendations

- Plan and manage the configurations you need to test
- Establish a way to have clean testing environments
- Get to a customer-installable environment, available every day

How Are You Going to Test All Those Configurations?

Mark Johnson
Cadence Design Systems
Portland Oregon
503.968.4801
mark.johnson@cadence.com

Abstract

Whether you are testing a traditional application such as a word processor, or you are testing web-based software, the prospect of all the possible configurations that could be tested is a nightmare. We ran head-on into this issue when we started producing web installable updates to our products on an every-other-month basis. We had way too many combinations of OSes, browsers, licensing methods, product combinations, and other factors to try in the time available.

This paper describes techniques that we evolved which help us expand the set of configurations we test and increase our confidence that our products will function correctly on them.

- First, we refined our software development process to the state we call 'source to CD everyday.' This means that each night not only do we build the current development products, but we also produce an installable CD image.
- Second, the test group adopted a process involving a disk drive imaging product and the establishment of a set of known clean baseline systems. These baseline systems cover the different configurations of OSes, browsers, etc. we want to test. The drive imaging software allows each test engineer to quickly restore a test system to a known clean state to begin the next testing session.
- Third, we created a table of the configuration options we consider when testing. We use this table to assign unique testing configurations to each test engineer each time they restore a clean system. This allows us to maximize the number of configurations we test during any particular testing cycle, by guiding the test engineers to a wider set of configurations than they might choose on their own.

The combination of these techniques has allowed our project teams to increase coverage from 3 or 4 basic configurations to as many as 20 important combinations of configuration variables during a 2-month development cycle, with only 1 or 2 test engineers. We feel that this has been one of the key success factors for our product team to be able to provide frequent web installable updates.

The Problem

The OrCAD family of products is targeted at the Windows PC market. While the hardware has become fairly standardized, there can be a fair number of supported software configurations to test. Our primary concern with different software configurations has been installation and setup. We have not found that many problems with our products behaving differently on different software configurations. Our products

run on most recent Windows operating systems (OSes) and use a web browser to access the Internet, so we can have combinations of the following:

- Operating Systems
  - Windows 98
  - Windows NT 4.0 with service packs 4 or 6a
  - Windows 2000
  - Windows ME
- Web browsers
  - Netscape 4.x to 6.x
  - Internet Explorer 4.x to 5.x

In addition we can have multiple product combinations and licensing methods, depending on the installation options our customers choose.

On top of this, we were making changes in our software development process that would increase the possible configurations while at the same time reducing the testing resources and testing timeline. Traditionally our product development team had all worked together on the next big release, typically with a 6 to 9 month release cycle. We wanted to sub-divide into smaller teams that would work in parallel, each small team completing work on a specific feature area of the product in 6 to12 weeks. The work of each of these small teams is released on our website as a download that updates our existing product. The net result was that each small team would typically have 1 or occasionally 2 testing engineers, and they would need to cover the OSes, web browsers, various mixes of products, and now product updates, all in a shorter period of time.

Even with the old process of 6 to 9 month release cycles and 6 to 8 testers, doing a full testing pass on multiple configurations was out of the question for us. The approach we had adopted was to assign one or more testers to each major configuration. They would use their assigned configuration to do their normal testing work for the release. When we went to small teams, the 1 or 2 testers on a small team now had to cover the configurations themselves.

Fortunately over the period of time leading up to going to small teams, we had been evolving several processes that improved our ability to cope with this situation.

Source to CD Everyday

Our product development team had a tradition of doing a nightly build of the current baseline software. This let the developers know that the checked-in code would compile and link. Once or twice a week the test group would run a ‘smoke’ test on the build results. If the test passed the build would be copied onto a testing file server. Testers did their day-to-day testing using the software copy on the file server.

We did a limited amount of installation and configuration testing at the very end of a release, when the ‘golden’ CDs with the final software were available. This seemed sufficient because we only had 3 or 4 products, and we released them one at a time on separate CDs. Since all software development was done at one site and the group was small, any problems we ran into with installation and configuration testing could be fixed by talking to the right people and generating a new set of golden CDs.

Then we added two products that changed our integration needs. First, we purchased a small company located 500 miles from our office. Their product is an add-in to ours that allows the user to access information from databases and the Internet using database support software and a web browser. Then we merged with MicroSim, a company of equal size to OrCAD, located 1000 miles way. Their flagship product was the PSpice simulator. The typical user of PSpice is doing frequent cycles of experimentation during design, so there needed to be very tight integration between our design entry product and the PSpice simulator. With these changes we decided that all products should be released at the same time on the same CD.

The tight product integration made us realize we wanted to make a fundamental change to the way we set up our testing environment. As integration was developed between the products, we needed versions of each product in which changes were synchronized, so that we could test the integration features. With the tighter product integration, we decided that we wanted to begin doing our testing using configurations that would match a customer's installation.

To support this, we set up the process we call 'source to CD everyday.' We took the automated processes we had for creating the product CDs, and set them up to run after the nightly build. In this way, each morning we had an installable CD image we could use to load the current baseline software for testing. Getting to a CD image every morning was not as easy as it may sound, since we were now dealing with software development at 3 sites, one of which had a slow connection to the rest of the company. To accommodate the site with a slow connection, we ended up picking up their files as best we could each night, and using a previous copy if we couldn't get the latest files. This requires that the bill of materials (list of files and installation locations they are copied to) be maintained as development progresses. While this sometimes causes problems immediately after file organization changes are made, the benefit is that these problems are uncovered while the changes are still fresh in the developer's mind.

Because licensing and installing our software takes only 5 to 20 minutes, depending on the configuration chosen, testers are able to uninstall and reinstall our products pretty much every day prior to beginning that day's testing. This greatly helps us find problems with missing files, interface incompatibilities, etc. the day after they occur.

When we decided to go to small teams working in parallel, we were concerned about how to continue this 'source to CD everyday' process. The first thing that had to happen to support multiple small teams working in parallel on the same product was to set up separate branches for each small team's work. Our configuration management team supports this by establishing the new branches when they are needed, and starting up a separate nightly build process for each branch. Since the work of the small teams would be released as a web update, each small team would need its own web update building process. One of our developers took the tools our installation team used to create web updates and created a template web update and instructions for starting a new one. With this template, a newly formed small team spends about 4 hours during project startup putting their build process together. They then have an automated nightly build process that generates a web update and places it on an internal website to download for testing.

So now we have a ‘source to web update everyday’ process running for each small team. This allows the tester for that small team to do an uninstall and reinstall each morning, so that they are testing the latest version of their team’s software.

Clean OSes by Drive Imaging

Once we had started on the source to CD everyday process, we realized that we needed to have clean systems to support installation and configuration testing. Our definition of a ‘clean’ system is one which has not had our software installed on it. We had been depending on the uninstall process for our products to clean off any files, directories, registry entries, etc. We knew that the uninstall process was not perfect at cleaning everything up. The best way to ensure that we were not overlooking anything would be to start a testing session with a PC that had freshly formatted hard drives and a newly installed OS. This is what we call a ‘clean OS’ system. We would do this periodically, but it would typically take 4 to 8 hours for the tester to complete, so we were reluctant to spend the time more than once every few months.

In looking for a solution, we talked to our IT department. They use a utility program for setting up new PCs with our standard software configuration. Basically, it is a backup/restore utility that makes a copy (called image file) of the physical layout of the hard drive and allows you to restore the entire hard drive later. When IT gets a new set of PCs, they format and install the OS and other applications on one of the new PCs. Then they use the utility to make an image of the hard drive on that PC. For the rest of the PCs, they simply restore that image and the hard drive of each PC is an exact match of the first PC.

In testing, we realized that this utility might provide an easy way for us to rapidly restore a testing PC to a known state. It only takes about 5 minutes to restore an image file. When a tester receives a new PC they format and partition the hard drives, and load one of the OSes they want to test with, such as Windows 98. They make an image of this ‘clean OS’ and save it. To create another OS configuration for the same PC, they reformat the hard drives and install another OS, such as Windows NT. In this way, a tester can have as many different clean OS configurations as they desire. A nice feature of drive imaging is that it works at the physical level, so you don’t have to worry about the current contents of the hard disk or its file system format. For example, if you are changing from Windows 98 where the hard drives are formatted FAT32, to Windows NT where the hard drives are formatted NTFS, you simply restore the Windows NT image and the file system is changes along with the disk contents. In this way the tester can simply load the image for the OS they want to use next and they are ready to finish setting up the configuration.

This process works well for providing a known clean starting point for testing and for easily switching from one OS configuration to another. Because the testing group has a mix of different PCs that have been acquired over time, we are not able to make one set of images that all testers will use. This is due to the different drivers and hardware specific configuration information needed for the different PC types. So it takes a tester about 4 hours to set up each new image when they get a new PC. Fortunately, once the images are set up, they require little day-to-day maintenance. Since OrCAD has been

acquired by Cadence, we have run into a problem with how Cadence administers its networks. Something happens on an every couple of months basis that makes stored Windows NT and Windows 2000 images stop logging into the network. When this happens, we have to have a system administrator delete and re-add the system on the network, and then have to make a new image. This takes about one-half hour, but it can take a day or two to get the administrator's help.

With the use of clean OS images, we have been able to find a number of installation and configuration problems with our products that we otherwise might have missed. These are typically wrong versions of files, missing files, or missing or wrong registry entries. And even if you need to work around the problem today by manually copying a file or editing the registry, the next time you restore a clean OS image, you can determine if the problem has been corrected.

Planning and Managing Configurations

We now had an installation CD image every day from the latest software build, and a way for testers to easily set up a clean OS environment. However, we found that we weren't getting the coverage of different product installation and licensing configurations we wanted. It seems to be human nature that testers fall into routines with preferred configurations they use for testing. We also found that after a week or two of testing, it was difficult for a tester to provide a record of the exact details of each configuration they had used. So it was hard to know exactly which configurations had been covered and which remained to be tested.

To overcome this, we have added detailed configuration plans to the test plans we write. We put together a table with columns listing the different configuration variables to track. In the rows, we select specific combinations to try and assign a tester to each configuration number. The table also gives us a shorthand way to record configurations in our testing note. See Table 1 for an example set of our testing configurations from a past CD release. The testers on the small teams use the same type of a configuration table.

We choose combinations of configuration variables by trying to cover most pair-wise combinations. However, we skip some pairings to reduce the total number of configurations. We do this by dropping configurations that would repeat the use of configuration variables our customers are less likely to use. For example, the majority of our customers are now on Windows NT and starting to move to Windows 2000. So we drop some combinations that would use Windows 95 and select Windows NT instead. For the small team projects which are released as web updates, our concern is more with configurations having earlier web updates installed and using different web browsers, because these factors can directly affect the success of web update installation.

We have found that because of the number of variables involved in the typical configuration, it can take several hours to fully set up a configuration that uses less common options. Because of this, we have scaled back from testing a new configuration every day to testing two or three configurations per week per tester. On a small team with a single tester, we have found that 10 to 12 configurations can be covered during the testing part of an 8-week project.

Table 1: Example configurations table for testing a CD Release

CD Testing Configurations																								
	Locking				Server/Client			OS				Browser		Products to Install											
CFG	NIC	Hasp	Rain bow	Inst	Svr	Cl	Oth	95	98	NT	2 K	Net sc	IE	Cap	eCap	De mo	Cap+ AP	CIS	PS AD	PSP	PSA DB	Opt	Lay EE	Lay +	Lay
1	x										x		x	x									x		
2				x							x	x		x											
3		x								x		x	x				x			x					
4	x									x			x					x	x			x			x
5		x								x		x		x						x			x		
6			x			x	x	x					x					x			x				
7	x								x			x						x			x	x			
8		x			x						x	x					x		x			x		x	
9			x							x			x					x		x				x	
10				x						x			x	x									x		
11		x									x		x					x							
12			x								x		x					x	x			x	x		
13	x									x		x	x					x	x			x	x		
14			x				x				x	x				x									
15	x												x					x	x						x
16		x				x		x				x			x										
17			x		x				x				x				x			x		x		x	
18	x										x		x	x					x			x	x		
19		x									x	x			x										
20	x									x		x					x			x	x			x	

CFG – Configuration to test using all the options specified in that row.
Locking – What method is used to license the installed software.
Server/Client – Install the software in a stand-alone (no 'x' in the column) or in a client/server combination.
OS – Which OS to use.
Browser – Which Internet browser to use.
Products to Install – Install this combination of our products and updates.

Results

We have found the idea of providing some test coverage on a larger number of configurations, rather than more test coverage on a single or fewer configurations to be effective. Over the year and a half we have been doing small team projects and handling configuration testing in this manner, we have shipped 7 web updates and 3 CD releases. We have coordinated our testing with 5 web updates for the two products that closely integrate with our product. We have a high level of confidence that our product will install and perform properly in the wide range of environments we support.

During this period, we have found many installation and configuration related issues during our internal testing. These are typically missing files, wrong versions of files (something has been updated but not made it into the installation process), or occasionally conflicts between installations of two different products.

I am aware of only two configuration-related problems that have escaped from our testing. One was due to a last minute change made to fix a problem. It required adding a

file to the set that made up a web update. This change happened after the majority of configurations had been tested, so we missed an installation ordering dependency. We were able to address this by listing the installation order on the web site containing the web update download files. The other problem was a configuration management issue where files added to one web update did not get checked in correctly and were left out of the subsequent web update. Since our web updates are cumulative, as part of the testing for each web update we verify that the functionality added in any previous web updates is present. These two web updates happened almost simultaneously and we didn't verify the second web update had all the correct contents until it had been released. These files were quickly added to the second web update and it was re-released on the website for downloading.

The Future

The product we have worked on for the past several years is being transferred to another group for maintenance. We have begun developing web-based applications that integrate multiple products into flows and provide more generalized access to databases and the Internet. These applications are typical web style client/server applications. We are expecting to use these same processes once we get farther into development of these products.

Tracking the configurations we need to test will be very important, as the products will need to run with multiple web browsers, multiple OSes, now including UNIX and Linux, multiple web servers, and multiple database products.

Drive imaging will continue to be important for establishing baseline testing environments. For the client side, we will need to have configurations with various web browsers, Java environments, downloaded plug-ins, etc. On the server side, drive imaging will help in establishing the baseline testing configurations. For example, a server can be set up with Windows NT and an image made. This image can then be used as the starting point for creating more images that have specific web server and database combinations. This way when the tester wants to try a different server configuration, reloading will only take a few minutes.

We are still working to get the installation process defined for these new products. Once we have the installations created, we will want to get to the same point of having a complete installation built each night.

Recommendations

The process we developed is made up of a few basic steps that seem like good software development and testing practices:

- Plan the configurations you need to test, track your progress, and adjust your plans as needed.
- Establish a fast and efficient way to set up clean testing environments.
- Create a system that gives nightly builds of your software under development, with the results coming out in a form that allows you to install it as a customer would.

We would recommend this process of testing across as many user configurations as you can for anyone dealing with software installed in the end user's environment. This applies to products and web updates as we have been developing, as well as web-based applications we are now working on. We believe that it does not add appreciable overhead to your testing. We also feel that it will help you find more configuration-related problems and find them earlier in your testing cycle, resulting in happier customers and possibly shorter release cycles.

QW2001 Paper 6W2

Mr. Rakesh Agarwal, Mr. Santanu Banerjee & Mr. Bhaskar Gosh (Infosys Technologies Ltd)

Estimating Internet Based Projects: A Case Study

Key Points

- Estimation of Internet projects using Use case point approach
- How to estimate use case points based on technical and non technical factors
- Evolving the weights for all these factors and deriving an estimate based on these

Presentation Abstract

Software organizations are in need of methods to understand, structure, and improve the way estimation is done for internet-based applications. The effort of estimation required in developing good quality Web-sites/application is a difficult task. Accurate estimates play an important role in the success of web projects. Estimating for an Internet based project is difficult to define as there is no single model for effort estimation with the number of focus areas that drives the project. As there is no established method of estimation it is difficult to arrive at the total effort and hence the staffing and schedule is also not derived correctly.

In this paper we will take a real case study of an Internet project that we have executed for our client. The methods have been used in our project and found to be close to the actual effort details after execution using the User Use Case Point (UUCP) approach. The UUCP is a modification of the Function Points method of estimation. This should be considered as a means of arriving at a ballpark estimate of the effort involved in developing the system. The steps are explained below:

1. Weighting Actors: The process starts by considering the actors in the system. For each actor, determine whether it is simple, average or complex. A simple actor represents another system with a defined Application Programming Interface (API). An average actor is either another system that interacts through a protocol such as TCP/IP, or it is a person interacting through a text-based interface. A complex actor is a person interacting through a graphical user interface.
2. Weighting Use Cases: A similar process is followed for use cases. Used use cases or extending use cases do not need to be considered. For each use case determine whether it is simple, average, or complex. The basis of this decision is the number of transactions in a use case, including secondary scenarios. For this

purpose, a transaction is defined to be an atomic set of activities, which is either performed entirely or not at all. A simple use case has 3 or fewer transactions, an average use case has 4 to 7 transactions, and a complex use case has more than 7 transactions.
3. Weighting Technical Factors: Start by calculating the technical complexity of the project. This is called the technical complexity factor (TCF). To calculate the TCF, go through the following table and rate each factor from 0 to 5. A rating of 0 means the factor is irrelevant for this project, 5 means it is essential. Now, for each factor multiply its rating by its weight from the table. Finally add together all these numbers to get the total T factors. The equation for arriving at the estimate is:

TFactor = ...(Tlevel) * (Weighting Factor)
TCF = 0.6 + (0.01 * TFactor)
The above method has been tested on some of the projects executed by us and we have found that the variance between the actual effort and the estimated effort is close to 15%. The paper will elaborate the details of this methodology.

About the Author

Santanu Banerjee is a project leaders at Infosys Technologies Limited, India. He has been working on various web projects and currently involved in the design and development of a complete investment portal.

Rakesh Agarwal is working with Infosys Technologies Limited, India, in the Education and Research Department for the past 3 years. He has published more than 60 papers in leading conferences and Journals.

Bhaskar Ghosh is working as Associate Vice President in Infosys Technologies Limited, India for the past 4 years. He has lead many projects in his carrier and Heads one of the Development Centers of Infosys.

Estimating Internet based projects – A Case Study

Banerjee.S, Agarwal.R and Ghosh.B
Infosys Technolgies Ltd
India

Agenda

- Introduction
- Background study
- About the approach
- Conclusion

Introduction

- Estimating for a internet based project is difficult to define a single model for effort estimation
- Number of focus areas that drive the project
- No established method of estimation it is difficult to arrive at the total effort
- We will discuss on an approach which we have adopted for number of our projects and have found that the estimated effort is close to actual effort.

Use case Point Approach

- The Use Case Point Approach, from Rational Software, is a modification of the Function Points method of estimation
- This can be used to arrive at ballpark estimates of web based projects .

How to use Use Case Point Approach

- Determine the actors of the system and find the weights of these actors .
- Determine the use cases and find the weights of these use cases – these give the type of interaction between the actors .
- The use cases can be classified and weighted based on its description and the number of analysis classes they correspond to.

Use case Point Approach

- The technical complexity of the project is determined and technical complexity factor is computed by

  TFactor = $\sum$(Tlevel) * (Weighting Factor)

  TCF = 0.6 + (0.01 * TFactor)

- TCF stands for the technical complexity factor .

Use case Point Approach

- Environmental factors are also considered for computing the total use case points.
- For details of EF and TCF calculation refer the paper.

Use case Point Approach

- The Total Use case Points =

  UCP = UUCP * TCF* EF

  UUCP stands for the user use case points .

Guideline for estimating projects using this approach

- UCP method suggests usage of 20 person-hours per UCP for a project estimate.
- Our experience says that for an UCP we can use 18-22 person hours for project estimate depending upon the experience of the resources
- For other details , please refer the detailed contents of the paper .

Conclusion

- Estimation of Internet projects is a challenge
- Using Use Case Point can be one of the approaches to arrive at the estimates, and we have found the estimates derived by this approach is close to the actual.
- The approach needs to be tested across different types of projects and the values can be further refined based on the experiences.

Thanks

Estimating Internet based Projects: A Case Study

Santanu Banerjee, Rakesh Agarwal and Bhaskar Ghosh
Infosys Technologies Ltd., Near Planetarium, N.H.5,
Bhubaneswar - 751013, India
Email: {santanub}{rakesh_a}@infy.com

Abstract

Software estimating for Internet based projects is an important concern for software managers and other software professionals. The model in this research suggests that an organization's use of an estimate influences its estimating practices that influence both the basis of the estimating process and the accuracy of the estimate. The model also suggests that the estimating basis directly influences the accuracy of the estimate. In this paper we will take a real case study of an Internet project, which we have executed, for one of our clients

Keywords: Web-based projects, Estimation, Use case

1. Introduction

Software organizations are in the need of the methods to understand, structure and improve of the way estimation is done for Internet based applications. The estimation of the effort required to develop a good quality website/web based application is a difficult task. Accurate estimates play an important role in success of these projects. Estimating for an Internet based project is difficult, to define as there is no single model for effort estimation, there are number of focus areas that drive the project. As there is no established method of estimation it is difficult to arrive at the total effort and hence the staffing and schedule is also not derived correctly.

Lack of proper estimation models lead to effort overruns and wrong price estimates. Statistics shows that 54% of the known Internet based projects have completed after schedule with more expenditure than estimated.

In this paper we will take a real case study of an Internet project, which we have executed, for one of our clients. In this project we have used ***UserUse Case Point Approach (UUCP)*** for estimation and have found that the actual effort is close to the estimates. The outline of the method is described in this paper.

2. User Use Case Point Approach

The Use Case Points Approach, was put forward by Rational Software, and is a modification of the Function Points method of estimation widely used in the software industry. The User Use Case is similar to the Use Cases with the special emphasis on the user events of the system under consideration. This method can be considered as a means of arriving at a ballpark estimate of the effort involved in developing the system.

In this approach first the actors of the system are identified, the actors are the entities which can identify themselves in the system. The weights of these actors are determined

depending on the type of the actors. Then the user use cases are then determined, the user use cases are shows the event of interaction of the actors focussed mainly on user events for example user entering the employee details which initiates the search. These use cases are then weighted with factors depending on the type of the use cases.

Depending on these factors, the Use Case Points are arrived at. The technical complexity and the environmental factors of the project are factored as TCF and EF. The project estimate can be arrived from these Use Case Points.

3. Weighting Actors

The process starts by considering the actors in the system. For each actor, determine whether it is simple, average or complex. A simple actor represents another system with a defined Application Programming Interface (API). An average actor is either another system that interacts through a protocol such as TCP/IP, or it is a person interacting through a text-based interface. A complex actor is a person interacting through a graphical user interface.

Count how many of each kind of actors are present in the system. Multiply by weighting factor. Add these products together to get a total.

Actor Type	Description	Factor
Simple	Program interface	5
Average	Interactive, or protocol driven interface	10
Complex	Graphical interface	15

4. Weighting Use Cases

A similar process is followed for use cases. Used use cases or extending use cases do not need to be considered. For each use case determine whether it is simple, average, or complex. The basis of this decision is the number of transactions in a use case, including secondary scenarios. For this purpose, a transaction is defined to be an atomic set of activities, which is either performed entirely or not at all. A simple use case has 3 or fewer transactions, an average use case has 4 to 7 transactions, and a complex use case has more than 7 transactions.

If analysis classes have been defined for the system and it has also been identified as to which ones are used to implement a particular use case, use this information in place of transactions to determine the use case complexity.

4.1 By Use Case description

Use Case Type	Description	Factor
Simple	3 or fewer transactions	5
Average	4-7 transactions	10
Complex	>7 transactions	15

4.2 By Analysis Classes

Use Case Type	Description	Factor
Simple	Fewer than 5 analysis classes	5
Average	5-10 analysis classes	10
Complex	More than 10 analysis classes	15

Count how many of each kind of use case are present. Then multiply each type by the weighting factor specified in the given table. Add these products to get a total.

Add the total for actors to the total for use cases to get the unadjusted use case points. This raw number will be adjusted to reflect the project's complexity and experience of the people on the project.

5. Weighting Technical Factors

Start by calculating the technical complexity of the project. This is called the technical complexity factor (TCF). To calculate the TCF, go through the following table and rate each factor from 0 to 5. A rating of 0 means the factor is irrelevant for this project, 5 means it is essential. Now, for each factor multiply its rating by its weight from the table. Finally add together all these numbers to get the total T factors.

TFactor = ∑(Tlevel) * (Weighting Factor)
TCF = 0.6 + (0.01 * TFactor)

5.1 Technical Factors for System and Weights

Factor	Factor Description	Weight
T1	Distributed system	2
T2	Response or throughput performance objectives	1
T3	End-user efficiency (online)	1
T4	Complex internal processing	1
T5	Code must be reusable	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Includes special security features	1
T12	Provides direct access for third parties	1
T13	Special user training facilities required	1

Consider the experience level of the people on the project. This is called the environment factor (EF). To calculate EF, go through the table below and rate each factor from 0 to 5. For factors F1-F4, 0 means no experience in the subject, 5 means expert, 3 means average. For F5, 0 means no motivation on the project, 5 means high motivation, 3 means average. For F6, 0 means extremely unstable requirements, 5 means unchanging requirements, 3 means average. For F7, 0 means no part-time technical staff, 5 means all part-time staff, 3 means average. For F8, 0 means easy –to-use programming language, 5 means very difficult programming language, 3 means average.

For each factor, multiply it's rating by its weight from the table given below. Finally, add all the numbers together to get the total E Factors.

EFactor = å(Flevel) * (Weighting Factor)
EF = 1.4 + (-0.03 * EFactor)

5.2 Environmental Factors for Team and Weights

Factor	Factor Description	Weight
F1	Familiar with internet process	1.5
F2	Application experience	0.5
F3	Object –oriented experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Stable requirements	2
F7	Part-time workers	-1
F8	Difficult programming language	-1

6. Use Case Points

Finally calculate use case points (UCP)

UCP = Unadjusted UCP * TCF* EF

7. Project Estimate

We have found that 20 person-hours per UCP is a good estimate for a project. But a close examination of the data suggests a refinement can be done based on our experiences. To obtain the project estimate count how many factor of F1-F6 are below 3 and how many factors in F7-F8 are above 3. If the total is 2 or less, then use 20 person-hours per UCP. If the total is 3 or 4, then use 28 person-hours per UCP. If the total is 5 or more, try to make changes to the project so the numbers can be adjusted. We have experienced that otherwise the risk of failure is quite high.

By following this we have found that the estimates are very close to the actuals. The % of deviation between estimates and actual effort is less than 20%.

8. Conclusion

Estimation of Internet projects is a challenge as of now as there is no single estimation model in which we can fit things, this leads to cost and schedule overruns. Using Use Case Point can be one of the approaches to arrive at the estimates, and we have found the estimates derived by this approach is close to the actuals. The approach needs to be tested across different types of projects and the values can be further refined based on the experiences.

9. References

[1] Boehm, B.W., Software Engineering Economics. Prentice-Hall: Englewood Cliffs, NJ, 1981.

[2] Cowderoy, A.J.C. and J.O Jenkins, 'Cost estimation by analogy as a good management practice', in Proc. Software Engineering 88, ed. Pyle, I.C., Liverpool: IEE/BCS, pp80-84, 1988

[3] DeMarco, T., Controlling Software Projects. Management, measurement and estimation. Yourdon Press: NY, 1982.

[4] Fenton, N.E., 'Software Metrics: a rigorous approach'. Chapman & Hall, 1991.

[5] Fenton, N.E. and S. Pfleeger, 'Software Metrics: a rigorous and practical approach'. Thomson Computer Press, 1997.

[6] Heemstra, F.J., 'Software cost estimation', Information & Softw. Technol., 34(10), pp627-639, 1992.

[7] Hughes, R.T., 'Expert judgement as an estimating method', Information & Softw. Technol., 38(2), pp67-75, 1996.

[8] Jack R. and M. Mannion, 'Improving the software cost estimation process', Software Quality Management, 1995 1 pp245-56.

[9] Jalote Pankaj, 'CMM in practice: processes foe executing software projects at Infosys' Addison-Wesley 1999

[10] Kemerer, C.F., 'An empirical validation of software cost estimation models', CACM, 36(2), 1993.

[11] Kitchenham, B.A., 'Empirical studies of assumptions that underlie software cost estimation'. Information and Softw. Technol., 34(4), 211-18, 1992.

[12] Londeix, B., Cost Estimation for Software Development. Addison-Wesley: Workingham, 1987.

[13] Londeix, B., 'Aspects of estimation practice in software development', in Proc. Software Engineering 88, ed. Pyle, I.C., Liverpool: IEE/BCS, pp 75-79, 1988

[14] Low, G.C and D.R. Jeffery, 'Function points in the estimation and evaluation of the software process', IEEE Trans. on Softw. Eng., 16(1), 64-71, 1990.

[15] Low, G.C. and D.R. Jeffery, 'Calibrating estimation tools for software development', Softw. Eng. J., 5(4), pp215-221, 1990.

[16] McDermid, J.A., Software Engineer's Reference Book, Butterworth-Heinemann: Oxford, UK, 1991.

[17] Pengelly, A., 'Performance of effort estimating techniques in current development environments', Softw. Eng. J.l, September 1995, pp162-169

[18] Putnam, L.H., 'A general empirical solution to the macro software sizing and estimating problem'. IEEE Trans. on Softw. Eng., 4(4), 345-61, 1978.

[19] Roetzheim W.H. and Beasley R.A., ’Software project cost & schedule estimating: best practices’, Prentice Hall, Inc, 1998.
[20] Shepperd, M.J., Foundations of Software Measurement. Prentice Hall: Hemel Hempstead, UK, 1995.
[21] Shepperd, M.J., C. Schofield and B.A. Kitchenham. 'Effort estimation using analogy', in Proc. 18th Intl. Conf. on Softw. Eng. Berlin: IEEE Computer Press, 1996.
[22] Symons, C.R., Software Sizing and Estimating. Mk II FPA, John Wiley: Chichester, 1991.
[23] Vigder, M.R. and A.W. Kark, 'Software Cost Estimation and Control, February 1994, Report available from the link.
[24] Walston, C.E. and C.P. Felix, 'A method of programming measurement and estimation', IBM Syst. J., 16(1), 54-73, 1977.

QW2001 Paper 7W1

Mr. Bhushan Gupta, Mr. Steve Rhodes
(Hewlett-Packard Co.)

Adopting A Lifecycle For Developing Web Based Applications

Key Points

- Web Development
- Lifecycle
- Process Improvement

Presentation Abstract

Web-based applications pose unique challenges to developers: “first to market”, “marketing the right product or service”, and “making the product sticky”. To “make the product sticky”, that is, to keep customers inside your application and keep them coming back, it is important that the product delivers high value, all the time, with acceptable quality. Under the constraints of scope, schedule, and resources, quality is often sacrificed. A product must have an optimal mix of value, stickiness, and quality to survive in the web marketplace.

Web development is in its infancy. Often developers are faced with difficult choices between languages, tools, and hardware. Most web development tools are in their Beta phase and have no user support. Development complexity increases in the open source environment where the ownership of the tools is not clearly defined. Adequate testing of a web application for usability and scalability further insures product “stickiness” and thus is very critical.

Our products and services are at increasing risk in the Internet age:
* We have new markets with inexperienced providers and customers.
* Our customers are moving targets.
* Accelerated time-to-market stresses our seasoned product generation processes.
* The competition forces us to pull up release schedules by weeks or months.
* Customer tolerance for defects has dropped precipitously. Customers can and will go elsewhere if we fail to meet their expectations.

We need new processes that are "light," nimble, and flexible to the competing pressures of resources, time, and delivered features.

In an effort to choose a suitable lifecycle for its Web-based development, Hewlett-Packard evaluated widely used software lifecycles. Iterative lifecycles seem well suited to Internet speed as they allow constant user feedback and repetitive quality assessment. In particular, development teams across

Hewlett-Packard and Agilent have switched to an Evolutionary lifecycle (EVO) for its strengths in user feedback, risk analysis and mitigation, and product releasability to meet “first to market" challenges. Results range from "schedule visibility" to "3X productivity" and "best in class." Although, EVO requires more attention to process than other lifecycles, it provides an early focus on every aspect of product development and delivery.

Of the various iterative lifecycles, EVO is best suited to managing all the risks cited above - simultaneously. It does so through a discipline of continuous process improvement, at short regular cycles. Plan, do, check, act . . . repeat! The target customer moves, we move; our customer learns, we learn. We can deliver features and quality at the scheduled release date, or prior to that date should circumstances demand it. Process maturity continues to advance cycle by cycle; but with EVO the cycles are weeks vs. months, and we mature those processes that net the biggest return first. EVO adjusts the clockspeed of our product lifecycle to the clockspeed of the marketplace, allowing us to avoid the Darwinian fate of organisms and organizations that can't keep up.

Our experience shows that EVO requires an early and adequate emphasis on the entire project planning as compared to other software lifecycles. The user feedback in individual EVO cycles clearly distinguishes EVO from other lifecycles. A work product must meet minimum quality requirements for an adequate user feedback. To meet this requirement, the development team must put in place the build, test, release and customer support processes as early as possible. While it sounds like additional work early in the development phase, it provides an opportunity for changes and improvements as the project scope grows. We have also noted that a well orchestrated decision making process is essential to maintain focus on the customer value proposition.

About the Author

Bhushan Gupta has been a Software Quality Engineer with the Vancouver Printer Division of Hewlett-Packard Company since June 1997. Recently he has moved to Commercial Publishing Division as a software development engineer and is pursuing software process architecture activities.

Bhushan Gupta was a faculty member in the Software Engineering Department of the Oregon Institute of Technology from 1985 to 1995. He has a MS degree in Computer Science from New Mexico Institute of Mining and Technology, Socorro, New Mexico and is a member of American Society for Quality.

Steve Rhodes is an internal consultant for Hewlett-Packard Company, developing and delivering key process technologies into HP's R&D organization. His primary areas of delivery are product lifecycle, software architecture, and configuration management. He has been with Hewlett-Packard for 20 years, as developer, manager, architect, and consultant. Steve has an MS of Computer Science from the University of California, San Diego, and BS in Mathematics from Northwestern University of Louisiana.

EVO for the Web

Bhushan Gupta and Steve Rhodes
Hewlett-Packard Company

14th International Conference on Internet & Software Quality
Quality Week 2001
29 May – 1 June 2001
San Francisco, California

EVO for the Web

Agenda

- **Challenges of Web Development**
- **Software Development Lifecycles**
- **Strengths/Weaknesses of EVO**
- **EVO Case Studies**
- **Setting the Stage for EVO**
- **Monitoring the Lifecycle Progress**
- **Conclusion**

Challenges of Web Development

- **Product**
  - **Changing value proposition**
  - **Vague customer requirements**
  - **Adequate quality**
  - **Scalability**
- **Process**
  - **Unproven, immature, and evolving technologies**
  - **Change management especially towards release**
- **People**
  - **Constant learning of evolving technologies**
  - **No in-depth understanding of a technology due to changes**

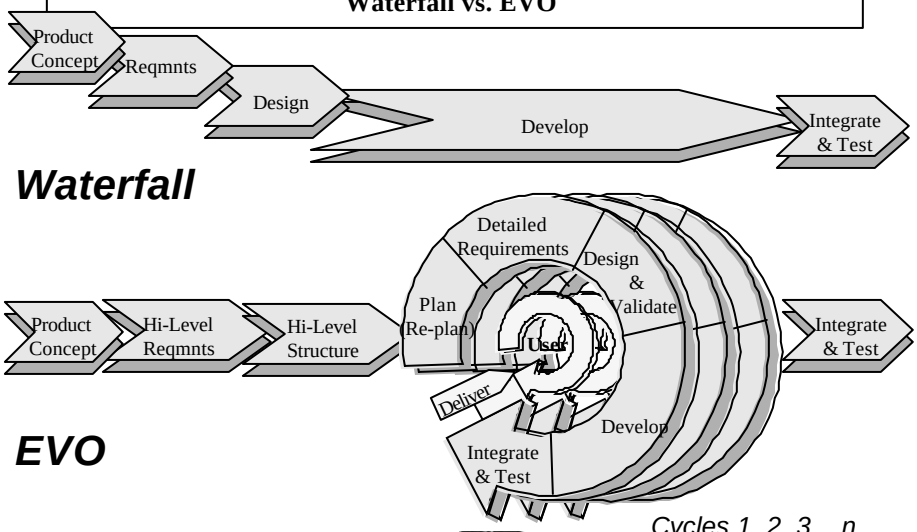
Overcoming Challenges

Utilize a software lifecycle that provides:

- **On-going visibility to scope, schedule, resources, and quality**
- **Constant focus on customer value**
  - **Early and effective feedback from customers**
  - **Incorporating customer feedback**
- **Immediate course correction when necessary**
- **Adequate management of tools and technology**
- **Effective collaboration between partners**

EVO for the Web

Waterfall vs. EVO

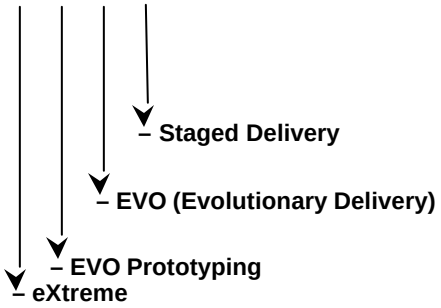


Bhushan Gupta and Steve Rhodes

EVO for the Web

"Flavors" of Iterative / Evolutionary Lifecycles

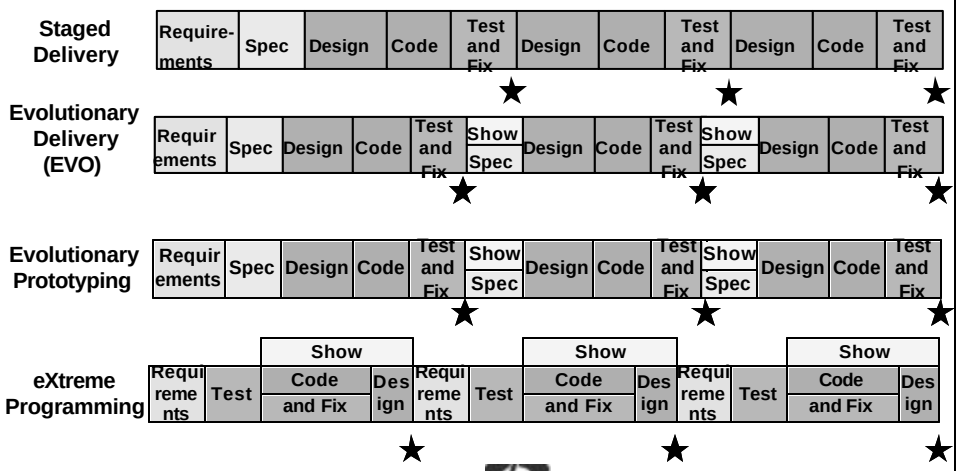
1. **Manage and reduce project risk,** plus …
2. **Manage and reduce program integration risk,** plus …
3. **Better fit user needs and market requirements,** plus …
4. **Respond to emerging and changing markets.**



Bhushan Gupta and Steve Rhodes

EVO for the Web

Iterative/Evolutionary Lifecycles



EVO for the Web

Criteria for Choosing the Appropriate Lifecycle

- **How well do my customer(s) and I understand requirements at the beginning of the project? Are significant changes likely as we progress?**
- **How much quality and reliability do I need?**
- **How well do I understand the system architecture? Am I likely to make major architectural changes midway through the project?**
- **Am I constrained by a predefined schedule?**
- **Do I need to provide my customers or management with visible progress throughout the project?**
- **How significant will the integration effort likely be?**

"Iterative" and "Evolutionary" Lifecycles meet these criteria.

EVO for the Web

Strengths of the EVO Lifecycle

- **Product**
  - **Better fit to user needs and market requirements.**
  - **Ability to respond to market changes during development**
  - **Consistent high-quality user feedback (managed vs. ad hoc)**
  - **Increased opportunity to hit market windows.**
  - **Accelerated sales cycle with early customer exposure.**
- **Processes/People**
  - **Increased product team productivity and motivation.**
  - **Better partitioning of work**
  - **Increased management visibility of project progress.**

EVO REDUCES YOUR RISK

EVO for the Web

Weaknesses of EVO

- **EVO appears to be much harder:**
  - **New paradigm for thinking about project structure**
  - **Perception of <u>more work</u>!**
- **Customers are changing:**
  - **Implicit EVO of "next bench" no longer applicable, as SW developers are no longer primary users of their products.**
  - **Customers more demanding**

    . . . and as a result:
  - **Greater customer commitment and involvement**
- **If your infrastructure has problems, EVO will make them even "louder":**
  - **Configuration management**
  - **Build process**
  - **Regression/release testing**

EVO for the Web

Overcoming Weaknesses of EVO

- **Schedule high risk activities first:**
  - **Resolve issues early and eliminate unknowns**
  - **Demonstrate feasibility up front**
- **Improve planning and scheduling skills through earlier insight; accelerate team learning.**
- **Avoid large-scale integration late in the project by small, frequent integration throughout.**
- **Make SW/FW development effort visible by delivering early, tangible results.**

EVO for the Web

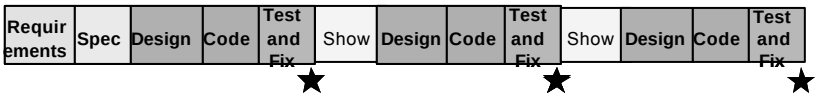
EVO Case Studies

Product Type	Staffing	Cycle Data	Results
SW	4 engineers, 1 PM	39 cycles @ 1-2 weeks	3x productivity
SW/HW	25 engineers, 3 PMs	12 cycles @ 2 weeks	early revenue
SW	8 engineers, 1 PM	6 cycles @ 3-4 weeks	best in class awards
SW	~12 engineers, 2 PMs	5 cycles @ 8 weeks	early market visibility, partner division feedback

EVO for the Web

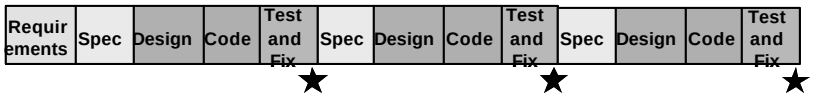
Example: The MaxiPull Project

***BackHand* Platform Provider/Partner**



. . . Evolutionary Prototyping

***DEF's MaxiPull* Development Teams**



. . . Hybrid: Evolutionary Prototyping -and- Waterfall

EVO for the Web

Example: The MaxiPull Project

- **DEF did not know how to acquire immediate, early feedback as soon as product was workable.**
- **DEF did not know to plan for making *major* changes based on that feedback.**
- **DEF found it difficult to set customer expectations to include experimentation.**
- **DEF and BackHand had different expectations for a demo-quality prototype.**
- **BackHand failed to understand DEF's requirements - reliability, availability, future enhancements**
- **Result: DEF never really did Launch and Learn**

Example EVO Cycle: XYZ Project

	Development Team	*Partners*
Monday	- Evaluate feedback from version N-1 - Decide activities for N - Design version N	Functionality Testing N-1
Tuesday	- Decide refactoring needs Refactor/develop code	Functionality Testing N-1
Wednesday	- Refactor/Develop code	Usability Testing
Thursday	- Complete code, build and release	Functionality Testing N-1
Friday	- Plan, Refactor, Develop Code N+1	Functionality Testing N-1

Getting Ready for EVO

- **Customer value proposition**
  - **Project Mission, Vision,**
  - **Use Cases**
  - **Major Risks**
- **Identifying genuine customer(s)**
  - **Business goals support value proposition**
  - **Early adopters of technological breakthroughs**
  - **Proponent of product workflow changes**
  - **Willingness to use imperfect product**
  - **Enthusiastic about providing frequent and quality feedback**
  - **Comfortable with limited product support**

Getting Ready for EVO

Establishing Software Processes

- **Customer feedback management**
- **Change management**
  - **Crisis management**
  - **Mediated change**
- **Development tools and environment selection**
  - **Tools/technology upgrade process**
  - **Identification of common tools between development teams**
- **Management of common components**
  - **Build and release of common code**
  - **Defect dependency management**

Identifying EVO schedule

- **Cycle time and number of EVO cycles**
- **Major milestones and EVO cycle alignment**
- **Cycle tasks and dependencies**
- **Scheduling cycle activities**

XYZ EVO Cycle #2 Theme: Infrastructure
Start Date: 10/30/00 End Date: 11/07/00

Group	Activity	Deliverables	Risks	Dependencies
Dev.	Create Site Map Define Interfaces Refine EVO Plan Validate Arch.	Site Map	None	Use Cases
OPS	Create Development Environment	Development plan	None	None
HFE	Customer Selection	Selection Criteria	None	Use Cases

OPS – Operations HFE – Human Factors Engineering

EVO for the Web

Evaluating EVO – Is it working for me?

Monitoring Progress

- **Ready to start next cycle on time**
- **No waiting between cycle N and N+1**
- **Planned progress towards value proposition**
- **Each cycle adds value to the customer**
- **Minimal turmoil (unplanned activities)**
- **The EVO rhythm is felt**

Identifying and Implementing Continuous Process Improvement

- **Questionable Customer Behavioral Patterns**
- **Does not understand the application**
- **Requires more than anticipated support**
- **Does not see the value in the product any more**

EVO for the Web

Conclusion

- **Challenges of Web development**
- **Software Development Lifecycles**
- **Strengths/Weaknesses of EVO**
- **EVO Case Studies**
- **Setting the Stage for EVO**
- **Monitoring the Lifecycle Progress**
- **Conclusion**

Adopting a Lifecycle for Developing Web Based Applications

Bhushan B. Gupta and Steve Rhodes
Hewlett-Packard Company

Abstract

Our products and services are at increasing risk in the Internet age. We have new markets with inexperienced providers and customers, our customers are moving targets, the competition forces us to pull up release schedules by weeks or months, and customer tolerance for defects has dropped precipitously. We need new processes that are "light," nimble, and flexible to the competing pressures of resources, time, and delivered features.

Of the various iterative lifecycles, Evolutionary[1] lifecycle (EVO) is best suited to managing all the risks cited above – simultaneously. It does so through a discipline of continuous process improvement, at short regular cycles. Plan, do, check, act . . . repeat! Process maturity continues to advance cycle by cycle; and we mature those processes that net the biggest return first. EVO adjusts the *clockspeed* of our product lifecycle to the *clockspeed* of the marketplace, allowing us to avoid the Darwinian fate of organizations that can't keep up. Development teams across Hewlett-Packard and Agilent have switched to an Evolutionary lifecycle (EVO) for its strengths in user feedback, risk analysis and mitigation, and product releasability to meet “first to market" challenges. Results range from "schedule visibility" to "3X productivity" and "best in class." Although, EVO requires more attention to process than other lifecycles, it provides an early focus on every aspect of product development and delivery.

Challenges of Web Development

The three main aspects of software development -- namely, **product**, **process** and **people** -- have all been challenged in the Internet era.

More often than not we guess what **product** our customers want, and we try to deliver it quickly to become first to market. Even if we clearly understand what our customers want at the on set of a project, the customer expectations change constantly and our understanding becomes obsolete. Web-based software products get instant market exposure and inadequate product quality results into loosing customer loyalty. And if we have a killer application on the web, we are instantaneously faced with the problem of scalability.

Accelerated time-to-market stresses our seasoned product generation **processes**. The development environments are new and supported by unproven tools[2]. In the open source environment, most tools are in their Beta release with no or minimal support. New technologies emerge everyday making most recent technologies obsolete. This results in unmanageable turmoil for the development environment, especially if the product is componentized and developed by a number of teams.

The third and most important variable, **people**, are the victims of the new emerging technologies. The software developers have to learn new technology on a regular basis. With a constant change in the technology, developers rarely master existing technology and move to an unknown technology.

It is normally possible to adjust the people and process variables to get desired product quality. But in a scenario where nothing is well defined, we must adjust all variables to meet our customers’ expectations. This makes a compelling argument for changing our development processes to match the product and people. Specifically, we should utilize a software lifecycle that provides:

- On-going visibility to scope, schedule, resources, and quality
- Constant focus on customer value

- Early and effective feedback from customers
- Incorporating customer feedback
- Immediate course correction when necessary
- Adequate management of tools and technology
- Effective collaboration between partners

Software Development Lifecycles

Anyone in the software business for very long realizes we are awash with process models. One of our beleaguered HP Project Managers calls it "the death of a thousand lifecycles." They are depicted in books and papers … that often have other books and papers piled on top of them! Steve McConnell[3] in his book Rapid Development does a nice taxonomy of lifecycles, including Pure Waterfall, Modified Waterfall, Code-and-Fix, Spiral, Staged Delivery, Design-to-Schedule, Design-to-Tools, Commercial Off-the-Shelf Software, and . . . Evolutionary Delivery.

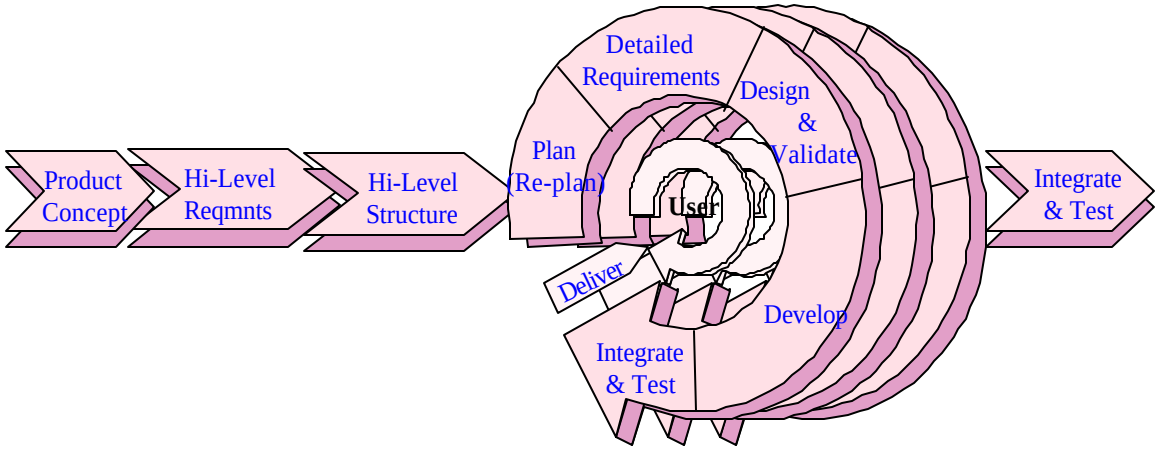


Fig.1 Evolutionary (EVO) Lifecycle

The most striking way to describe Evolutionary Delivery is to compare it with the Pure Waterfall. To be fair, even the Waterfall has cycles. A lifecycle, by definition, is an explicit repeatable process. Where other lifecycles distinguish themselves from the Waterfall is in what author Charles Fine calls "clockspeed."[4] In the halcyon days of "build it and they will come," product lifecycles could run from months to years in length. You might not be sure if you really were cycling through the Waterfall again, because you weren't with the company during the last cycle. Those who were had moved on or retired.

Stating the obvious, EVO is a software development method that replaces traditional "waterfall" development with small, incremental product releases or builds, frequent delivery of the product to users for feedback, and dynamic planning that can be modified in response to this feedback. As originally presented by Tom Gilb[5], the method had the following key attributes:

- Multi-objective driven
- Early, frequent iteration
- Complete analysis, design/refactoring, build and test in each step
- User orientation
- System approach, not merely an algorithm
- Open-ended basic systems architecture
- Result orientation, not SW development process centric

Using EVO, a product development team divides the project into small chunks. Ideally, each chunk is less than 5% of the overall effort. The chunks are then ordered so the most useful and easiest features are implemented first and so that some useful subset of the overall product can be delivered every one to four weeks. Within each EVO cycle, the software is designed, coded, tested and then delivered to users. The users give feedback on the product and the team responds, often by changing the product, plans, or process. These cycles continue until the product is shipped.

EVO is thus characterized by early and frequent iteration, starting with an initial implementation, followed by frequent cycles that are short in time and small in content. Drawing on ongoing user feedback, plan, design, code and test are completed for each cycle, and each release or build meets a minimum quality standard. This cycle offers opportunities to optimize results by modifying plan, product or process each cycle. The basic product concept or value proposition, however, does not change. (In Hewlett-Packard, we relax some of Gilb's instructions regarding EVO. In particular, it isn't absolutely necessary to deliver the product to real customers with customer-ready documentation, training, support, etc., to benefit from EVO.)

This begs the question: "When is EVO – Tom Gilb's EVO – appropriate?" As we will describe in a moment, switching from a traditional lifecycle, like the Waterfall, to an iterative lifecycle, like EVO, is **not free**. It requires a redistribution of effort, focus, and resources. It requires cultural change for developers and managers. And change is hard. Perhaps the most compelling reasons to shift from Waterfall to Iterative are:

- Manage and reduce project risk.
- Manage and reduce program integration risk.
- Better fit user needs and market requirements.
- Respond to emerging and changing markets.

There are several iterative lifecycles to choose from: Staged Delivery, Tom Gilb's EVO, Evolutionary Prototyping, and of more recent invention, the eXtreme programming[6] model. The additional effort and discipline required for each addresses one or more of the aforementioned benefits.

- Staged Delivery meets the objectives of managing project risk and managing program integration risk through early indicators of progress vs. plan, frequent iteration cycles to reduce the "big bang" effect, and flexibility to meet fixed ship dates by postponing cycles till future releases.
- EVO is optimized to insure the end product matches user needs and market requirements, by making course corrections as developers and users better understand each other. The EVO lifecycle is user-centric, with continual user feedback throughout, managed high-quality user elicitation, and incremental tuning of the product.
- EVO prototyping (sometimes called "Adaptive Delivery"), is similar to Gilb's EVO, but spends less time up front in requirements and architecture, and more time with multiple prototypes to test and evolve product concept. EVO prototyping responds well to emerging and changing markets, whereas Gilb's EVO is designed to enhance and track understanding of customers in a more stable marketplace.
- eXtreme programming, explained by Kent Beck in his 1999 book, was conceived and developed to address the specific needs of software development conducted by small teams in the face of vague and changing requirements. While described as a "lightweight methodology," it prescribes three of the tenets of EVO: constant integration, constant customer feedback, and frequent re-planning.

Microsoft's Synch-and-Stabilize

What of the most successful software company in the world? Do they use an iterative lifecycle? In their book, Microsoft Secrets[7], Michael A. Cusumano and Richard W. Selby, label Microsoft's project development style "synch-and-stabilize." Microsoft TechNet refers to their process model as "MSF," for Microsoft Solutions Framework.[8] With it, Microsoft has truly been able to make large teams work like small teams.[9] The overall strategy is to quickly introduce products that are "good enough" to gain market position, then to enhance and expand them into product families with multiple versions and upgrades. To get that speed and that scale, teams follow a strategy of "do everything in parallel with frequent synchronization." Cusumano and Selby summarize the key principles of that strategy in their 1997 article, "How Microsoft Build Software"[10]:

- Work in parallel teams but "synch up" and debug daily.
- Always have a product you can ship, with versions for every major platform and market.
- Speak a "common language" on a single development site.
- Continuously test the product as you build it.
- Use metric data to determine milestone completion and product release.

Any of the iterative lifecycles we've described matches those principles. But what Microsoft does is a compromise between Waterfall and Iterative: each product release is chunked into 3 or 4 milestone subprojects, each with about a 3rd or 4th of the features or components of that release. During the development phase, each subproject goes through a full cycle – design, code, test, stabilize – for its cluster of components, before moving on to the next milestone. All code for a subproject is synched with a daily build. At each major milestone, code from parallel subprojects is synched and stabilized.

Alan MacCormack, Harvard Business School, writes about Microsoft's development of its IE3 browser in "How Internet Companies Build Software[11]." He notes that IE3 development was iterative, but that product design remained flexible, in response to user feedback, and that the EVO model "mirrors the way IE3 was built." But Synch-and-Stabilize does differ from evolutionary lifecycles in several ways: Its cycles are long -- approximately 3 months -- in contrast to EVO's 2-to-3 weeks. Also, Microsoft projects include a portion of the schedule as buffer time, from 20% in application projects to 50% in systems or totally new projects. Gilb's EVO uses a fixed cycle time, with no optional slack before beginning the next cycle. (Some EVO teams do plan a periodic "catch-up" or "clean-up" cycle, say every 4th cycle.) Whether we equate Synch-and-Stabilize to EVO or not, Microsoft achieved many of the advertised benefits.

Choosing the Appropriate Lifecycle

Steve McConnell, in his book Rapid Development[12], differentiates lifecycles. Each has at least one characteristic that makes it a better choice, under certain project conditions. If you are shopping for a lifecycle, key questions you should ask include:

- How much do you understand the customer's requirements? In many cases, our customers can't really articulate what they want. Or, their needs change mid-way through the project. Or, we misunderstood what they wanted, or we chose the wrong customer to target, or … using a lifecycle that incorporates customer feedback can be really helpful in these situations.
- How much quality does your product need? Most products are not used in life-threatening situations, so a "reasonable" quality level is appropriate. But, if you're working on the flight control software for the space shuttle or a defibrillation machine or something else that could actually kill someone with a software defect, you need an extremely high-reliability lifecycle.
- How much architectural flexibility do you need/want? Your choice of lifecycle may make it easier/harder to change the architecture, but we think that programming paradigm has a much bigger impact.

- Can you successfully use this lifecycle? Evolutionary development on a two-week cycle is extremely stressful if you don't have robust project management in place. There are some lifecycles where you can be marginally successful without, for instance, rigorous scheduling techniques. There are others where you will be a visible, miserable failure unless you have good scheduling practices. Make sure you choose a lifecycle you can use.

Strengths of the EVO Lifecycle

One way to evaluate whether EVO is appropriate for web development, is to compare its advertised benefits against the web development challenges we described earlier:

- Product – changing value proposition

  EVO: Better fit to user needs and market requirements.
  EVO: Ability to respond to market changes during development.
  EVO: Consistent high-quality user feedback (managed vs. ad hoc)
  EVO: Increased opportunity to hit market windows.
  EVO: Accelerated sales cycle with early customer exposure.

- Processes/People – broad responsibilities

  Short frequent EVO cycles have distinct advantages for internal process and people considerations. First, continuous process improvement becomes a more realistic possibility with one to four week cycles. Second, the opportunity to show their work to customers and hear customer response tends to increase the motivation of software developers and consequently encourage a more customer-focused orientation. In traditional software projects the customer response payoff may only come every few years and may be so filtered by marketing and management, it is meaningless. Finally, the cooperation and flexibility of each developer required by EVO results in greater teamwork. Since scheduling and dependency analysis are more rigorous, there is less time, less "dead time," spent waiting on another person to complete his/her work.

  EVO: Increased product team productivity and motivation.
  -- Focus maintained on product vision / value proposition.
  -- Engineers see the results of their hard work much earlier.
  -- Less turmoil and fire fighting just before MR
  -- Stimulates engineering creativity
  EVO: Better partitioning of work
  -- Developers focus on providing a solution
  -- Users better able to describe needs
  EVO: Increased management visibility of project progress.

Each of the challenges we just addressed introduces the element of risk into a software project. From a business perspective, the biggest benefit of EVO is significant reduction in risk. This risk might be associated with any of the many ways a software project can go awry, including schedule, unusable products, wrong feature sets, or quality. By breaking the project into smaller, more manageable pieces and by increasing the visibility of the management team into the project, these risks can be addressed and managed.

Some examples of HP projects that used EVO to mitigate their risk:

- The product team had been unable to make truly easy-to-use software in two prior attempts. By using EVO, they were able to get very rapid feedback from the users on the usability of the software that was under development.
- The division had a new idea for testing digital integrated circuits by treating them as analog devices (making them much simpler to test from a programming standpoint). However, the

target audience for the product, manufacturing managers, are fairly skeptical of new technology. By using EVO, the product team was able to demonstrate the value of the new testing technique in actual production lines.

- A platform team knew they would need to demonstrate actual progress to upper management and to get early enthusiastic support of at least one product team using their platform. EVO gave them both of these. In addition, they were able to assess system performance (a critical attribute of the new platform) very early on.

In some respects, you can think of EVO as a “miner’s canary” for a project.

Overcoming Weaknesses of the EVO Lifecycle

While the benefits can be substantial, implementation of evolutionary development holds significant challenges. It requires a fundamental shift in the way one thinks about managing projects and definitely requires more management effort than traditional software development methods.

The challenges in using EVO successfully are mostly, but not exclusively, human resource issues. These include the shift in thinking about a new project structure paradigm and perceptions that EVO requires:

- More planning
- More tasks to track
- More decisions to make
- More cross-functional buy-in and coordination
- More difficulty coordinating software/firmware development with hardware
- Customers are changing

    - Implicit EVO of "next bench" no longer applicable, as SW developers are no longer primary users of their products.
    - Customers more demanding … and as a result:
    - Greater customer commitment and involvement

    Another “weakness” of EVO is a direct result of its most important strength: customer intimacy in the development process. The weakness: Customers are changing. They are no longer the poor cousins who trust that we know what’s best for them. The "next bench" syndrome is no longer valid as, more often, we developers are no longer the primary users of our own products. Not only that, but once we give users an ear, they become even more vociferous. They demand more input, more involvement in product definition and design phases which have traditionally been the undisturbed sanctum of the engineer. And as a result, our project involves greater customer involvement. The flip side for the customer is that we give them a voice, but expect them to work with us. It requires greater customer commitment to the development process. Where once they needed only muster the energy to grouse about what was released, now they are expected to repeatedly install, use and critique what comes out every few days.

- If your infrastructure has problems, EVO will make them even “louder.”
  Another weakness of EVO, compared to lifecycles with longer or more flexible release cycles, is its huge demands upon the build and test infrastructure. If your infrastructure has problems, EVO will put a search light on them. Practices and tools that appear to work fine with a Waterfall become capacity limited with EVO, practices and tools like:

    - Configuration Management
    - Build Process

- Regression Testing
- Release Testing

These perceived weaknesses of EVO are often valid; but they allude to very advantageous cost-benefit tradeoffs. For example, since many software developers are no longer primary users of their products, they must understand the primary users' needs, skill level and motivation. Forging an atmosphere of customer intimacy with developers will ultimately yield a product customers really want.

Some helpful ways to mitigate the weaknesses of the EVO lifecycle:

- Schedule high risk activities first:
  - Resolve issues early and eliminate unknowns
  - Demonstrate feasibility up front
- Improve planning and scheduling skills through earlier insight; accelerate team learning.
- Avoid large-scale integration late in the project by small, frequent integration throughout.
- Make SW/FW development effort visible by delivering early, tangible results.

EVO Case Studies

Despite the objections and the transition pains, EVO works. It is working at Hewlett Packard Company. Beginning in '96, HP's process architecture group championed EVO inside HP; it has launched EVO lifecycles with dozens of teams inside the company. Table 1 shows 5 sample projects from among 28 we have profiled. They succeeded because those teams stuck to the discipline of EVO, and because they reaped the benefits of EVO, early:

- Increased developer productivity
- Early release and revenue
- Awards for best-in-class by the user community
- Partner feedback
- Catching issues early

Product Type	Staffing	Cycle Data	Results
Software	4 Engineers, 1 Project Manager	39 Cycles @ 1-2 weeks	3x Productivity
Software/ Hardware	25 Engineers, 3 Project Managers	12 Cycles @ 2 Weeks	Early Revenue
Software	8 Engineers, 1 Project Manager	6 Cycles @ 3-4 weeks	Best in Class Award
Software	12 Engineers, 2 Project Managers	5 Cycles @ 8 Weeks	Early Market Visibility Partner Feedback

Table 1. HP Projects that Used EVO

Following are details from three Hewlett-Packard divisions from 3 entirely different businesses. While all division and product names used here are fictitious, the case descriptions are real.

Jaguar Project

The first project to use EVO at ABC Division was Jaguar. It consumed four software developers for a year and a half, eventually shipping over 120,000 lines of C and C++ code. Over 30 versions were produced during the eleven-month implementation phase in one- and two-week delivery cycles. Jaguar's reasons for using EVO were: (1) reduce the number of late changes to the user interface, and (2) reduce the amount of defects found during system testing.

Jaguar adapted Gilb's EVO methods, with one exception: user surrogates. ABC Division produces testers that are used in manufacturing environments. If the tester goes down, the manufacturer cannot ship their product. Beta sites, even when customers agree to them, are carefully isolated from any "real work," so Beta software is rarely, if ever, exercised. Fortunately, the Jaguar project had access to a group of "user surrogates": application engineers in marketing and test engineers in ABC Division's manufacturing department. The use of surrogates did not appear to have any negative impact.

Jaguar abandoned EVO about 2/3 of the way through the project as the final deadline loomed and additional work was acquired. The rigorous testing and defect repair of earlier EVO cycles was discontinued. The cost of this decision was quality! Developers began adding code at a rate double that of previous months. With all efforts focused on finishing, over half of the critical and serious defects were introduced in this last 1/3 of the project. Abandon EVO at your peril.

We discovered that EVO is rarely all-or-nothing, success-or-failure. Even though EVO was not used to complete Jaguar, the product was successful. The team attributed key results to EVO: First, it contributed to better teamwork with the users and more time to think of alternative solutions. Second, ABC Division still had significantly better critical/serious defect levels during system testing than with other non-EVO projects. Third, the team was surprised to see a marked increase in productivity (measured in KNCSS per engineer-month). The project manager attributes this higher productivity primarily to increased focus.

DEF Division

The DEF Division creates publishing e-services. Before launching its new MaxiPull project, DEF's management attempted to find an industry-tested product lifecycle by profiling major software business segments and picking a match. The profile covered the variance among users, accessibility of users, the "sizzle" they expected from products, staff turnover, and expectations for reliability. DEF found that, as an Internet service developer, they themselves cut across those business profiles:

- E-commerce development is technologically most similar to financial application development:
  - Requires development of large new systems – lots of documentation and design – and minimized risk
  - Fast development of add-on releases – lots of prototyping, and constant interaction with the customer
- E-commerce customer expectations are most similar to those of mass-market customers
  - Sacrifice reliability for time-to-market
  - Add reliability back into release N+1
  - Heterogeneous customer base (the 80/20 rule)
  - Maximize sizzle
- Techniques from financial apps often do not work well in the mass-market world, and vice versa.

Given this conundrum, DEF deferred to its e-Business platform provider/partner. BackHand designed an e-Business construction process of 12-18 months, involving the following steps:

1. Write down an idea
2. Build a "paper-thin" prototype
3. Use the paper-thin prototype to make the idea concrete in peoples' minds
4. Refine the idea and get funding for an industrial-strength prototype
5. Build an industrial-strength prototype: v1.0
6. Launch the prototype service and refine the business idea based on experience with real users, making constant changes to the idea/code base (resulting in iterative improvement of the idea and iterative destruction of the code base)

7. Step back from the prototype and figure out what abstractions would make the code base smaller, cleaner, and more maintainable
8. Build v2.0
9. Re-launch the service with expanded marketing budget

DEF identified the following lifecycle requirements for its portion of the project:

- Must support early feedback
- Encourages re-planning to optimize business opportunity
- Supports rapid development practices:
  - Minimizes chaos by making some activities very easy and reliable
  - Maximizes staff flexibility with regularized processes

DEF chose Evolutionary Prototyping from among the iterative lifecycle alternatives. The MaxiPull project started at Step 1 around January 1999. The HP team built a paper-thin prototype by April 1999. BackHand started work on the industrial-strength prototype in June. And the prototype service was launched in August. A tremendous number of business idea changes were pushed into the code base from that August debut through December 1999.

Conclusions, after the 1st year of MaxiPull:

The Evolutionary Prototyping lifecycle is appropriate for this business. But the devil is in the details:

- DEF did not know how to acquire immediate, early feedback as soon as the product was workable. DEF did not have the skills to do adequate requirement analysis. (This is a skill taught in the financial world, where the customer requirements are numerous and often conflicting and not well thought out, but not taught much in some of the other software fields.)
- DEF did not know to plan for making major changes based on that feedback.
- DEF found it difficult to set customer expectations to include experimentation.
- DEF and BackHand had different expectations for a demo-quality prototype.
- BackHand failed to understand HP's requirements - reliability, availability, future enhancements
- Result: DEF never really managed to Launch and Learn. DEF did launch x.0, launch x.1, launch x.2, learn from x.0, launch x.3; learnings were delayed till launch+2. (DEF did learn, but it was unnecessarily painful and continues to be so.)

XYZ

XYZ is a work in progress to develop a Web-based application. It started with a firm decision by 8 engineers and a project manager to use EVO. The first major challenge for XYZ was to build the infrastructure. Once the infrastructure was in place, the XYZ team scheduled one-week EVO cycles. Very early in the product development, the team was capable of weekly and nightly builds. XYZ has actively pursued user feedback. The early cycles included the customer evaluation of loosely defined product features. A set of working features has undergone the usability testing and the product is further evolving. Frequent refactoring -- cleaning up and restructuring code internals **after** external behavior has been implemented --has become a team ritual.

XYZ has now been joined by two other teams, PQR and ABC, building applications using XYZ code base. The PQR team also uses EVO and has synchronized its activities with the XYZ team. The two teams, XYZ and PQR, have successfully set up joint build processes and common tools and build environment. Communication and coordination of joint activities has been a challenge but the two work products have become a powerful application. The ABC product has become an integrated part of XYZ.

Day	XYZ Team	Partners
Monday	Decide activities for N* Decide refactoring needs for N	Functionality Testing N-1
Tuesday	Refactor/develop code for N	Functionality Testing N-1
Wednesday	Refactor/develop code for N	Functionality Testing N-1 Usability Testing 1…N-1
Thursday	Build/Release N	Functionality Testing N-1 Usability Testing 1…N-1
Friday	Evaluate feedback from N-1 (next cycle)	Functionality Testing N-1 (just completed)

Table 2. Typical XYZ Weekly EVO Activities

* N represents the current cycle.

Setting the Stage for EVO

EVO mandates a close coordination between different facets of a project namely, development, quality assurance, human factors engineering, and marketing. It is essential that everyone involved with the project understands the EVO lifecycle especially, his or her role, and is committed to support the effort. Having real-customer feedback starting from the beginning of the project is a must, and marketing should be prepared to manage the customer relationship throughout the life span of a project.

At Hewlett-Packard we achieve this commitment by educating project personnel on EVO. Normally, project personnel go through a three-day workshop on EVO. The XYZ project team attended a condensed version of EVO presented by one of the authors (Steve Rhodes). This motivated the entire group to use EVO. More importantly, the group had an opportunity to decide upon important process parameters such as EVO cycle length, planning cycles, and change management process. Following are some of the project artifacts that should be in place by the end of EVO planning cycles.

Customer Value Proposition

The project must have a compelling customer value proposition. Since, product functionality is driven by the customer feedback, a development team may be constantly working with ever changing customer requirements. The project plan must include a well-defined mission, vision, and objectives.

Customer Selection Process

Customer feedback must we honest and genuine to be meaningful. A customer must be an early adopter of technology, supporter of workflow changes, willing to use an imperfect product, use constructive criticism, and be comfortable with limited product support.

Customer Feedback Management Process

To keep customers interested in the project we must treat their feedback with sensitivity. The process must cover feedback solicitation, evaluation, and incorporation of customer suggestions into product requirements. If customer suggestions are not incorporated into the project, the customer should get a meaningful explanation.

Change Management Process

In EVO, the requirements are not frozen as in the Waterfall lifecycle. The team will be faced with the major and minor changes in the product value proposition. It is therefore necessary to develop a well defined change management process to avoid delays in making critical decisions.

Build and Deployment Process
At the end of each EVO cycle, a working set of product functionality is normally delivered to the customer. Consequently, the build and deployment process should be well defined and efficient. In the Internet age where new build tools often emerge, the process should be flexible enough to take advantage of new tools with the least possible turmoil.

Minimal functionality Set
Development should begin with the identification of a minimal set of product functionality preferably, e.g. in the form of use cases. These use cases must be validated by a group of potential customers. In addition to clarifying customer requirements, validation also helps identify possible customers during the EVO development.

Monitoring the Lifecycle Progress

Each cycle has scheduled activities based upon the targeted product functionality to be delivered to the user. These cross-functional activities are specific to project management, refactoring, development, testing, and deployment. Any deviation in the schedule indicates problems in EVO planning.

Planned activities completed significantly early in the cycle
This indicates that the cycle length is longer than optimal and should be adjusted. Adding more functionality in the middle of a cycle would cause a high level of turmoil.

Planned activities not completed in the cycle
Indicates that the planned functionality is more than optimal. The functionality in each cycle must be adjusted so that it can be completed during the cycle length. It is not always necessary to deliver new functionality in each cycle.

Customer does not have meaningful feedback
Indicates that the functionality delivered to the user does not have adequate business impact. Evaluate or reschedule the planned functionality for each EVO. Make sure that the functionality is delivered in the order of importance to the user. Another possible reason may be that the customer does not feel that his/her feedback is incorporated in the product. Managing customer relationship is an important aspect of EVO. To summarize, if the team is waiting for the next cycle to start or rushing the current cycle to finish and the customer is not participating enthusiastically, EVO is not making adequate progress.

Identifying and implementing continuous process improvements

This section is related to the previous section on “Monitoring the Lifecycle Progress” and discusses some of the activities that facilitate continuous improvements in EVO. Customer feedback is the major differentiator between EVO and the other iterative lifecycles and can provide a strong basis for continuos process improvement. The following customer behaviors illustrate opportunities to improve the EVO process.

Customer did not fully understand the usage of the application
The customer may not have received adequate education on the application. He/she may not have realized how to incorporate the proposed application into their workflow. Re-educating the customer and evaluating their actual participation should be very effective.

Customer requires more than anticipated support
Indicates that application has either too many defects to be functional or is too complex to use. If the application is defective, management must focus on quality aspects including managing customer requirements. If the application is too complex to use, it is time to focus on usability.

Customer is unenthusiastic in providing feedback

The customer may not find the newly added functionality useful or may sense that his/her feedback is not being incorporated in the application. The customer management team must assure that decisions related to customer feedback are communicated to him/her in a timely manner. The team must also diligently evaluate the priority of functionality for immediate and future cycles.

Customer no longer sees the value in the application
This could be a serious issue. It may require switching to an alternate customer. Be prepared to re-validate the customer value proposition and overall business plans.

Conclusion

This paper has covered:
- Challenges of Web development
- Software Development Lifecycles
- Strengths/Weaknesses of EVO
- EVO Case Studies
- Setting the Stage for EVO
- Monitoring Progress

To quote Alan MacCormack[13], professor of technology at Harvard Business School:

> "The benefits of an evolutionary approach to software development have been evangelized in the software-engineering literature for many years. However, the precise form of an evolutionary model and the empirical validation of its supposed advantages have eluded researchers. The model has now been proved successful in the Internet-software industry."

References:

[1] Alan MacCormack, "Product-Development Practices That Work: How Internet Companies Build software," MIT Sloan Management Review, Winter 2001
[2] Jessica Burdman, Collaborative Web Development, Addison Wesley, ISBN 0-201-43331-1
[3] Steve McConnell, Rapid Development: Taming Wild Software Schedules, Microsoft Press, 1996, ISBN 1-556-15900-5
[4] Charles Fine, Clockspeed: Winning Industry Control in the Age of Temporary Advantage, Perseus Books Group, 1998, ISBN 0-738-20001-8
[5] Tom Gilb, Principles of Software Engineering Management, Addison-Wesley, 1988
[6] Kent Beck, Extreme Programming Explained: Embrace Change, Addison Wesley, 1999, ISBN 0-201-61641-6
[7] Michael A. Cusumano and Richard W. Selby, Microsoft Secrets – How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People, 1995, The Free Press, ISBN 0-02-874048-3
[8] *"MS Solutions Framework: Process Model for Application Development,"* Microsoft's TechNet website, http://www.microsoft.com/technet/Analpln/process.asp
[9] Michael A. Cusumano, "How Microsoft makes large teams work like small teams," *Sloan Management Review*; Cambridge; Fall 1997
[10] Michael A. Cusumano and Richard W. Selby, "How Microsoft Builds Software," *Communications of the ACM*, June 1997, Vol. 40, No. 6
[11] Alan MacCormack, "How Internet Companies Build Software," *MIT Sloan Management Review*, Winter 2001
[12] Steve McConnell, Rapid Development: Taming Wild Software Schedules, Microsoft Press, 1996, ISBN 1-556-15900-5
[13] Alan MacCormack, "How Internet Companies Build Software," *MIT Sloan Management Review*, Winter 2001

QW2001 Paper 7W2

Mr. Eric Patel
(Nokia Home Communications)

Rapid SQA: Web Testing At The Speed Of The Internet

Key Points

- The quality attributes that are important in web testing
- RapidSQA philosophy, goals, and program elements
- Defect management, metrics, and release criteria for web testing

Presentation Abstract

The demand for high quality websites in business, especially for e-commerce, has led to the reengineering of traditional testing practices in order to keep up with this e-business pace. Business pressures have mandated the QA organization to test more software, in more complex environments, in abbreviated (and often unrealistic) timeframes, while maintaining high quality. This iterative, rapid release schedule has called for an optimized, rapid testing strategy. Rapid Software Quality Assurance (RapidSQA) is a modified testing methodology containing best practices for optimizing Web testing.

About the Author

Eric Patel is QA & Test Manager at Nokia where he leads a team that tests digital communications solutions for the home. He has 10 years of experience in software testing, test management, and software quality assurance. In addition to QW2001, Eric has presented for the Software Quality Group of New England and will also be presenting at BOSCON, PSQT/PSTT East, SWaNH SWEQSIG, NESQAF, STAR East, and TCS this year.

He is a member of ASQ, IIST, and NESQAF. Eric is an ASQ Certified Quality Manager (CQM) and ASQ Certified Software Quality Engineer (CSQE), and will teach the CSQE Test Prep Class for the ASQ Boston Section. This year he will participate in the Software Test Managers Roundtable (STMR), a periodic gathering of senior practitioners in the QA industry. Eric is also a reviewer for Software Quality Professional and The Journal of Software Testing Professionals. He holds a Bachelor of Science degree in Electrical Engineering from the University of Vermont.

QualityWeek 2001

The Internet Wave

*Rapid*SQA

Web Testing at the Speed of the Internet

Eric Patel, CQM, CSQE
QA & Test Manager
Nokia Home Communications
eric.patel@nokia.com
http://www.nokia.com/home

NOKIA

Agenda

- Challenges of Web Testing
- Web of Quality
- *Rapid*SQA
  - Philosophy
  - Goals
  - Program
  - Test Sequence
- Defect Management
- Metrics
- Example Release Criteria
- Summary

NOKIA

Challenges of Web Testing

Challenge(s)	Effect(s)
• Muti-tier architecture • Numerous end user variables	• Multiple failure points • Difficult to pinpoint source of errors
• Time-to-market pressures • Date driven releases	• Shorter testing cycles • Frequent website updates
E-commerce functionality mandates high reliability	• Emphasis on security • 3rd party cooperation
No user documentation for website	Usability becomes a vital issue
Users with low (failure) risk tolerance	Higher quality expectations

NOKIA

Web of Quality

- Usability: ease of use, navigation, clarity
- Performance: efficiency of response times
- Credibility: reliability + scalability + accuracy
- Functionality: fitness for intended purpose
- Recoverability: failure switchover
- Security: protection from unintended use
- Compatibility: operating system and browser

NOKIA

*Rapid*SQA Philosophy

- Quality is defined by the customer, not the specification
- Paradigm shift from traditional lifecycles
- Proactive, effective, and efficient testing
- “Start early, test often, end never”
- Cross-functional, multi-department, team effort
- ‘Power of duplication’ testing strategy
- Continuing education and training

  NOKIA

*Rapid*SQA

- Goals
  - Deliver customer-defined quality in rapid, sequential releases
  - Maximize testing efficiency through concurrently running programs
  - Eliminate test and project blockers
  - Produce documentation as needed
  - Hold the gains
- Program
  - Testing foundation
  - Risk-based testing
  - “24/7 testing”
  - Critical Issue Resolution Team (CIRT)
  - Tools
  - Acceptance test program

  NOKIA

Recruitment Program

- Ongoing recruitment
  - It takes time to find good people
- Qualifications
  - Formal education
  - Hands-on SQA and/or Web testing experience
  - Excellent communication and interpersonal skills
- Additional beneficial technical skills
  - Programming background
  - Test automation experience
- Ongoing training and education
  - Classes, seminars
  - Certification
  - Self-study
  - Conferences, meetings

  NOKIA

Working Environment

- Work hours
  - Quantity vs. quality
  - Avoid burnout and turnover
- Management and HR issues
  - Job descriptions
  - Performance management system
  - Flex time, paid time off
  - Reward and recognition system
  - Employee retention
- Cross-functional team synergy
  - Integrated team of developers, designers, and QA
  - Project manager
  - Rapid and empowered decision making
  - Frequent written and face-to-face communications
  - Release Committee

  NOKIA

Testing Environment

- Three (3) separate environments
  - Development (staging)
  - QA (test)
  - Production (live)
- QA (test) environment equivalent to production (live) environment
  - Web server
  - Application server
  - Database server
  - Restricted access
- Test Lab
  - Test every supported configuration
  - Run automated test scripts
  - Accommodate usability (user) testing

NOKIA

Risk-Based Testing

- Customer risk tolerance
  - What is the level of risk that the customer will *knowingly* accept?
  - Early Adopters => high tolerance for risk
  - Early Majority => low tolerance for risk
- Prioritized testing approach
  - Identify business-critical functionality
  - Test the highest risk areas => lowest risk areas
  - Conduct a periodic risk assessment
- Configuration testing
  - Identify supported platforms and browsers
  - Create a testing matrix
  - Prioritize the various combinations
- Use risk factors to create a risk profile
- Rollback strategy for risk mitigation

NOKIA

Example Risk Profile

	Coding	Problem			Defect	Customer	
Tool	*Quality*	*LOC (est.)*	*History*	*Usage*	*Occurrence*	*Feedback*	*Released*
High Risk:							
Quiz	0	44,400	8	3	3	2	pre-1.0
Student/Grader Management	0	15,900	10	2	2	2	pre-1.0
Image DB	0	4,000	4	1	1	3	pre-1.0
Student Presentations	0	2,200	2	1	2	3	pre-1.0
Bulletins	1	24,800	8	4	1	2	pre-1.0
Homepage Design	1	17,800	3	2	1	2	pre-1.0
Path (including audio, video)	1	15,000	7	4	2	2	pre-1.0

NOKIA

Example Testing Matrix

			Platforms		
Clients & Browsers	**Windows NT Server 4.0**	**SPARC Solaris 2.7**	**Redhat Linux 6.1**	**HP/UX 10.20**	**IBM AIX 4.2**
Microsoft					
Windows 95 (OSR 2.5)					
Navigator 3.04					
Communicator 4.72					
IE 4.0					
IE 5					
Windows98 (Second Edition)					
Navigator 3.04					
Communicator 4.72					
IE 4.0					
IE 5					
Windows NT Wkst. (w/SP5)					
Navigator 3.04					
Communicator 4.72					
IE 4.0					
IE 5					
Macintosh					
Navigator 3.04					
Communicator 4.72					
IE 4.5					
IE 5.0					
UNIX/LINUX					
Navigator 3.04					
Communicator 4.72					

NOKIA

"24/7 Testing"

- Ongoing, multi-level test program
  - Quality Engineering (QE) vs. Quality Assurance (QA)
  - Review functional & design specifications
  - Train developers in unit and integration testing
- Test on QA (test) and production (live) servers
  - Test as thoroughly as possible on the QA (test) servers prior to release
  - Continue testing on production (live) servers after release
- Outsourcing
  - Employ the testing services of a third party
  - Allows for additional concurrent coverage

  NOKIA

Test Automation Program

- Performance and functional test tools
- Jump-start the automation effort
  - Hire employees with test automation experience
  - Send employees to training classes
  - Hire consultants or contractors to write the scripts
- Automate tests from software components that don't frequently change
  - Smoke
  - Functional
  - Performance/Load/Stress
  - Regression
- Use cases will help you mimic the user's experience

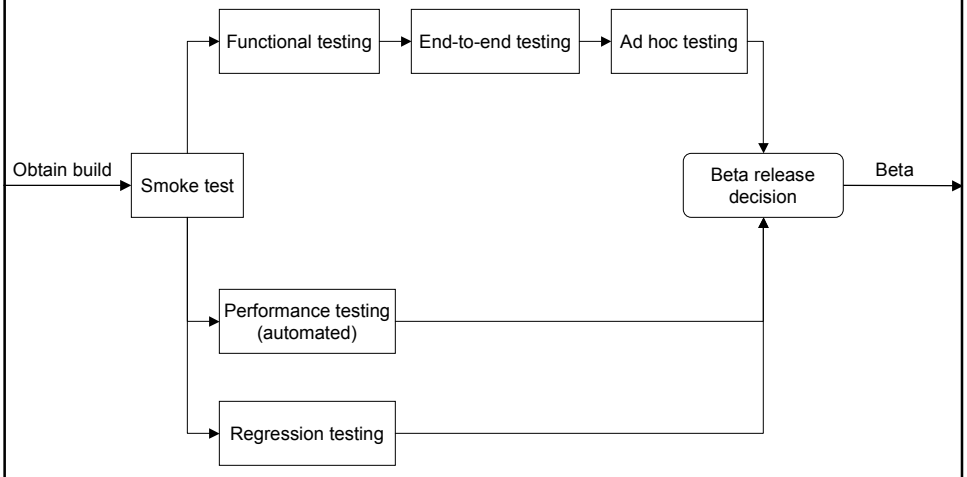
  NOKIA

Acceptance Test Program

- Acceptance Test Team (ATT)
  - Employees
  - Third-party participants (e.g., vendors, partners)
  - Customers and end users
- Power of duplication
  - Not just the SQA organization performing testing
  - Coordinated and concurrent effort
- Increased coverage for compatibility testing
- Usability (user) testing
  - Test prototypes and user interface (UI) mockups
- Beta testing
  - Transform beta testers into "release testers" and keep on testing

NOKIA

*Rapid*SQA Test Sequence



NOKIA

Defect Management

- Documented defect management process
  - Definitions of defect and enhancement priorities
  - Workflow
- Web-based defect tracking system
  - Remote access
  - 24/7 access
- Fast resolution of critical bugs
  - CIRT
- Bug review (triage) meetings
  - Cross-functional sit-down meetings
  - Daily stand-up meetings
- User feedback button on web page
  - Issue submittal form for defects or enhancements

17

NOKIA

Metrics

- Defect List

	Yesterday's	---------- T o d a y ----------			Today's
Priority	Total	Found	Checked In	Fixed	Total
Critical					
High					
Medium					
Low					
Total					

Today's Total = (Yesterday's Total) + (Found) – (Fixed)

- Quality Condition (QUALCON) Rating

Rating	***Critical Defects***	***Find/Fix Ratio***	***Tester Feedback***	***Customer Reported Critical Issues***	***DRE***
5	none	< 1	A	none	≥ 80%
4	≤ 3	≤ 1	B	1	≥ 80%
3	≤ 5	≤ 1	C	2	≥ 70%
2	≤ 10	≤ 2	D	≤ 3	≥ 60%
1	> 10	≤ 2	F	> 3	≤ 50%

18

NOKIA

Example Release Criteria

- All functional requirements have been satisfied
- All reported Critical Priority defects have been closed
- Reported High Priority defects have been deemed acceptable by the Release Committee or have been addressed as a workaround
- Results of performance testing are equal to or better than the current version
- High risk code has been reviewed
- All graphs show positive trends for X consecutive days (minimum) leading up to the release date
- Sign off has been granted by the Release Committee

NOKIA

Summary

- 'Rapid SQA' should not result in 'diminished quality'
- Web testing demands smart testing (effectiveness and efficiency)
- Communicate tradeoffs with cost, time and quality
- Balance ad hoc and disciplined testing practices
- Collaborate closely with developers and designers
- Objective release criteria and informed decision making
- Don't forget the interpersonal stuff
- Keep learning

NOKIA

QW2001 Paper 8W1

Dr. James Helm
(Univ. of Houston Clear Lake)

Web-Based Application Quality Assurance Testing

Key Points

- Website Application Testing
- Quality Assurance
- Web Testing

Presentation Abstract

Web-Based application quality assurance testing can be defined as a planned and systematic pattern of actions necessary to instill confidence that the website client and server products conform to an established set of measurements. Web-based testing is a repetitive process of identifying defects, where a defect is any variance between actual and expected results. A flaw in either the client or server application software can cause a defect. The website is essentially client/server applications - with web servers and browser clients. Web testing must be given to interactions between browser pages, TCP/IP communications, Internet connections, firewalls, applications that run in web pages (such as applets, Javascript, plug-in applications), and applications that run on the server side (such as CGI scripts, database interfaces, logging applications, dynamic page generators, asp, etc.). In addition there are a wide variety of servers and browsers, various versions of each, with differences between them, variations in connection speeds, rapidly changing technologies, and multiple standards and protocols.

Web site testing will become a major ongoing quality assurance function where web-testing tools will ensure a repetitive and repeatable testing process. This paper looked at six web-based testing tools. The tools were evaluated based on: timelines, structural quality, content accuracy and consistency, response time and latency, and performance. Timeline evaluate how often and rapidly a website has changed since the last upgrade. Structural quality measures how well all parts of the website link together. The links and images inside and outside the site must exist and be on line. Content of the critical pages must match what the uses require to be displayed. Key phrase must continue to exist in dynamically changeable pages and critical pages must maintain the same quality from version to version. Accuracy and consistency means that web pages downloaded from one time to the next are still accurate and consist with previous versions. Response time and

latency measures the website server response to a browser request within a tolerable performance parameter. It should also test pages of the site that are so slow that a user will discontinue using the site. Performance measures the cycle of browser to web, web to website, website back to web, web back to browser. It also measures the web load based on usage, number of users, and critical times. The six web-based testing tools used to evaluated these quality assurance criteria were: Rational Siteload, Doctor HTML, Dr. Watson, NetMechanic HTML ToolBox, Web Performance Trainer 2.0, and WebART .

About the Author

James C. Helm received the BS in Mathematics and Physics from Missouri Valley College, the MS in Mathematics from the University of Missouri at Rolla, and the Ph.D. degree in Industrial Engineering, Operations Research from Texas A&M University. He is presently the Chair of Systems Engineering and associate Professor of Software Engineering in the School of Natural and Applied Sciences at the University of Houston Clear Lake. He has had thirty years of industrial experience as: Senior Principal Engineer with the Boeing Co. at NASA/JSC Sunny Carter Test Facility supporting the International Space Station; a Senior Computer Scientist with SAIC; Principal investigator of Ada Research and Development (R&D) with Ford Aerospace and Communications; HAL/S Project and Contract Manager for HAL/S Compiler for the Space Shuttle at Intermetrics; and with IBM FSD supported NASA on GEMINI and APOLLO missions. He was an instructor in Mathematics at the University of Missouri Rolla, and has taught Industrial Engineering and Computer Science courses at Texas A&M, was an adjunct faculty at UHCL for twenty-five years. His areas of interest are Systems & Software Engineering, operations research, computer science, and simulation and modeling.

Web-Based Application Quality Assurance Testing

James C. Helm
Assistant Professor, Systems Engineering
School of Natural and Applied Sciences
University of Houston Clear Lake
helm@cl.uh.edu http://nas.cl.uh.edu/helm
Commercial Quality 8W1

Presentation Overview

- Introduction
- Testing Tools
- Rational SiteCheck
- Summary
- Reference

Introduction

- Web-Based application quality assurance testing can be defined as a:
  - planned and systematic pattern of actions necessary to:
    - instill confidence that the website client and server products conform to an established set of measurements.
- Web-Based testing is a:
  - repetitive process of identifying defects
    - a defect is any variance between actual and expected results

This paper looked at six web-based testing tools.

The tools were evaluated based on:

1. Timelines
2. Structural quality
3. Content
4. Accuracy and Consistency
5. Response Time and Latency
6. Performance

Quality Assurance Testing Criteria

1. Timeline evaluate how often and rapidly a website has changed since the last upgrade.
2. Structural quality measures how well all parts of the website link together. The links and images inside and outside the site must exist and be on line.
3. Content of the critical pages must match what the uses require to be displayed. Key phrase must continue to exist in dynamically changeable pages and critical pages must maintain the same quality from version to version.

Quality Assurance Testing Criteria (cont)

4. Accuracy and consistency means that web pages downloaded from one time to the next are still accurate and consist with previous versions.
5. Response time and latency measures the website server response to a browser request within a tolerable performance parameter. It should also test pages of the site that are so slow that a user will discontinue using the site.
6. Performance measures the cycle of browser to web, web to website, website back to web, web back to browser. It also measures the web load based on usage, number of users, and critical times.

The Six Web-Based Testing Tools Were:

1. Rational SiteCheck
2. Doctor HTML
3. Dr. Watson
4. NetMechanic HTML ToolBox
5. Web Performance Trainer 2.0
6. WebART

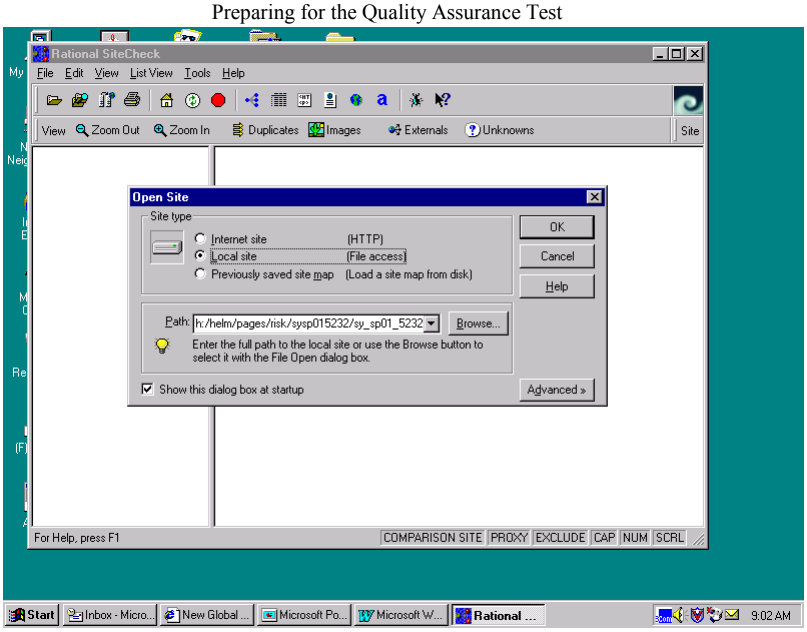
Rational Suite Enterprise

- Rational Test is an integrated product under Rational Suite Enterprise for the automated testing of enterprise-level client/server applications.
- Rational Test combines client/server testing, management tools, and a formal methodology for automated testing of cross-Windows client/server applications.
- Rational Test is comprised of the following component products:
  - Rational Robot (test recording tool),
  - Rational ClearQuest / TT Edition (defect tracking),
  - Rational SiteCheck (web site management),
  - Rational SiteLoad (web site load analysis),
  - TestManager WebEntry (web-based defect entry tool)
  - TestManager (test planning, management, and analysis tool).

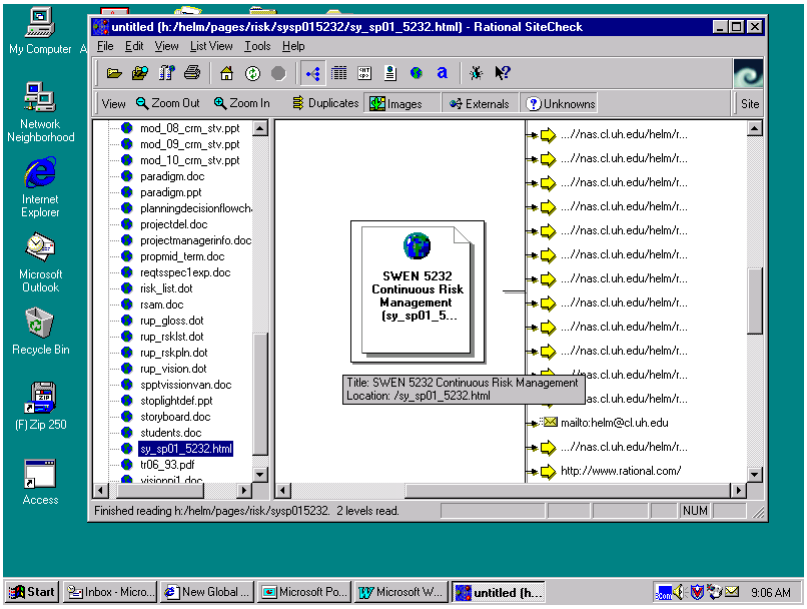
Rational SiteCheck

- Used to administer any Intranet or World Wide Web site.
- Perform Quality Assurance on:
  - folders and files
  - check links
  - monitor sites for changes and updates
  - examine the site for defects
  - and more

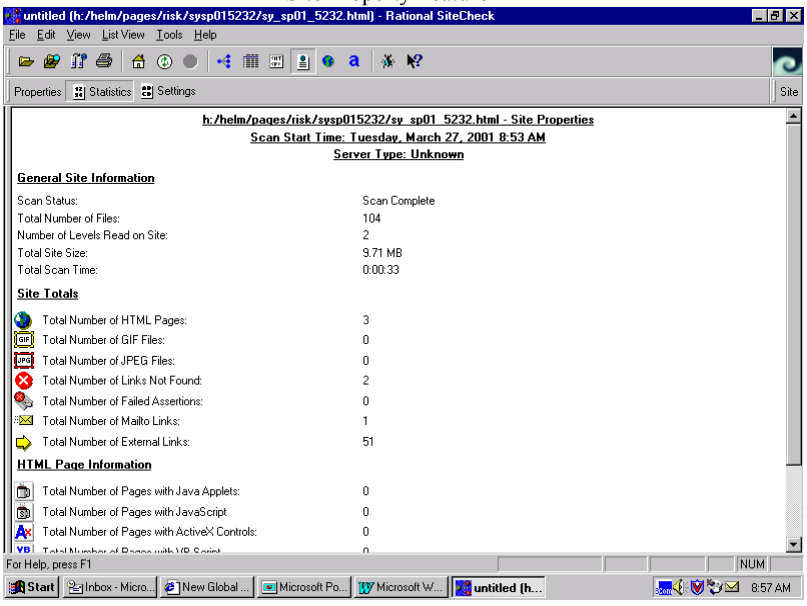
Preparing for the Quality Assurance Test



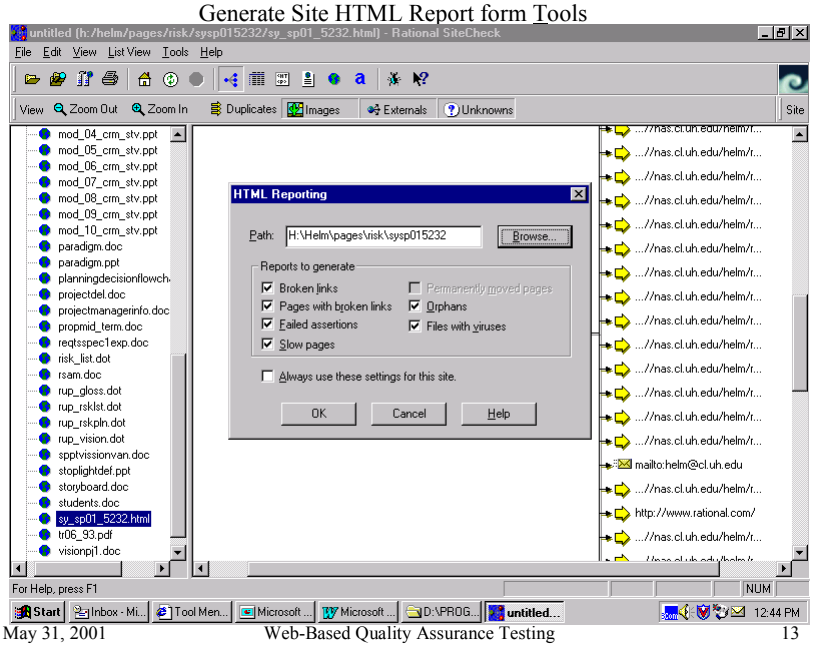
Page View selected form View



Site Property Feature



Generate Site HTML Report form Tools



Summary

- The goal is to prevent defects and Provide Quality Assurance
- Web-based testing is now a major ongoing quality assurance function where web-testing tools will:
  - Ensure a repetitive and repeatable testing process
  - Sustain Quality for Web Servers and Browser Clients

Reference

- H. Berlack 1992. *Software Configuration Management.* New York, NY: John Wiley & Sons, Inc.
- J. Buckley 1993. *Implementing Configuration Management, Hardware, Software and Firmware.* Los Alamitos, CA: IEEE Computer Science Press.
- G.G. Schulmeyer and J.I. McManus1992. *Handbook of Software Quality Assurance.* New York, NY: Van Nostrand Reinhold.
- David Whitgift 1991. *Methods and Tools for Software Configuration Management.* New York, NY: John Wiley & Sons, Inc.
- http://watson.addy.com
- http://www.cigital.com/marick/
- http://www.methods-tools.com/tools/frames_testing.html
- http://www.netmechanic.com
- http://www.rational.com
- http://www.softwareqatest.com/qatweb1.html

Web-Based Application Quality Assurance Testing

James C. Helm
Assistant Professor, Systems Engineering
School of Natural and Applied Sciences
2700 Bay Area Boulevard
Houston, Texas 77058-1098
VPN 281-283-3875 FAX 281-283-3828
helm@cl.uh.edu http://nas.cl.uh.edu/helm

Abstract

Web-Based application quality assurance testing can be defined as a planned and systematic pattern of actions necessary to instill confidence that the website client and server products conform to an established set of measurements. Web-Based testing is a repetitive process of identifying defects, where a defect is any variance between actual and expected results. A flaw in either the client or server application software can cause a defect. The website is essentially a client/server application - with web servers and browser clients. Web-Based testing will become a major ongoing quality assurance function where web-testing tools will ensure a repetitive and repeatable testing process. This paper looked at six web-based testing tools. The tools were evaluated based on: timelines, structural quality, content, accuracy and consistency, response time and latency, and performance. The six web-based testing tools used to evaluated these quality assurance criteria were: Rational SiteCheck, Doctor HTML, Dr. Watson, NetMechanic HTML ToolBox, Web Performance Trainer 2.0, and WebART.

Introduction

Web-Based application quality assurance testing can be defined as a planned and systematic pattern of actions necessary to instill confidence that the website client and server products conform to an established set of measurements. Web-Based testing is a repetitive process of identifying defects, where a defect is any variance between actual and expected results. A flaw in either the client or server application software can cause a defect. The website is essentially a client/server application - with web servers and browser clients. Web testing must be preformed for interactions between browser pages, TCP/IP communications, Internet connections, firewalls, applications that run in web pages (such as applets, Javascript, plug-in applications), and applications that run on the server side (such as CGI scripts, database interfaces, logging applications, dynamic page generators, asp, etc.). In addition there are a wide variety of servers and browsers, various versions of each, with differences between them, variations in connection speeds, rapidly changing technologies, and multiple standards and protocols.

Web-Based testing will become a major ongoing quality assurance function where web-testing tools will ensure a repetitive and repeatable testing process. This paper looked at six web-based testing tools. The tools were evaluated based on: timelines, structural quality, content, accuracy and consistency, response time and latency, and performance.

Timeline evaluate how often and rapidly a website has changed since the last upgrade. Structural quality measures how well all parts of the website link together. The links and images inside and outside the site must exist and be on line. Content of the critical pages must match what the uses require to be displayed. Key phrase must continue to exist in dynamically changeable pages and critical pages must maintain the same quality from version to version. Accuracy and consistency means that web pages downloaded from one time to the next are still accurate and consist with previous versions. Response time and latency measures the website server response to a browser request within a tolerable performance parameter. It should also test pages of the site that are so slow that a user will discontinue using the site. Performance measures the cycle of browser to web, web to website, website back to web, web back to browser. It also measures the web load based on usage, number of users, and critical times.

The six web-based testing tools used to evaluated these quality assurance criteria were: Rational SiteCheck, Doctor HTML, Dr. Watson, NetMechanic HTML ToolBox, Web Performance Trainer 2.0, and WebART.

1. Rational SiteCheck

Rational Test is an integrated product under Rational Suite Enterprise for the automated testing of enterprise-level client/server applications. Rational Test combines client/server testing, management tools, and a formal methodology for automated testing of cross-Windows client/server applications. Rational Test is comprised of the following component products: Rational Robot (test recording tool), Rational ClearQuest / TT Edition (defect tracking), Rational SiteCheck (web site management), Rational SiteLoad (web site load analysis), TestManager WebEntry (web-based defect entry tool), and TestManager (test planning, management, and analysis tool).

Rational SiteCheck can be used to administer any Intranet or World Wide Web site. Using Rational SiteCheck, the user can manage the folders and files, check links, monitor sites for changes and updates, and examine the site for defects. In addition, the following can be performed using Rational SiteCheck:

- Visualize the structure of Web site through PageView with an effective graphical display, and centering of pages, it not only displays the structure of the Web site, but can also display the relationship between each page and the rest of the site.

- It Identifies and analyzes web pages with active content (forms, Java, JavaScript, ActiveX, and Visual Basic Script) through Active Scan View.

- It filters information using the List View, so that specific file types and defects can be viewed, including broken links.

- It can be used to examine and edit the source code of any web page, with color-coded text, through Source View.

- Another feature called Site Monitor enables monitoring of web sites.

- It can manage or analyze a sub site of the main site using the Site Root settings.

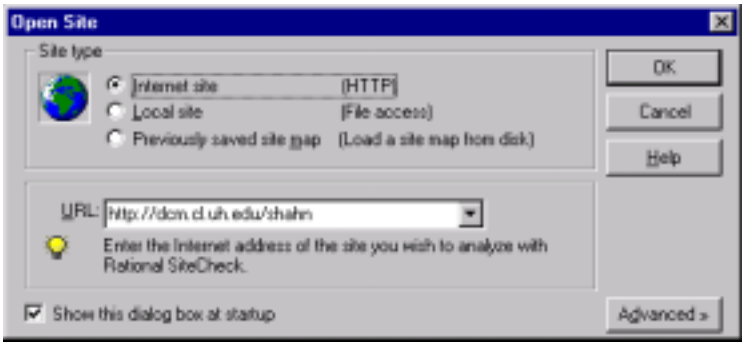
- Rational SiteCheck can update and repair files using the integrated editor or configure HTML editors to perform modifications to HTML files.

- It helps reorganize or maintain files on the web site at the click of a mouse. When the file is moved, renamed, or deleted, LinkWizard automatically repairs broken links.

- Performs comprehensive testing of secure web sites. Rational SiteCheck provides Secure Socket Layer (SSL) support, proxy server configuration, and support for multiple password realms.

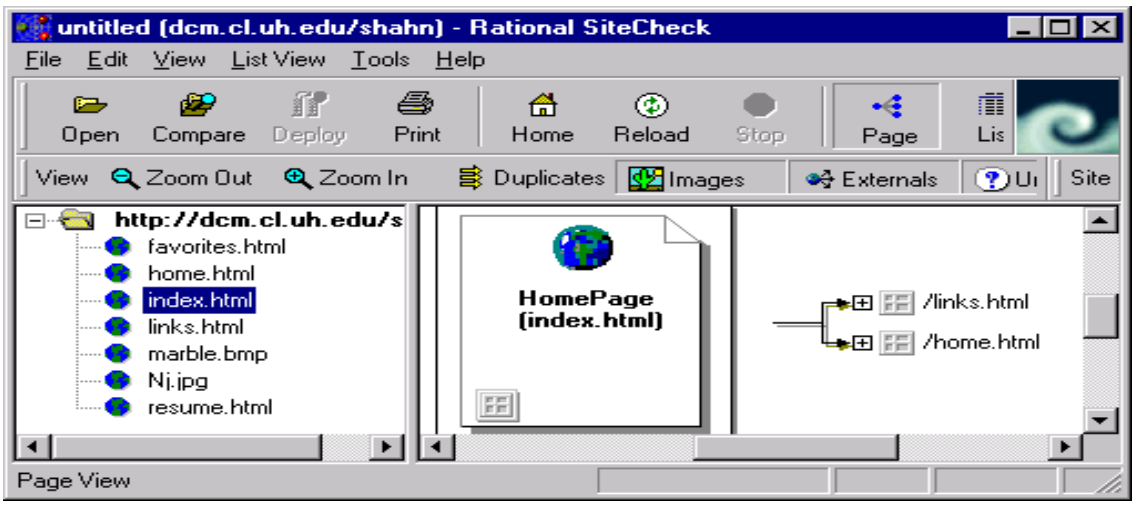
Testing Process And Results Using Rational SiteCheck

Preparing For Testing - This step involves providing the URL of the web application to be tested. The URL can be specified in different domains as indicated in the sample shown below



Testing Process By Rational SiteCheck - Once the URL of the site is specified, Rational SiteCheck loads the site and performs thorough checks using various features. The various testing features and their results are as follows:

PageView Feature - This feature gives the visualization of the structure of Web site with an effective graphical display, and centering of pages, it not only displays the structure of the Web site, but can also display the relationship between each page and the rest of the site. It takes the default as the main page and lists all the input links to this site and all the out going links from this site to the external links. Sample view of the PageView Feature when applied to the users web application



ActiveScan View Feature - A page with active content is any page that contains a form, an ActiveX control, a Java applet, JavaScript, or VB script. ActiveScan View is used to follow links on the Web site that is only available through user input. In ActiveScan view, the selected page appears on the bottom of the screen as it would in a browser. Data can be entered (provided the Web page accepts data entry) and the entries will be registered in the ActiveScan Entries section in the top portion of ActiveScan View, once the form is submitted. This procedure verifies that there are valid links from the active content page. Although ActiveScan View resembles Browser View, links cannot be followed; the user must switch to Browser View.

Site Properties Feature - The Site Properties View displays valuable statistics pertaining to the test Web site. This information is placed into three categories. General Site Information gives information such as the site root, the total number of levels that were read by Rational SiteCheck, and the total number of files located on the Web site. The Site Totals section records the totals for HTML pages, GIF files, External Links, etc. The third section provides statistics on what is contained in the HTML Pages. Totals are provided on the number of HTML pages containing Java Applets, JavaScript, ActiveX Controls, VB Scripts, Frames, and Forms.

The Sample view of this Feature on the test Web Site is as follows:

http://dcm.cl.uh.edu/student/index.html - Site Properties	
Scan Start Time: November 20, 2000 8:43 PM	
ServerType: Microsoft-IIS/5.0	
General Site Information	
Scan Status:	Scan Complete
Total Number of Files:	7
Number of Levels Read on Site:	3
Total Site Size:	0.27 KB

Total Scan Time:	00:00:01
Site Totals	
Total Number of HTML Pages:	5
Total Number of GIF Files:	0
Total Number of JPEG Files:	1
Total Number of Links Not Found:	1
Total Number of Failed Assertions:	0
Total Number of Mailto Links:	1
Total Number of External Links:	6
HTML Page Information	
Total Number of Pages with Java Applets:	0
Total Number of Pages with JavaScript	0
Total Number of Pages with ActiveX Controls:	0
Total Number of Pages with VB Script	0
Total Number of Pages with Frames:	1
Total Number of Pages with Forms:	0
Total Number of Pages with Active Content:	0

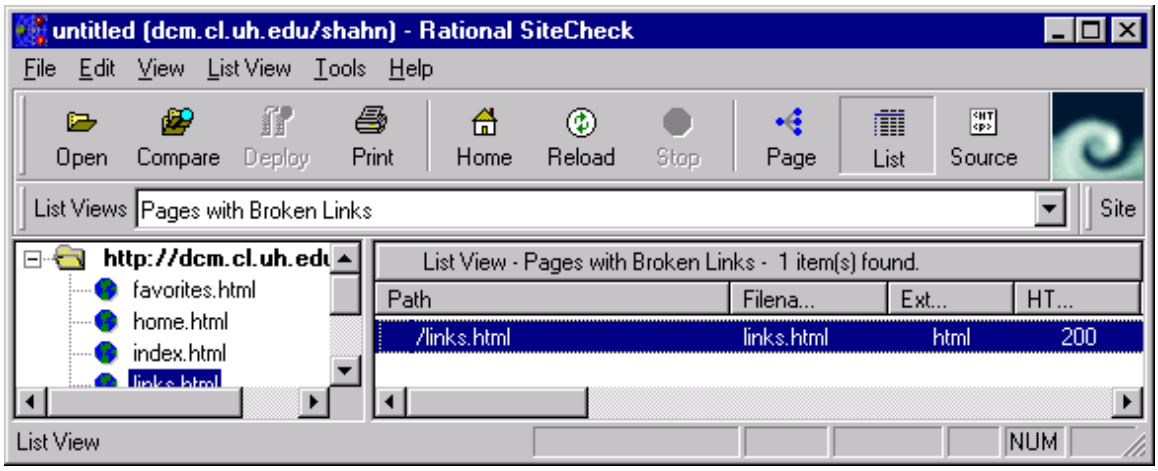
ListView Feature - This view allows the user to see a list of files located on the current site. By default, a List View includes the following information about the file:

Heading	**Description**
Path	The complete path from the root of the site.
Filename	The name of the file excluding the path.
HTTP Code	Used only for Internet sites. When a request is made to a server, a Return code is sent back to the browser.
Title	The title of an HTML document.
Author	The author of an HTML document.
Size	The size of the file, in bytes.
Total Size	The size of the file including the size of other files (such as frames or image files) contained within it.
Created	The date the file was created.
Last Modified	The date the file was last modified.
Inbound Links	The number of links pointing to the file.
Outbound Links	The number of files that the selected file points to.
External Links	The number of links to files located outside the current Web site.
Virus	Used only on sites opened through the local file system. Gives the name of the virus.

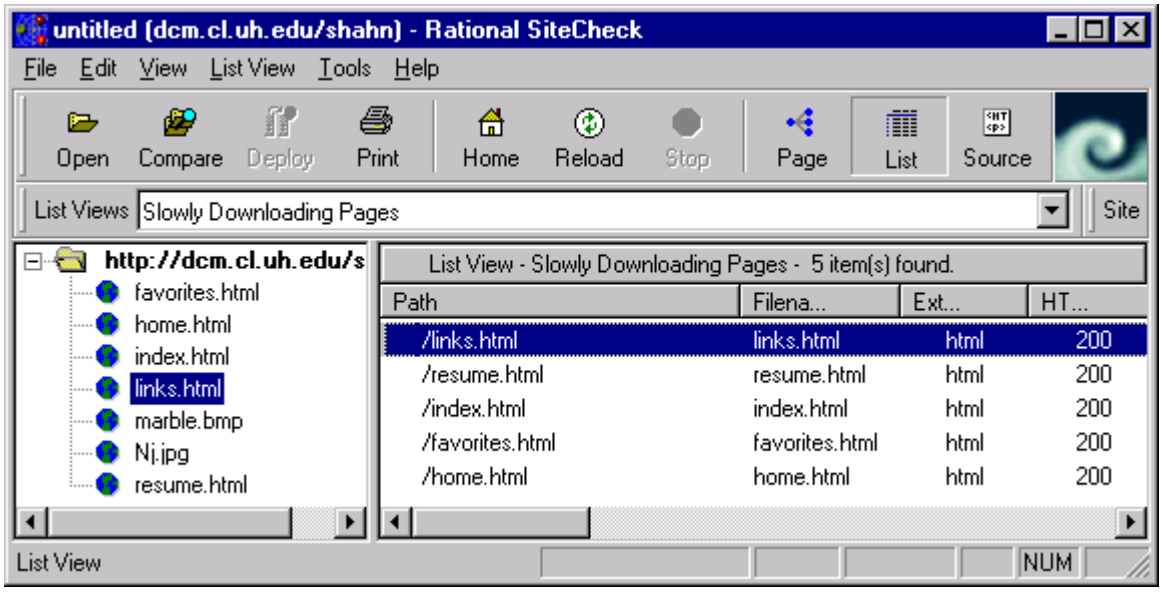
The download speeds columns indicate the estimated time a file takes to download over different connections. This is an especially useful feature when used on a graphic-intensive site.

- In List View, the information can be displayed on all files found when Rational SiteCheck scans a Web site, or the information can be displayed on certain types of files. For example, the user can display pages with broken links or pages with JavaScript.
- In this view various different results can be obtained depending on the assertion that the user specifies.
- For Example: Using the Defects Option of the ListView Feature the use can print information regarding pages that have broken links or gather information regarding the pages that have slow loading time.

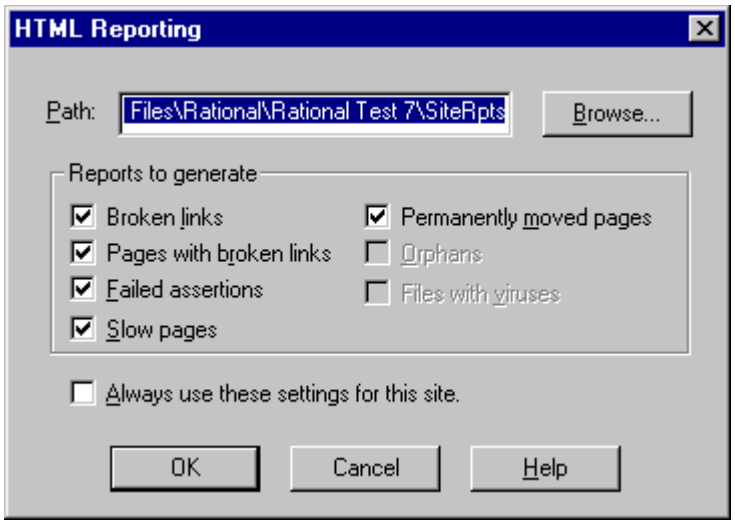
The following is a sample view of the Defect option that gives the information of the pages in the test site that has links, which are broken:



The following is a sample view of the Defect Option, which gives information about all the slowly downloading pages:



Generate Report Feature - This feature generates a comprehensive report of testing the test web site. The sample figure shows all the options for which the Rational SiteCheck can generate reports for:



Here is the sample of the “Executive Summary Report” That Rational SiteCheck produced for this test site:

Rational the e-development company	**Executive Summary** for http://dcm.cl.uh.edu/shahn
Executive Summary Problem Reports Site Properties	

This report highlights key problem areas on http://dcm.cl.uh.edu/shahn as detected by Rational SiteCheck.

Scan Details	Analysis completed on Friday, November 24, 2000 5:12 PM using Rational SiteCheck.
Broken Links	1 (6.67%) of the 15 links on this site are broken.
Pages with Broken Links	1 (9.09%) of the 11 pages on this site have links that are broken.
Slow Pages	5 (45.45%) of the 11 pages on the site have download times exceeding 50s at 28.8 modem speeds.
Permanently Moved Pages	0 (0.00%) of the 15 links are permanently moved. Links to each permanently moved page should be updated to reflect its new location.
Failed Assertions	0 (0%) of the 0 Assertions on this site failed. 0 (0%) of the 0 Global Assertions on this site failed. 0 pages on the site failed Global Assertions.

The above is the HTML summary report that is generated for the test web site. The executive summary report has various links, which give information about the test site like:

- Scan Details – This link gives the various statistics of the given test site
- Broken Links – This link gives the details of pages that are broken
- Pages With Broken Links – This link gives the pages that contain the broken links

2. Doctor HTML

Doctor HTML is an online web page checker by Imagiware. It checks spelling, forms, table structure, form structure, tag usage and validates links. The primary focus of this tool is to provide a clear, easy-to-use Web Interface for report configuration that is relevant for improving your Web page. The user can get Informative, nicely formatted report with hyperlinks to additional information. The Dr. HTML has ability to test Web pages that are password-protected. The user can take single page analysis or whole site analysis.

- **Single page analysis**

In single page analysis, to test the URL, the user can enter the **URL** of the tested Web site in the text window, then select the testing format**.** By default, **Doctor HTML** performs all available tests on the page and displays a report containing only the errors that were found. If the user wants to see everything tested (not just the errors), then select the "Long" format. The user can also check the button "Select from list below" and then just select the available tests. For example, the user can check the home page of School of Nature and Applied sciences of UHCL:

As can be seen from above, the **single page analysis** can take the following nine report options:

- **Check spelling errors in the document** (provide suggestions for potentially misspelled words) - This testing removes HTML directives and accented text before running the document through a spelling checker, eliminating most of the false alarms.

- **Analyze the images** (size, number of colors, etc.) - It loads all of the images in a document, provides the bandwidth consumed by each image, roughly displays download times over a 14.4kbps modem (now the most common speed for dial-up access users. Excessive load times for individual images are highlighted in different shades of red), reports the dimensions of the images in pixels and the number of colors in the image which has a direct bearing on how much bandwidth the image consumes.

- **Test the document structure and flag invalid HTML** - This feature tests the overall document structure except tables, which are dealt with separately. The test looks for unclosed HTML codes, which may cause problems on some browsers. When used in conjunction with "Show command hierarchy", this report can be helpful in hunting down extra or missing HTML tags.

- **Examine image syntax** (flag missing, but recommended elements) - This test deals with one of the most common mistakes in HTML coding: overlooked image command tags. Specifically, it checks each image command for HEIGHT, WIDTH and ALT tags, and reports if they are absent. These tags are important for quick image loading and page formatting, as well as providing information for browsers lacking images.

- **Examine table structure** - This feature tests the table structure on the page. It specifically looks for unclosed <TR>, <TH> and <TD> tags inside a properly defined table (i.e., one which has both an open and close <table> tag). It also reports on <TR>, <TH> and <TD> tags that appear outside of any properly defined table, since these may cause formatting errors on some browsers.

- **Verify that all hyperlinks are valid** - It looks for dead hyperlinks on your pages and reports whether the URL is still present or the server returns an error. To make this feature work with a typical number of links on a page (about 30), the timeout for each link test is 10 seconds. This may cause some slow links to timeout, and the user will have to check them manually. The report also informs the user how large the destination URL is, so that they can check unusually small returns for short messages such as "This page has moved!"

- **Examine form structure** - For those sites that employ forms, this tool can be handy for checking input types and variable names. Currently, **Doctor HTML** only looks at <INPUT> commands, and does not currently test <SELECT> or <TEXTAREA> command.

- **Show the command hierarchy** - This task presents the HTML commands that are found in the document, with regular text removed. The source is indented to reflect inclusion in containers, which is helpful in hunting down extra commands in the code. This option is most useful when combined with one or more of the above structure tests. The outline is displayed in a scrolling <textarea>. If your browser is Javascript-enabled, then clicking on the button labeled "Show Printable Version" will display the outline in a separate window, which can then be printed for easy reference.

- **Shows the page being tested** - If your browser supports Javascript, selecting this option will cause the creation of a window containing your Web page. This allows the user to view both their Web page and the **Doctor HTML** report at the same time.

Site analysis

The Site Doctor is a program that is used to diagnose your entire web site or a subset of pages at once. It simply provide a top level URL to receive a site map showing page connectivity and individual Doctor HTML reports for each page. The site doctor program consists of two components: sitemap and sitedoc.

- sitemap program produces a map of all the web pages connected to a top level URL down to a given depth. The user selects which pages to diagnose based on the output of sitemap.
- sitedoc program produces reports on the desired documents.

Comments

The user can sign up for five free single site analyses or make a whole site analysis of a subset of pages. It is a powerful web-hosting tool. A Site Analysis will descend through the user's Web site looking for pages to test. It will then produce a summary report on all of the problems found, as well as full Doctor HTML reports for each page. The user may purchase access to the program through RxHTMLpro or license the program to run on their local Intranet.

Feature/Option	Cost
Junior Account	$35.00/month
Senior Account	$75.00/month
Additional Bandwidth	$7.50 per 10 MB/day per month
Additional Disk Space	$0.125/MB per month
Additional Login Accounts	$5.00/month
Additional Virtual Host	$25.00 set-up fee and $10.00/month
Domain Alias	$25.00 set-up fee
Real Audio	*free* (HTTP based delivery)
Secure Server	*free*
PHP Scripts	*free*
Personal CGI Access	$10.00/month*
Frontpage Support	$10.00/month
E-mail Alias	*free*
Mailing Lists	*free* (except for bandwidth)
Access Statistics	*free*
Page Validation	*free*

3. Dr Watson

Description:

- Dr. Watson is a free service to analyze a web page on the Internet.
- Dr. Watson understands the latest HTML 3.2 standards, as well as Netscape and Microsoft extensions up through version 4.x.
- It also checks out many other aspects of the site, including link validity, download speed, search engine compatibility, and link popularity.
- It guesses on how long it takes your page to download at various connection speeds.
- In addition to spell checking the page, Dr. Watson also generate some simple word-count statistics, like average word length.
- Finally, Watson can query the AltaVista database to see how many other pages have links to the specified page.
- Dr. Watson is not available for purchase.
- I have used version 4.0 of the tool to test my site.
- For each option, click on the -? - next to it for an explanation of that option.
- For some options, there are additional details available. Click on the  after the option to see the details.

Testing Website Using Dr. Watson

To analyze a single page from the website, in the text window, enter the URL of the Web page. Then select the tests to be performed. A description of the available tests and options is provided:

- **Analyze HTML Syntax** – These option checks the HTML syntax using the options that have been selected for “Browser Extensions Allowed” and “HTML Analysis Depth”.

- **Browser Extensions Allowed –** Netscape and Microsoft have both made “extensions” to standard HTML. This option lets the user choose which, if either, of the sets of extensions will be considered legal.

- **HTML Analysis Depth –** The level of HTML compliance strictness and whether or not style warnings will be included can be decided.

- **Verify Regular Links** – This option verifies that all the links are working. It only checks that if on clicking on a particular link a person gets somewhere and not that the final destination is correct.

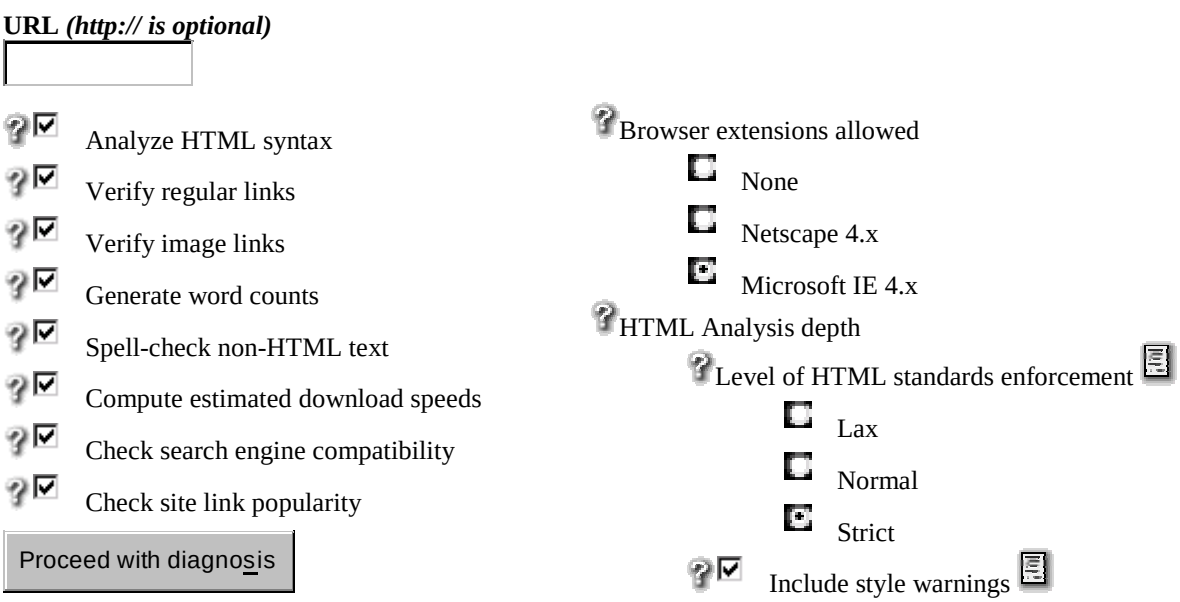
- **Verify Image Links –** This option verifies that links to images are working. It only verifies that the link will load something, it cannot verify that the URL points to the correct image or even an image at all.

- **Generate Word Counts –** This option gives the number or words, average word length and number of unique words on the page after taking out all HTML tags.

- **Compute Estimated Download Speeds –** This option estimates how long the page takes to load into a browser at various connection speeds.

- **Check Search engine Compatibility –** This option checks to see how well the page will cooperate with search engine and indexing robots by analyzing the META tags.

- **Check Site Link Popularity –** this option queries the AltaVista search engine to see how many other pages have links to this page.



4. NetMechanic HTML ToolBox

Description: NetMechanic HTML ToolBox scans web pages and interrogates the structural quality, content accuracy and consistency of the page. It repairs common HTML errors like finds broken links, gives help with HTML tags, check load time, schedule automatic tests and is easy to use. HTML Toolbox comes with these state-of-the-art tools:

- Link Check
- HTML Check and Repair
- Browser Compatibility
- Load Time Check
- Spell Check
- It tests sites of up to 400 pages.

- **Identifies and fixes the majority of common HTML errors.** - HTML Toolbox will spot common HTML code errors, automatically fix the code, and generate a repaired file for the user to upload. It doesn't fix every problem, but it will still identify those problems that it can't fix.

- **Automatic testing.** - Schedule weekly, biweekly or monthly tests of the site. It will notify the user by email when the results are ready.

- **Testing on demand.** - Log into the user account and test the site at any time.

- **Configurable.** - Tell the user what tests they want to run. Tailor our tools to the user preferences.

To Test Website Using NetMechanic HTML ToolBox

- Enter URL of the WebPages and then enter email address.
- The user has option of selecting one page or 20 pages
- Results are sent by email but if the user wants an instant result and the WebPages is not too long, then the user doesn't need to enter email address and the results are shown within a minute.
- It is displayed as follows:
- Check user links, HTML, page load time, spelling, and more!

URL:	http://
Email:	
Pages:	1 Page 20 Pages
Free Monthly Tune Up Free Monthly Site Tips Newsletter	

Customize Your Test Go! Go!

By default all five tests are carried out, but there is an option to customize the tests as shown below:

(1) Enter Your URL:	http://
(2) Enter Your Email Address:	
(3) What tools would you like to use?	
Link Check HTML Check & Repair Browser Compatibility	Load Time Check Spell Check
(4) How many pages should we test?	One Page 20 Pages
(5) Get our free monthly newsletter?	Yes!

Thus, NetMechanic HTML ToolBox is good tool to check a website and make the content free of errors like broken links, syntax errors and it also measures the load time of the WebPages. When testing is finished, a detailed summary report is sent via email or instantly within minute. Herewith, I have attached summary report of the website I tested.

5. Web Performance Trainer 2.0

Web Performance Trainer 2.0 simulates multiple users hitting a web site to find performance bottlenecks, increase performance, or do capacity planning. Web Performance Trainer 2 is a solution to the problem of finding out how many users a web-based application can handle. It's designed to be up and running in a few minutes, so it is possible to get an accurate picture of the web sites scalability in under an hour. Once the user has the basic information they need, they can re-run the tests while tuning out the back-end or swapping out equipment until the optimal combination is found.

Because Web Performance Trainer is based on recording browser/server interaction rather than emulating a browser, it is extremely accurate. By using recording, the user can see exactly what is happening between the browser and server, and pinpoint bottlenecks using the analysis tools.

Testing Methodologies

To get Web Performance Trainer 2 free, register on their web site. Once that is done, the user will receive an email with the location and instructions for downloading the installer. To download the installer, chose the right file for the operating system, and click on it to download.

- **Pick a Test Machine** - The test machine should be at least a 200MHz system with 64 megabytes of free memory. The test machine should be comparable to the server. In order to run Web Performance Trainer the user needs to have a high-speed network connection between the web server they are testing and the machine where Web Performance Trainer is installed.

- **Run the Installer** - For Windows, double-click on the install program to execute it, and follow the program's instructions. For Solaris, after downloading, open a shell and cd to the directory where the installer was downloaded. At the prompt type: sh ./wpt2_0.bin.

- **Install The License** - The license file is named "License.class" and is sent as an email attachment. Save it to disk in the directory <INSTALL_DIR>/com/webperfcenter, where <INSTALL_DIR> is the directory where the user installed the Web Performance Trainer.

- **Test The Installation** - running Web Performance Trainer can test the installation.

- **Configure the Browser** - Configure the browser to use Web Performance Trainer as a proxy server. Netscape and Internet Explorer, as well as other browsers, support the use of proxy servers. Web Performance Trainer sits between the browser and the web server, recording all communication between the browser and the web server. For Internet Explorer, Bring up the Internet Options Dialog by choosing the Tools Menu, and then Internet Options. Click on the LAN Settings button to view the screen below. In the Proxy Server section select "Use a proxy server". Type in "127.0.0.1" for the Address, and 8081 as the port. Configure the HTTP connection for the

browser for a proxy using the "Advanced" tab of the same Options Dialog. Make sure that the "Use HTTP 1.1 through proxy connections" option is unchecked.

- **Testing the Proxy Configuration** - To test the proxy configuration of the browser, start Web Performance Trainer and try to use the browser as normal. The best web server to use in the test is the web server on the local LAN that the user is planning to test. Try to browse the web site normally and try to view any of the web pages. If the web page does not appear as normal, try setting the browser back to its normal configuration and verify that the web page is currently accessible.

- **Record Business Cases** - The next step is to think about how users interact with the web site, and divide up the interactions into business or use cases. Typical business cases include such things as:
  - Signing up for membership
  - Searching for a product
  - Purchasing a product
  - Visiting the product support page

  Select Record->New Business Case and a dialog will ask to name the business case. Click OK. The business case will appear in the list of business cases that appear throughout the program. Select the business case by clicking on it, and either select Record->Start or click the start button to start recording. Now start the browser if it isn't started already, and view the web page(s) that comprise the business case. This can be any combination of online forms, JavaScript, or applets. As the user records, the tables below will fill up with the HTTP commands that were sent to the server: The HTTP commands will be parsed into Web Pages and URLs. If the browser works, but the web pages are being displayed slowly, be sure it isn't the web site or the network that is causing the slowdown, finished, click on the Stop button to stop recording.

- **Browse Web Pages** - Once there are web pages recorded the user can browse through them, examining the low-level headers that were sent between the browser and web server. There are three main tables in the Recording Tab. The first table lists all of the business cases. Clicking on the name of a business case, displays the contents of that business case in the Web Page Table that is in the middle. Clicking on a row in the Web Page Table displays the contents of that web page in the URL Table. This approach lets Web Performance Trainer support the most complex web pages, which may contain links to other web servers on a variety of ports.

- **Preparing the Test Machine** - Running a performance test is a CPU and memory intensive operation. In order to get the most out of the test machine, and insure the most accurate statistics make sure there are no background processes running on the test computer. While the performance test runs. Web Performance Trainer 2.0 will monitor the CPU usage and make sure the machine is not overloaded. When the CPU Usage/Load Average gets too high Web Performance Trainer will stop adding new virtual users. Note that if the machine's CPU load is too high at the beginning of the

test, no users will be added at all. At this time check your machine's CPU load average. To check the load average on Windows NT, bring up the Task Manager by hitting control-alt-delete and clicking on the Task Manager button. The CPU Usage is displayed on the bottom of the dialog as a percentage. Normal usage when the computer is idle should be under 5%. Check the list of applications to see if any of them or than the Task Manager or Explorer are taking up CPU time.

- **Configuring The Performance Test** - Start with a low number of users initially and then increment the number of users every minute. All this can be specified on the Playback tab. Duration's can be in units of hours, minutes, or days. The duration of the test should change depending on the testing goals. To get just an idea of the speed of certain operations on the site, useful performance information can be gained for tests that are a few minutes long. We can then tweak parameters in scripts or machine configuration and see if it has an affect on performance.

- **Running the Performance Test** - Once the performance test is configured clicking on the Start button can start it. The test can be ended at any time by clicking on the Stop button.

- **View Statistics** - The statistics view gives numerical information that allows the user to determine in a bottleneck has occurred. The same information can be viewed in graphical form in the Graph Tab. The statistics view shows data values for high and low level objects, either for an entire business case, or for an individual image or back-end script. The statistics view consists of a test results browser, on top, and a test detail table below.

- **View Graphs** - To view graphs of the performance statistics click on the Graph tab. The graph tab helps to create and keep multiple graphs consisting of any of the statistics available during a performance test.

6. WebART

WebART is a test-automation tool developed by Online Computer Library Center OCLC for testing World Wide Web, Internet, and intranet applications and content. It provides a direct, cost-effective solution for creating, executing, and evaluating automated tests that verify an application and web site.

WebART is a comprehensive solution to web-testing needs, addressing all major aspects of validating your applications, including:

- Link Validation
- Load and Performance Testing
- Script Capture/Replay
- Automated comparator with extensible masking
- Full-featured Scripting Language

Testing Methodologies

WebART is available free from their website. To download the installer, click on it.

- **Installation** - To install on Win95 & Windows NT start the Installation Program. The WinZip Self-Extractor window displays. Click on the Setup Button to unzip the files and launch the WebART setup program. Install to the displayed directory. Then choose a Browser.

- **Link Checking with SmARTMonkey** - In the At initial URL field, enter the URL of the page at which the user wants to start. Select the Only check documents on this server checkbox. Select the Only check documents to a depth of radio and enters 99 in the input box. Clear the remaining input boxes. Click the Start button. The SmARTMonkey process begins checking for bad links at the specified starting point. An ongoing progress report is displayed in the bottom frame.

- **Creating Test Scripts** - Test scripts are a sequence of automated user interactions that are used to execute test cases for functional and regressions testing and produce load for load and performance testing. Test scripts can be created in the following ways:

    1. **Creating Test Scripts Using SmARTMonkey -** In the At initial URL field, enter the URL of the page to start. Select the Only check documents on this server checkbox. Select the Only check documents to a depth of radio and enter 99 in the input box. Enter the name of the script to be created in the Create text box (10 or fewer alphanumeric characters). Check the Script check box. Clear the remaining input boxes. Click the Start button. The SmARTMonkey process begins checking for bad links at the specified starting point and creating a script to visit each page. An ongoing progress report is displayed in the bottom frame.

    2. **Creating Scripts Using Capture -** In the Script field, enter the name of the script to be created. In the Title field, reenter or modify the script title. For Protocol, select HTML. In Initial URL, enter the URL of the page to start capturing. For Options, select no session. Clear the browser's cache in order to ensure that all images will be captured in the baseline. Click the Capture button to start the capture session. The browser loads the page specified in Initial URL, and the Capture control window appears with the Stop, Pause, and Comment controls. Browse on the website so that script is generated. When done, click the Stop button in the Capture Control Window

- **Executing Scripts -** Verify that the Project, Target, and Interface settings are correct. In the Script field, enter the name of the script to execute. Click the Execute button. The user simulator process starts up in a separate window and executes the script.

- **Load Test -** Verify that the Project, Target, and Interface settings are correct. In the Script field, enter the name of the script to execute. In the Number of Users field, specify the initial number of users to start with. Enter the maximum number of new

connections, in connections per minute, in the Connect Rate field. Click the Start button. The load test execution process starts up in a separate window and begins the test. To stop the load test, in the Command area of the Load Test Control Window, enter stop to shut down the test gracefully, allowing each user to receive responses for in-progress requests or enter stop to shut down the test immediately. Dismiss the Load Test Control Window.

Summary

Six web-based testing tools were used to evaluate the quality assurance criteria based on: timelines, structural quality, content accuracy and consistency, response time and latency, and performance. The tool were: Rational SiteCheck, Doctor HTML, Dr. Watson, NetMechanic HTML ToolBox, Web Performance Trainer 2.0, and WebART. Rational SiteCheck is a primer quality assurance set of tool that provides all of the QA criteria. This set of tools is not inexpensive, but worth the investment for e-commerce. The other tools have similar capabilities and feature but lack the extensive background of the Rational software unified development process. For the beginning web developer and the student not familiar with QA techniques the free tools should be used to insure the quality of their sites. With these tools anyone developing or managing a website even a novas to the software development process has the capability to perform QA on their site.

References

H. Berlack 1992. *Software Configuration Management.* New York, NY: John Wiley & Sons, Inc.

J. Buckley 1993. *Implementing Configuration Management, Hardware, Software and Firmware.* Los Alamitos, CA: IEEE Computer Science Press.

G.G. Schulmeyer and J.I. McManus1992. *Handbook of Software Quality Assurance.* New York, NY: Van Nostrand Reinhold.

David Whitgift 1991. *Methods and Tools for Software Configuration Management.* New York, NY: John Wiley & Sons, Inc.

http://watson.addy.com
http://www.cigital.com/marick/
http://www.methods-tools.com/tools/frames_testing.html
http://www.netmechanic.com
http://www.oclc.org/webart/
http://www.rational.com
http://www.softwareqatest.com/qatweb1.html

QW2001 Paper 8W2

Mr. Kim Davis, Mr. Robert Sabourin
(My Virtual Model Inc.)

Exploring, Discovering and Exterminating Bug Clusters In Web Applications

Key Points

- Areas of uncertainties define bugs clusters having natural affinities of causes
- Bug Clusters are easier to locate and manage than individual bugs
- Root-cause analysis & risk-based bug clusters fixes help to converge faster on e-com sites

Presentation Abstract

E-commerce development demands fast, incrementally useful, results. We propose to simplify the bug flow process by going after bug clusters, instead of individual bugs which are usually hiding one another. These clusters come from areas of uncertainties associated with missing, incomplete or rapidly changing requirements or specifications. The clusters are composed of the bugs that have natural affinities of causes.

We first identify these clusters, and track them. We help perform fast probabilistic root-cause analysis on clusters to assist in determining probable causes. We finally prioritize the clusters extermination and iterate throughout the project.

This method aims to achieve faster convergence in short timeframes. Real examples applying to e-commerce applications will be used.

About the Author

Robert Sabourin (rsabourin@amibug.com) is an author, lecturer, and president of AmiBug.Com, a firm specializing in software management consulting, teaching, and professional development. Robert is the author of the popular children's book I am a Bug (ISBN 0-9685774-0-7) which explains what SQA folks really do at work.

Kim Davis graduated in Physics in 1990, and in Computer Science & Psychology in 1994. As a Research Agent he investigated ways of speeding up the Internet by simulating network-optimisation learning algorithms. In 1997 Kim co-founded a web site design company called Inter@ction Technologies to help companies develop database-driven web sites. As Technical Lead, Director of Integration and Director of Operations of the My Virtual Model Team, Kim has helped develop the technology behind the e-commerce web sites of My Virtual Model inc. The

Service he manages, called SET (Software Engineering Team), offers ways to the whole company to improve people / projects / product / processes effectiveness in terms and productivity and quality.

Exploring, Discovering and Exterminating Bug Clusters in Web Applications

Kim Davis, MyVirtualModel, Inc.
kim@myvirtualmodel.com
www.myvirtualmodel.com

Robert Sabourin, AmiBug.Com, Inc.
rsabourin@amibug.com
www.amibug.com

Quality Week - 2001
San Francisco

My Virtual Model

AmiBug.Com, Inc.

Overview

- Method Motivation & Project Presentation
- Risks: Business, Technical
- Potential Areas of Instability & Test objectives
- Exploratory Testing & Mapping
- From Mapped Areas of Instability to Bug Clusters
- Finding, Prioritizing and Exterminating Clusters
- Conclusion
- Future work
- Web References

My Virtual Model

Motivation

- 80/20 Rule: Can we save lot's of time and effort by focusing on larger structures of bugs, i.e. *Bug Clusters*?
- Apply successive approximations
- Identify the cluster – without finding all of its individual bugs!
- Identify, Prioritize and Exterminate at the Bug Clusters Level
- Fix bugs that you did not find!

My Virtual Model

My Virtual Model Community

- **My Virtual Model Concept & Community**
- **3D, Web and Fashion Industries Collide**
- **Visualization**
- **Personalization – Virtual Identity**
- **Network of Interoperating Web Sites – Mobility**
- **Affiliate Services & E-Commerce**

My Virtual Model

My Virtual Model Community

Business Risks

- Fixed timeframe projects tied to industry dates for seasonal collections of garments
- Community is large and demanding in terms of consistency, performance, reliability
- It has to work and it has to be on time
- Functionality always tied to market "$$" related revenues e.g. a new type of model can be introduced to target a new market (for example Tall Men)
- New business model is evolving as project continues!
- Organizational change
- Requirement turbulence

My Virtual Model™

Example Project

- Constraint from moment clothing in-hand to release - max 8-12 weeks
- Community per client includes 100s of thousands of users - target large retail clients - we will get many many hits fast
- No forgiveness in retail fashion business - (nor Web business)!
- Trading down functionality to hit a release target is problematic
- Business requirement - release date moved up!

My Virtual Model

Technical Risks

- Technology change Management
  - Fast technology churn
  - Use technologies in production environments with limited knowledge of baseline
  - Limited time for evaluation, training
- Staff change
  - Programmer not familiar with entire code base - could miss something
  - Programmers not familiar with “live” production parameters - even if the developer knows how to make it work on his own machine (NT vs UNIX)
- Requirement turbulence

My Virtual Model

Example Technical Risks

- Many parallel activities not easy to synchronize
- New application server framework, business logic layer
  - From JServ to JRUN
- New presentation layer
  - to JSP model
- Some reused code at risk because running in a new context
- Changed requirements on technologies used

My Virtual Model

Testing Objectives & Risk

Mon Mannequin Virtuel

Project Name:	
Author:	KD
Revision:	3/30/01

ID	Summary	Note	Proposed Owner	Relative Importance Suggested % of Total Testing Effort				Consequences of Failure (Point Form List)	
				Develop-ment Lead	Product Manage-ment	Testing Lead	Average	Business	Technical
Proj-HLTO-01	Functionality	Suitability, Accuracy, Interoperability, Compliance, Security	Product Management	30	20	25	**24**	Loss of confidence - High Maintenance & patches	High Maintenance
Proj-HLTO-02	Reliability	Maturity, Fault Tolerance, Recoverability	Product Management	20	30	30	**27**	Loss of confidence - High Maintenance	High Maintenance
Proj-HLTO-03	Efficiency	System behaviour over time, Resources Usage	Development Management	20	25	15	**22**	Loss of confidence - High Maintenance	High Maintenance
Proj-HLTO-04	Usability	Understandability, Learnability, Operability	Product Management	10	15	20	**14**	Loss of user interest - Loss of customers	
Proj-HLTO-05	Maintainability	Analysability, Changeability, Stability, Testability	Development Management	10	5	5	**7**	Maintenance Difficulty	
Proj-HLTO-06	Portability	Adaptability, Installability, Conformance, Replaceability	Development Management	10	5	5	**7**	Difficulty to get clients on different platform	Lack of interoperability
			Total	**100**	**100**	**100**			

My Virtual Model

Potential Areas of Instability

- Functions particularly related to new technologies
- New stuff or changed stuff (Turbulence Index)
- Functionalities added after design due to business change

My Virtual Model

Exploratory Testing Approach

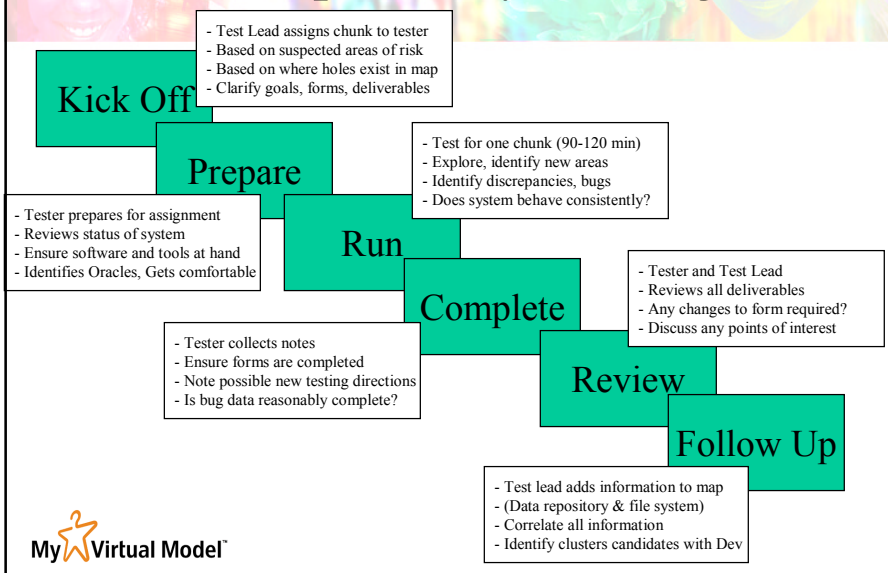
- *"In operational terms, exploratory testing is an interactive process of concurrent product exploration, test design and test execution."*

  - James Bach

- Test lead triage testing in chunks building up a "map"
- Assignment based on looking for areas of potential instability
  - Looking for clusters - observe aggregate results from all testers

My Virtual Model

Exploratory Testing



My Virtual Model™

Finding a Cluster from Bugs Identified

- How should you/could you or do you identify a cluster?
- Correlation between bugs identified to get an understanding of whether they can be grouped
- Mapped Areas of Instability define clusters composed of the bugs that have natural affinities of causes
  - Requirement or Technology Turbulence
  - Copy-paste and reuse of code without checking

My Virtual Model™

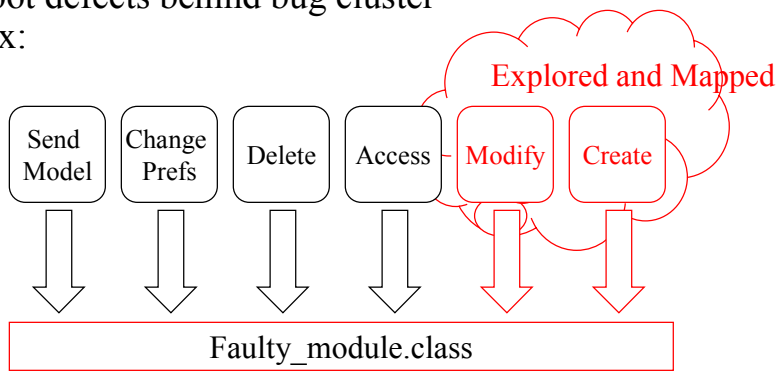
Clusters of Bugs

- What makes a Bug Cluster?
- Example Metrics Definitions
  - Functionality - several bugs related to same functionality are discovered
  - Reliability - different functions fail in similar way
  - Efficiency - several operations are similarly using resources inefficiently
  - Time - several content sources are out of sync
- Could indicate a process-related problem

My Virtual Model

Cause of Clusters

- Test team works with developers to find common root defects behind bug cluster
  Ex:



My Virtual Model

Prioritize Based on Business Impact

- Instead of prioritizing on a bug by bug basis, we work on a cluster (on a higher level)
- Less red tape
- Nice for speed!
- Can
  - Fix cluster
  - Do not fix cluster
  - Treat bugs individually
  - Detailed analytic testing

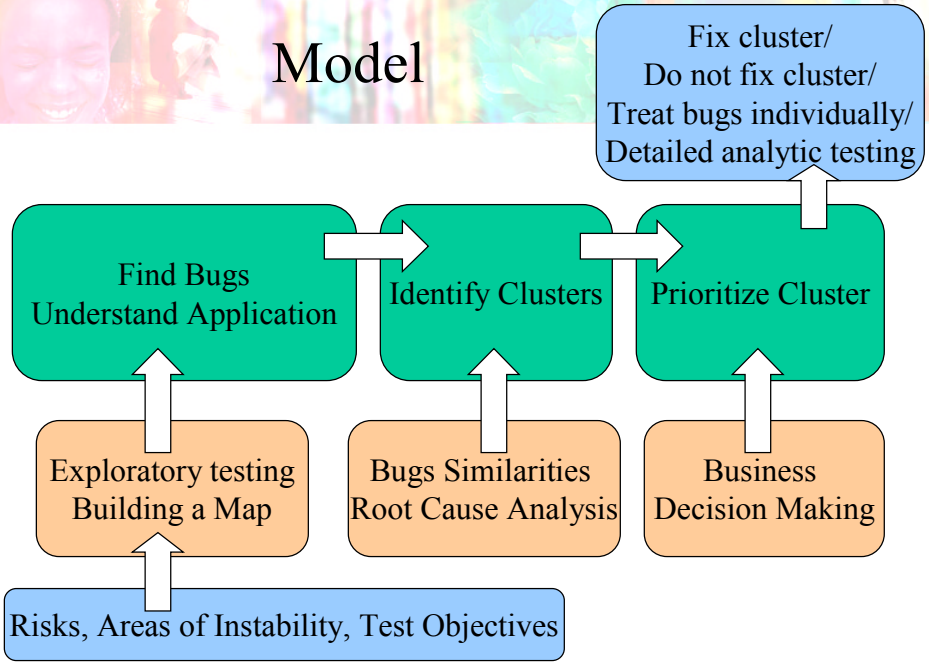
My Virtual Model

Practical Results

- Extermination techniques, developers and testers working together (fast)
- Try to identify probable root causes of sets of seemingly related bugs – Use empirical knowledge
  - Testers and developers worked well together
  - Improved developer awareness of tester role
  - Buy-in to approach
- Analysis based on root cause identified an additional 30%, so for every 10 bugs identified 13 were fixed
- **Continue using technique!**

My Virtual Model

Model



Conclusion

- Relevant in our context
- Shortening bug management overhead
- In less time we were able to deal with more bugs
- Flexible Method – Hybrid
- Buy in at all levels!
- In a turbulent environment, this will likely be a good approach on future projects

My Virtual Model

Future Work ...

- As method is applied to more projects we will better document and generalize it
- Exploratory Testing can be replaced by more analytical approaches as feeding method
- So far - developers, testers and management all like method - and the results
- Cluster Testing Method's Effectiveness & Efficiency – Using it to its fullest potential – Limits of applicability
- Will evolve based on REAL WORLD needs

My Virtual Model

Web Reference Slide

- **www.satisfice.com**
  - **James Bach web site, exploratory and risk based testing**
- **www.testing.com**
  - **Brian Marick web site, articles about exploratory testing**
- **www.amibug.com**
  - **Robert Sabourin Web site, various presentations**
- **www.mvm.com**
  - **My Virtual Model Web site**
- **www.stickyminds.com**
  - **Much relevant content**

My Virtual Model

Exploring, discovering and exterminating Bug Clusters in Web Applications

Kim Davis, SET SQA Director, My Virtual Model inc.
Kim@myvirtualmodel.com
www.myvirtualmodel.com

Robert Sabourin, President, AmiBug.com
Rsabourin@amibug.com
www.amibug.com

Abstract

In e-Commerce applications, time attrition, fleeting turbulent requirements and lack of documentation make testing an exercise in managing people who are entangled in constraints. Our experience shows that a good risk-based bug flow process is essential. However, tracking myriads of little web site bugs in a fast-paced environment is difficult in practice. The good intentions of web integrators, scripters and web managers often disappear, in the name of competition, in front of customers demands of an ever-faster site development.

Fortunately there may be a way of dealing with this difficulty without sacrificing features or quality in such an environment: *why not stop tracking the myriad little web sites bugs* – and *start going after the bug clusters!*

Bugs are social animals, they tend to associate by affinity of causes. They may come from a variety of causes. Major causes are usually related to requirements, be them incomplete, misunderstood, unclear etc., and turbulence, i.e. the effect of change of a project variable in a short timeframe.

This results in the e-Commerce application having areas of uncertainties, which may require large amounts of rework. These areas of uncertainties define clusters composed of the bugs that have natural affinities of causes.

So instead of tracking hundreds on individual bugs in an extremely short timeframe, which is not efficient, we instead focus on finding the important areas of uncertainties, which are easier to locate and track than individual bugs. We then perform fast probabilistic root-cause analysis on them to determine probable causes pairing developers and testers to identify and correct the defects causing them.

Motivation

80/20 Rule

We assume that generally the 80/20 rule can be applied to our testing activities. It is generally assumed that 80% of the important bugs are identified with 20% of the testing effort. It is unfortunate that we can not figure out in advance which 20% of the testing effort will find these important bugs. Can we save time and effort by focusing on larger structures of bugs, i.e. *Bug Clusters*?

One thing we know for sure is that we are severely restricted in the amount of time available for testing. So we want to focus our testing efforts on finding as many important bugs as possible in a short time, and present to management the state of the system under test so that informed business decisions can be made.

Apply successive approximations

Optimization techniques involve finding the best solution, or fit, to a problem by running a series of experiments, varying certain parameters each trial and then judging whether the results are optimal, or closer to optimal, based on our previous results and experiences. Purely statistical models are sometimes used in software testing (such as clean room based testing) to confirm that sufficient test cases have been run to prove or disprove a reliability hypothesis.

Bugs are caused by defects. Defects are introduced into a system by people using processes or tools. Defects may be introduced at any time of the development process, requirement, analysis, design, development, integration, deployment etc. The bugs found during testing are a very important input to the detection of defects in the system under test.

In the context of projects at MyVirtualModel, Inc. we know that we will fix the bugs deemed important from a business and technical point of view. Indeed the priority of bugs, indicating if and when it must be fixed, is a *business decision*.

If we find some bugs we may suspect that there is some a relationship between them. This can be considered a weak relationship which, as a result of more testing, and further bug identification, can be made successively stronger.

We use an iterative approach when testing the system. Each iteration is based on exploring the application to further enhance our knowledge of the software and validate any hypothesis about functional areas of the application exhibiting instability.

Identify the cluster

Ideally we would like to confidently identify a bug cluster without having to finding all of its individual constituent bugs! This would save us a lot of time and effort. But how can bug clusters be identified?

Bugs are social animals, they tend to associate by affinity of causes. Our practical experience tells us that after a project, when reviewing data about bugs identified and associated defects corrected, there is generally a series of bugs related to the same root cause. If we had only found one bug of a certain "family" or cluster, then we could have fixed the lot of them without even being aware of their existence! As Brian Marick defines in "Evaluation Test Suites", Conference Proceedings, Quality Week 2000, (with James Bach and Cem Kaner) a perfectly effective test suite is one that reveals at least one failure for every fault in the program.

Prioritize and Exterminate Bug Clusters

Once suspected bug clusters are identified we can prioritize them relative to other bug clusters in the system. We can make a cluster based decision which will apply to bugs which have not yet been identified.

Fix bugs that you did not find!

By fixing the fault in the cluster of bugs we correct failures which may have existed but were not identified and recorded.

My Virtual Model Community

My Virtual Model concept and community

My Virtual Model has created a network hosting a virtual community. Members of the community create electronic replicas of themselves, called Virtual Models. Shoppers can experiment with fashion, dressing their Virtual Models with virtual garments without the intimidation of store dressing rooms.

Users log on to the site to begin building their model. By answering specific questions, users can create a virtual replica of themselves with exact measurements, skin and hair color. Users can then try on various looks using their model in a virtual dressing room. Shoppers can choose to accept advice from a "Fashion Advisor" on the basis of their body type, coloring, and fashion goals.

Over 1.5 million models have already been created at MyVirtualModel.com affiliated sites.

Forrester Research described My Virtual Model™ as "the most viable alternative" among tools which support online apparel sales.

3D, Web and Fashion industries collide

Requirements for all My Virtual Model development efforts include 3D animation, the latest is Web GUI and interactive technologies. My Virtual Model clientele are members of the Fashion industry which operates under very tight seasonal delivery pressures. You cannot delay the spring fashion season due to a software bug! All projects have fixed timeframes.

Visualization

My Virtual Model provides the ability to mix and match garments and colors that are not only photo-realistic but are also right-sized to eliminate concerns of true fit or appropriate sizing. It includes :

- Photo realistic images of garments
- Close-up views of garments (Zoom in)
- Multiple views of garments (360 degrees rotation)
- Try-on of the garment on body with real measurements

Personalization – Virtual Identity

Virtual Models are personalized for each member of the community. All characteristics of a model can be adjusted as the end-user sees fit. If a new characteristic is modeled, then special consideration must be made for previously existing models to operate with the new characteristics. Backward compatibility is critical.

Users build what is called a Virtual Identity. The first layer is a physical representation in the form of the 3D Virtual Model, which is tied to the garment try-on service and the e-apparel market. Other layers tied to other services, for example financial, may be added later so that the model will resemble more closely the user in order to better serve it.

E-Commerce

My Virtual Model technologies are an integral part of the e-Commerce offering from client vendors. We must be able to provide specifics regarding the style, color, size and all other characteristics of garments purchased via B2B interchange of order information. Encrypted XML is used extensively to interchange the required information. My Virtual

Model must be non-intrusive to the client e-Commerce site and must be able to interact with all clients solutions.

Network of interoperating web sites – Mobility

My Virtual Model is implemented as a series of independent web servers. For each client there exists at least one of each of the following:

- Client e-Commerce
- Fashion Server Site
- Model Server Site
- Various Data Servers

A three tier architecture is used, using the familiar Web Server / Application Server / Data Server layering.

Virtual Models are mobile. Users can access their models on any of the affiliate sites, can travel from site to site, and can send their Virtual Model by e-mail. Mobility pumps the heart of the My Virtual Model Network.

Business risks

Business risks must be reviewed on a frequent basis to ensure that testing priorities are aligned with the corporate realities. We may have to react to a competitive threat or new short time frame opportunity.

Generally business risks are driven by:

- Fixed timeframe projects tied to industry dates for seasonal collections of garments
- Large and demanding User community (consistency, performance, reliability)
- Functionality always tied to market "$$" related revenues e.g. a new type of model can be introduced to target a new market (for example tall men)
- New business model is evolving (including revenue) as project continues!
- Organizational change
- Requirement turbulence

Typical Project Example

The typical project encountered has a tight time frame. From the moment clothing is in hand (hundreds of garments), to commercial deployment, takes a maximum of 12 weeks elapsed. Often projects require less than 8 weeks to complete.

The user community per customer includes 100s of thousands of users, all clients are large retailers, thus generating many hits fast!

The retail business is similar to the Web business in that customers offer no forgiveness! If the software or site is not operational they will move elsewhere.

It is very problematic to remove, or trading down, functionality in order to hit a release target. In fact quite the opposite happens and due to compelling business requirements the release date is often moved up!

Technical Risks

The projects involve considerable technical risks. Technologies used are continuously changing and evolving. The company addresses a segment of the market which continuously demands the latest and greatest web widgets!

So our projects experience fast technology churn. The technologies being used in production releases are new to the company, marketing team, developers, testers and system integrators. Developers and testers are learning about the limitations of the solutions on the fly and often have to adapt architectures in mid project.

Technologies are often used in a production environments with limited knowledge. We have limited time for training and for evaluation of the technologies.

Staff turnover can have a devastating effect on such projects. New programmers are not familiar with the existing code base and can accidentally miss something when adding a new feature or adapting to a new embedded technology.

Developers are often not familiar with the "live" production parameters. Often the customer has not decided whether parts of the server component will work on NT or Unix based computers.

Whenever requirements change there is a risk of changing development priorities which in turn will impact the code base. How do we elegantly drop what we are doing while not accidentally breaking something!

Changes in mid-project included:

- A new application server framework, at the business logic layer, moving to an Apache server with JServ to one with JRUN.
- A new JSP driven presentation layer.

Important technical risks were also introduced due to the reuse of existing code in a new context.

Testing Objectives & Risk

Summarizing the specific technical and business risks associated with a project lead to the identification of primary testing objectives. We were able to prioritize these based on our knowledge of the areas of highest risk. Testing effort was spread across testing assignments proportional to the associated risk.

We considered the potential areas of instability to be:

- Functions particularly related to new technologies
- New stuff or changed stuff
- New functions added due to business

Exploratory Testing Approach

An exploratory testing technique was used to help find bugs, and identify problem areas of the application. To quote James Bach, *"In operational terms, exploratory testing is an interactive process of concurrent product exploration, test design and test execution."*

Exploratory testing is a systematic approach. It allows for the concurrent design and execution of tests. As the application is explored a detailed record is kept of the areas explored. Information is captured on standardized templates. Information gathered includes:

What was tested?
How was testing done?
What was discovered about the application?
What bugs were identified?
What oracles were used in an attempt to validate results?
What new discoveries may be of interest to future exploratory testers!?

A senior test lead is responsible for triaging test assignments to various members of the testing team.

The test lead uses knowledge gathered in testing combined with the risk assessment and the potential areas of instability to define the next series of test assignments.

The test lead is operating at the testing mission control. Different testers act as emissaries exploring along the directions indicated by the test lead. The test lead collects all results and saves them in a repository (file system) as a map which is being built up as the application is further explored.

Testing assignments include specific objectives. Testers must use their own judgement in following them. If a tester uncovers an area of instability on the route he may explore it or document it for future exploration. The decision has to be made on the spot.

Bug Cluster Identification

Mapped areas of instability define clusters composed of the bugs that have natural affinities of causes. As the test lead collects results from each "chunk" of exploration, two important assessments are done with his peer in development.

1- All new bugs are reviewed to determine if they are indicative of a pattern, are they due to similar or potentially related faults? Is there a correlation between them?
2- Are newly identified bugs potentially related to previously discovered bugs? Is a pattern emerging?

When a suspected cluster is identified the testing team works closely with the development team to find the probable root cause of the problem.

Clusters can be observed in a variety of ways. Here are four examples of "distance metrics" in bug space:

Functionality:	several bugs related to same functionality are discovered
Reliability:	different functions fail in similar way
Efficiency:	several operations are inefficiently using system resources in a similar manner
Time:	several content sources are out of sync

Prioritize based on business impact

Once a bug cluster is identified it must be prioritized. Just like you would prioritize an individual bug, a cluster of bugs must be prioritized. We assigned clusters one of four priorities:

P1 - Fix immediately, next build
P2 - Fix before commercial release
P3 - Fix in some future commercial release
P4 - Do not fix

Certainly the decision to correct the defect had a lot to do with the severity and business importance of the associated cluster. If for example a memory leak is discovered on a server application which causes an occasional failure which results in a very short term delay in service, but without any loss of data, then it may be prudent not to correct it. If however the same memory leak is aggravated by increased concurrent use of the site then it may critical to fix it before going live.

By working on bugs in a collection we can make decisions more efficiently.

Practical results

The techniques developed lead to health communication and team work between developers and testers. Developers and testers enjoyed working together to try and identify probable root causes of sets of seemingly related bugs.

Developers were much more aware of the role of testing in the process. This is especially beneficial as one of the teams goals is also to improve the Personal Software Process (PSP) of individual team members. Empirical knowledge about the processes and people in place can help since weaker points are sometimes inferred from past projects.

Management and team members all bought-in to approach. Combining exploratory testing with detective work makes sense especially when time is tight and we are always working with incomplete data. People were in sync.

We found that we were able to identify an additional 30% more software problems with this approach. More specifically we measured for every 10 bugs reported the correction of 13 defects in the code base. It is impossible to say however whether the correction of all of these defects would have been necessary.

The method is flexible and can use several alternatives as feeding mechanism to the Bug Cluster Identification phase. The one we chose was Exploratory Testing but more detailed and analytical methods may be used. This will be explored in future work.

Our three projects were released on time with minimal field reported defects. All critical problems were corrected during the final integration phase and hundreds of thousands of users are live using this software as we write this article.

We plan to continue using this technique, improving our understanding of the model and generalizing it as we continue. The effectiveness and efficiency, as well as the limits of the method will be further explored as well. This model is used in a commercial and very turbulent e-Commerce environment. It will evolve. All indications are positive and it is presently being used in several projects under development.

Some Web References

www.satisfice.com

James Bach's web site, exploratory and risk based testing

www.testing.com

Brian Marick's web site, articles about exploratory testing

www.amibug.com

Robert Sabourin's web site, various presentations

www.mvm.com - www.myvirtualmodel.com

My Virtual Model community Web site

www.stickyminds.com

Much relevant content.

QW2001 Paper 9W1

Ms. Patricia D. Humphrey
(Neoforma.com)

Quality Assurance and the Internet Site - How To Effectively Hit a Moving Target

Key Points

- Rapid Development and Quality Assurance
- Rapid Testing and Delivery of a Quality Product
- Quality Assurance and it's Ally, Change Control

Presentation Abstract

Quality Assurance, Internet Site Development and On-time delivery into production can be a moving target that is difficult for even the accomplished development team to hit. The major contributor to the strain and hysteria of Internet development is primarily due to the shortened and demanding shelf live that is a mandatory requirement for many funded companies and their participation in the Internet market. Development groups are constantly pressured to deliver more functionality in less time. The rapid development and shortened shelf live is further impacted by the fluctuation of personnel required to accomplish the delivery of the companies complex Internet market needs, therefore propelling the products quality into a significantly condensed or fictional state. Survival of the rapid Internet development cycle can be easily accomplished, therefore producing an effective QA and test process that enables adequate assurances relating to the quality of the products delivered into production.

About the Author

Summary

Over 15 Years of Experience as:

E-Commerce and B2B Quality Assurance Project Manager Director of Quality Assurance Development Project Manager

California State University of Dominguez Hills Computer Science/Mathematics

Current Experience

March 2000 to Present Position: Director of Quality Assurance and Project Manager for eCommerce and B2B Sites Microsoft Corporation and Neoforma.com

Currently I am the Directory of Quality Assurance and Project Manager for the B2B site called Neoforma.com. Neoforma.com, in conjunction with The Microsoft Consulting Division transports me to the San Jose area from Los Angeles, (daily) to control and project manage the quality assurance department relating to the development of the Neoforma.com web development and product delivery schedules.

In addition, my job tasks include the incorporation and management of Microsoft's proposed data scalability and architecture for various clients. I am responsible for orchestrating tests and drive test teams to verify that the data architecture would scale-out in order to support the client's scalability issues and requirements. Cashing sub-systems were verified so as to validate performance and reduction of redundant pages. In addition, security programming guidelines for all application tiers, with a special focus on authentication, storage and transmittal of sensitive information and state management was validated.

Applicable tests were established for many web reporting and analysis sub-projects. These sub-projects were invaluable for laying a comprehensive foundation for the generation of pre-defined and interactive reporting vehicles for the Microsoft clients. These reports included Web Activity (IIS logs), transaction activity and user registration (track as many as 8,000,000 users per minute).

In many cases, it was evident that change and release management procedures were weak, or in many instances, non-existent. One of my major responsibilities was to work with the client's development, test and operations team assisting them with the execution of:

A successful rollout of the software to production
To design and implement efficient procedures (configuration management) for the distribution and installation of changes to the production systems
To ensure that the e-commerce software being changed was traceable, secure and that only correct authorized and tested versions were installed
To communicate and manage expectations of the Client during the planning and rollout of new releases
To agree on the exact content and rollout plan for the release – through Change Management
To ensure that master copies of all software were secured in the Definitive Software Library (DSL) and updated when needed
To establish procedures that would guarantee minimal disruption of the production site Implement roll back planning

During the implementation and test phase of the project, it became obvious that inadequate test methodologies and procedures were being practiced at the e-commerce and b2b sites. It was then my responsibility to perform an audit of the current Quality Assurance Department. My audit would include current practices and procedures along with the identification of system and application development currently being implemented for the site. I was then responsible for the generation of documented recommendations for the Quality Assurance

Department and overall project infrastructure.

September 1998 to Present Position: Project Manager Viacom Entertainment, Inc.

Famous Music Publishing

Responsible for the management of the business and requirement analysis for a document management system that will be implemented by the fall of 2000. I currently manage the business analysis team that is developing the requirements and functional specification documents required for a proposed document management system that will be purchased and installed at Famous Music Publishing Company and Paramount Studios. The project involves the management of personnel who were responsible for defining the requirements for this system with respect to business objectives, application and information objectives and the technology that is currently available on the market to meet the requirements defined. The management of this project has involved numerous meetings that correlate the technology information with the current end-user needs within Famous Music, Paramount Studios and Viacom.

UPN (United Paramount Network)

UPN contacted me regarding the need for a project manager to assist in the requirement and functional specification analysis of a new accounting and financial system. Their current system runs under a DOS based system, which is no longer supported by the vendor. I was responsible for managing the personnel that has been used to evaluate the current workflow of the accounting department along with the marketing departments and human resources. Part of the analysis of the current processes also included new system workflows that show how the new system will need to support unique and proprietary procedures within the entertainment industry. The teams were responsible for end-user analysis, business objectives, application and information objectives and technology analysis. Once the requirements and functional specifications have been defined, than an RFP can be sent out to various software vendors so as to solicit bids for the system purchase.

Paramount Studios

As a consultant, I was responsible for auditing Paramount Studios Y2K test activity. This entailed managing various development groups and the Y2K test activity that took place until after the millennium change. Activities included review of production products for the TV Systems Group, Studio Group, Home Video Group, Financial Group, and Human Resources. Projects ranged from stand-alone system to mainframe applications. I was responsible for verifying that test plans and test cases were appropriate for the products targeted for Y2K test activity along with managing the audit team that reviewed all test results. Verified that test systems were configured to best simulate production systems. Test results were reviewed, documented, evaluated for retest and then archived.

Quality Assurance and the Internet Site - How to Effectively Hit a Moving Target

Patricia D. Humphrey
Neoforma.com
March 30, 2001

Quality Assurance and the Internet Site - How to Effectively Hit a Moving Target

- Topics that will be discussed:
  - Differences between Testing the Internet Site and the Conventional Application
  - Unique Techniques for Management and Staffing the Internet Site's QA Department
  - Control and Configuration Management - The Hidden Ally for Web Site Development and Quality Assurance
  - The Power of Buy-In

Understanding the Differences between Testing the Internet Site and the Conventional Application

- Internet Site Development is much Different than Conventional Application Development
- Not All Web Systems are the Same - Types of Web Systems and Various Configurations
- Testing Differences
- Internet Requirements and Configuration Management

Unique Techniques for Management and Staffing the Internet Site's QA Department

- Manage the QA Test Activity for a Shorter Shelf Life
  - Drive Requirements
    - Identify Key Individuals Who Know the Product
    - Participate in Interview of Key Individuals
    - Establish Preliminary Requirements
    - Preliminary Test Cases
    - Review Meetings
    - Establish Change Control
    - Drill Down for Physical and Logical Characteristics
    - Check physical/logical designs into change control
    - Create separate, non-functionality requirements documents for system interfaces (internal and external), back-end processing and hardware dependencies

Unique Techniques for Management and Staffing the Internet Site's QA Department

- Manage the QA Test Activity for a Shorter Shelf Life (continued)
  - Establish Test Strategy, Not Test Plan that Contains:
    - Introduction
    - Purpose of Test Strategy and the Intended Audience
    - Identification of proposed software and hardware to be used
    - Objective of Testing Effort
    - Web Product Overview
    - Related Documents
    - Overall Project Organization and Personnel/Contact Information
    - Dependencies, Deliverables and Risks
    - Testing Priorities and Focus
    - Test Scope and Limitation
    - Test Approach

Unique Techniques for Management and Staffing the Internet Site's QA Department

- Manage the QA Test Activity for a Shorter Shelf Life (continued)
  - Establish Test Strategy, Not Test Plan that Contains (continued):
    - Test Environment
    - Test Data Setup Requirements
    - Database Setup Requirements
    - Defect Tracking
    - Test Automation Tools
    - Test Script/Test Code Maintenance Process and Version control
    - Regression Tests
    - Open Issues
    - Test Schedules
    - Appendix/Glossary of Terms

Unique Techniques for Management and Staffing the Internet Site's QA Department

- Understand and Diversify the Quality Assurance Team
  - Understand the Skill Set in the QA Team
  - Hire a Mix of Individuals that can Perform Various Tasks
    - DBA Test Engineer
    - QA Developer
    - Senior Test Engineers
  - Communication with QA Team is Essential
  - Establish *Obtainable* Test Schedules and Priorities
    - Allow for slippage and development overlaps
    - Develop test plans, test cases and scripts in parallel to requirement activities

Control and Configuration Management - The Hidden Ally for Web Site Development and Quality Assurance

- Establish Rapid Development Process for Control and Accountability During Development
- Understand the Importance of ***Daily*** Building and Regression Testing the Internet Site
- Control, Control and More Control
  - Establishing a Configuration Management Department
  - Checking Code In and Out of CM Libraries
  - Establish Change Control Team Members
  - Frequent Meetings Regarding Releases to QA, Staging and Production
  - Daily Builds with Regression Test Activities
  - Published Meeting Minutes in Order to Communicate Configuration Board Meetings
  - Using Defect Tracking Systems to Monitor Critical Issues Reported Daily
  - Using Tracking Systems to Monitor Requested Changes to Requirements and Functionality

The Power of Buy-In

- Establishing Empowerment for the Internet Site QA Department
- Communicate the Importance of Specialized QA Activities for Web Application Development
- Adapt Antiquated and Conventional QA Methodologies Used on Non-Web Applications for Rapid Development and Short-Shelf Life Projects

Quality Assurance and the Internet Site – How to Effectively Hit a Moving Target

Patricia D. Humphrey
Neoforma.com
March 30, 2001

Introduction

Quality Assurance, Internet Site Development and On-time delivery into production can be a moving target that is difficult for even the accomplished development team to hit. The major contributor to the strain and hysteria of Internet development is primarily due to the shortened and demanding shelf live that is a mandatory requirement for many funded companies and their participation in the Internet market. Development groups are constantly pressured to deliver more functionality in less time. The rapid development and shortened shelf live is further impacted by the fluctuation of personnel required to accomplish the delivery of the companies complex Internet market needs, therefore propelling the products quality into a significantly condensed or fictional state. Survival of the rapid Internet development cycle can be easily accomplished, therefore producing an effective QA and test process that enables adequate assurances relating to the quality of the products delivered into production.

Key factors include a clear understanding relating to the differences between conventional application development and the Internet development activity. The QA department needs to be savvy when adequately staffing the QA department. Rapid development and testing methodologies need to be implemented so as to avoid bureaucratic and time-consuming procedures. QA needs to identify and work with their hidden alliance with Configuration Management since control of daily changes is inevitable and effective and use of Configuration Management procedures can be a powerful tool for product quality.

Understanding the Differences between Testing the Internet Site and the Conventional Application

Internet site development is much different than conventional application development. Internet sites are typically targeted for a specific market with many evolving and moving components. Many times the complexity of the site's development is taken over by the daily pressures to get the site to production faster than the competition. The decisions to move the site through at lightning speeds can leave the QA department with many unresolved and unplanned issues. It's important that the QA department

understands these differences in order to generate a process that will fulfill the web system life cycle.

Web Systems

Web systems can be huge, with millions of pages, many interconnections, and incredibly high hit rates. Users can be connected to the network via a thin client or a fat client. A firewall determines the kind of access, encryption and security levels. Web servers provide much of the application code and can have accelerators for caching dynamic pages in order to improve user access time. The network can be specialized into an Intranet, Extranet or virtual private network (VPN). An Intranet is an internal network behind a firewall that allows only users within the company to access it. An Extant allows outside partners to have access to the Intranet. A VPN is a secure and encrypted connection between two points across the Internet. It acts as an Intranet or Extranet except it uses the public Internet as the networking connection rather than a company's own wiring. This enables, for instance, a company's branch offices to be inexpensively connected via the Internet.

Attached to the network can be other types of networks such as storage area networks (SANs) and portals. SANs are networks that pool resources for centralized data storage. They may include multiple servers working against a centralized data store built with redundant hardware such as RAID (high-volume storage) devices. Portals (such as Yahoo!, AOL) are full-service hubs of e-commerce, mail, online communities, customized news, search engines and directories, all suited to the particular needs of an audience. Portals are evolving into corporate enterprise portals. Such portals, for instance, enhance corporate decision-making by integrating the company's applications, thereby removing barriers that exist between business units.

Other resources that can make up web systems are: Data Base Management Systems; workflow applications used for optimizing business processes, such as Enterprise Resource Planning tools (e.g., SAP, PeopleSoft, Baan); database applications such as OnLine Analytical Processing systems, which allow users to perform *multidimensional* analysis on data via their browsers; document management tools for providing access into shared libraries of documents; imaging systems for optical character recognition of documents; data warehouses containing terabytes of data; multimedia databases for holding archives of music, speech, videos; mainframes which contain approximately 70 percent of legacy data for large companies; data-marts, which are data warehouses with their own unique interpretation of business data to suit certain functional needs of a business unit; and, non-PC devices, such as pagers,

personal digital assistants, WebTV, and smart phones.

Web systems are made up of various combinations of the resources. Each of the resources imply code that can be dynamically added, changed, deleted, accessed, manipulated, along with their relationships and hyperlinks. QA will need to verify the changes that goes into the web system.

Types of Web Systems

It is difficult to classify the types of web systems being built today as there is no universal blueprint for such systems, the design is still an immature art and the systems themselves are evolving fast. In a broad sense, a web system which is visible via its web site, either acts as a provider of information or is an application. But the applications can be of different types. A web system can be categorized as having the properties of one or more of the following classes:

> **Informational:** information sites with read-only usage, commonly called "brochureware" e.g., information presented on a site that gives details about a company and its products. First-generation web systems are this type and are static.
>
> **Delivery system:** download content to user or resource e.g., download upgrades or plugins
>
> **Customized access:** access is via a customized interface or based on user's preferences e.g., my customized view of my Internet Service Provider's home page, or favorite portal
>
> **User-provided information:** user provides content by filling in a form e.g., subscription to a magazine or registering for a company's seminar
>
> **Interactive:** Two-way interaction between sites, users and resources e.g., business-to-business
>
> **File sharing:** remote users collaborate on common files e.g., users coordinate schedules
>
> **Transaction-oriented:** user buys something e.g., buys books or travel tickets
>
> **Service provider:** rentable applications; user rents an application on a per user, per month basis e.g., virus scan program

Database access: user makes queries into a database e.g., supplier looks up catalog of parts

Document access: libraries of online documents are available e.g., view corporate standards

Workflow-oriented: a process has to be followed e.g., order entry automation

Automatic content generator: robots or agents automatically generate content e.g., “bots” scour the Web to bring back specific information such as best price on products.

Given these classes, it becomes obvious that changes to the web system can be essentially created by anyone or any other resource. Complexity of the systems quality continues when companies partner with others in order to provide a stronger development process.

Enterprise Challenges for Web Systems

QA is not a problem for small, static web systems managed by a few developers. It is a problem for medium and large, enterprise systems that involve many developers creating many changes that will have a high hit rate involving high-volume database accesses and updates every minute. For instance, the one company may experience system loads and usage that acquire as much as 20 million hits per day where as many other companies can have millions of pages that are hosted by thousands of Intranet sites and more than 1,000 web servers.

Developing and maintaining such large systems with large volumes of development changes offers many challenges to companies and most importantly, their QA departments. These challenges span technical, people, process, and political issues.

Variant Explosion

Web systems imply a variant explosion problem. Consider that web systems are either created from scratch, are redesigned or merged web systems, or are web-enabled legacy applications (1). In many cases, companies must live with all these systems in parallel. Thus, a company could easily have a nightmarish number of versions of their latest baseline and a variation of users/customers that have specific web requests and/or information direction. The development process for web systems can become more complex when reconciliation of the Development, QA, Staging and Production servers have failed conventional Configuration Management procedures, therefore leaving numerous variations of critical components that are needed for the customer

iterations. Therefore, adding the method of the web sites original creation, and the four development/QA variations, the beginning total is 5.

Complexity continues since each variant must work with at least two different browsers, including the latest three versions of those browsers—and may need to support five different languages for international use. Hence, we have (1 * 5) + (2 * 3 * 5) = 35 potential variants that must be tested for corporate quality. Most companies have different teams working on separate variants without much communication, reuse or change propagation across common code. With the variants, come all the complexity of parallel development support for simultaneous changes and concurrent baselines, along with significant change propagation to selected variants, thereby demanding change set support, more sophisticated change tracking along with help-desk support and much better release planning and change scheduling. The ramifications for the Quality Assurance Department are dramatic.

The changes and pressures for web systems becomes even more complex when you introduce the Marketing and Product groups who drive this process. Marketing and Product groups are out to boost productivity and raise ROI everywhere they can. Yet technology evolves at such warp-speed today that, in spite of developer's best intentions and technological prowess, there are times when new products fall short of their marketing claims. Or worse, their expectations. The IT department is now confronted with how to control marketing's mandatory and sudden changes. Developing techniques on controlling this change gives the IT and Marketing Group an advantage since they can clearly predict the impact that the changes will make.

Testing Differences

Web development is a relatively new industry. Many QA engineers and IT managers employed by Internet Departments or Companies are accustomed to the development of system delivery and integration relating to mainframe and client server applications. Many of their techniques promote time consuming, and bureaucratic testing strategies that are cast for client server applications, with longer shelf lives and not for the swift Internet highway. The rigid and formal procedures learned in precedent projects will not accommodate the immediate and high-speed testing solutions that are mandatory for web development. Today's web systems not only have unique configurations, custom applications and hyper links, they have introduced testable components that are non-existent in mainframe and client server applications such as:

- Web Reporting
- Scalability Forecasting
- Web Browsers
- Internet, Intranet and Extranet Applications

- Firewalls
- Variations in Network Traffic
- Application and Database Servers
- Variety of Operating Systems

Web system Quality Assurance Departments are obligated to understand the importance and unseen alliance with change control and release management. Web Quality Assurance Departments need to provide communication, direction and procedures for controlling thousands of alternations that becomes an essential function for web development. They are required to test for these changes, keep the test process on course during rapid development and keep the project accountable during development without delaying the development or delivery cycle.

Product Requirements and Configuration Management

The web enables the paradigm of new features and change at the speed of thought. The mindset is typically: I have an idea or see a problem and can, or need, to implement or fix it immediately because it *could* reduce our competitive edge or is globally visible, therefore a liability. Further communications leads all parties to believe that ignoring immediate implementation could cause corporate embarrassment or even worse, litigation. Development, Quality Assurance and Operation teams are pressured and led to falsely believe that there may be no time to follow through a normal requirement, development and change life cycle (such as with documented requirements, change requests, Change Control Board meetings, change authorization, development, testing and re-release). Because the change can be done so easily, process is often bypassed. All the benefits of clearly defined requirements and change tracking are lost. Repeatability of any problems in QA now becomes a difficult benefit to achieve. Risks continue since the security of a controlled rollback of a site, if the change fails, is no longer a viable option and will be devastating for many companies. Understanding that web site accountability is a necessity may be to far gone to recapture.

One of the considerable differences with project requirements, configuration management and projects from the past is that many of the senior projects had requirement and design documents along with configuration management procedures. These systems were able to significantly control changes and demonstrate accountability. Unfortunately, extended time and bureaucratic procedures were offset in exchange for longer shelf lives and accountability. Web systems have not been successful at implementing a balance with a rapid development process that would allow for full requirement documentation and change control of their web sites. This becomes a further liability when the development and technical teams who understand the process move on and this knowledge goes with them. Their

sites are then propelled into hysteria and confusion as changes to the site are further delayed.

Unique Techniques for Management and Staffing the Internet Site's QA Department

Managing and staffing the Internet Site can be challenging and frustrating for not only the QA manager but for the entire project team. Since the Internet Site moves at warp speed, the QA test efforts are more prone to budget and time cuts than during Conventional Application Development. It is important and essential that the QA manager understands rapid testing techniques and methods on how to identify the key personnel that he/she needs to hire in order to assure that a quality product is released to a competitive and accountable market.

Manage the QA test activity for a shorter shelf life

A system professional that has managed the development of various software applications understands the value of clearly defined and documented requirements. The difficulty with rapid development is that requirements are to often rushed through and exchanged for hallway conversations and email transmissions that are targeted for a select few. This type of requirement documentation makes it difficult to understand the web development activity since other key individuals are not party to understandings and issues that are communicated to a select few. The project does not have a tangible understanding and the required detail of the web product and the full content of the web system. This process is further compacted with the promotion and enthusiasm of the marketing and product groups that tend to complicate the process with an immediate need to get the site to production as soon as possible.

The project and the QA department are further impacted since QA will have extreme difficulty developing in depth test cases that test the characteristics and logical expectations of the web system.

In order to obtain requirements, QA should drive the project to deliver the requirements. Working with the marketing and product groups will assure that the projects expectations are met and that QA has testable components that can developed by the development group. When assisting with the development of the requirements, QA should verify that the requirement process, at a minimum, contains the following:

1. Identify the key marketing and product individuals who collectively understand the direction of the site and who have the credibility to define the site that the team is building.

2. Interview the key individuals in order to create a set of preliminary requirements.

3. Once the preliminary requirements are established, develop a simple interactive User Interface Prototype.

4. Immediately begin the development of preliminary test cases from the requirements and the prototype.

5. Begin design and test case review meetings with the key marketing individuals in order to identify areas of ambiguous understandings and to solicit their feedback. Continue revising the documented requirements, the simple prototype, and the test cases until all teams agree on the baseline requirements. Remember… try and keep it simple. Many projects are difficult to complete if the 'big picture' is so large that pushing the site to production takes such an excessive amount of time that the organization's profitability is severely compromised.

6. Insist on putting the requirements, prototypes and test cases under change control. This needs to be set as a top priority since controlling changes in web development projects is difficult to manage as the project begins to move forward.

7. Drill down the design of the prototypes in order to define the physical and logical characteristics of the web site. Physical and Logical Design requirements are usually of a technical nature so involvement from development and business analysts is required. Conduct technical design review meeting in-order to further define the systems functionality. Modify test cases in so as to include the detailed system nature and definitions.

8. Check the physical and logical design requirements into change control. Require an approved change control process that is agreed by all team leads. The approved change control process should contain a provision that changes will not be made without approval. Set change meetings that will be called when changes to the requirements are necessary.

9. Create separate, non-functionality requirements documents for system interfaces (internal and external), back-end processing and hardware dependencies. Check into change control and generate test cases for the additional requirements.

If the generation of requirements is not possible or not available, QA should keep an archive of ALL documents that relate to the project. These documents should include emails, memos, high level requirements, prototypes, screen shots, development defect logs, and meeting minutes. If these documents are organized and frequently updated within the archive, QA may be able to expedite the test case process by capturing information from the documents.

Once a baseline for the web requirements has been established and checked into change control, QA should then generate a high-level test strategy. The test strategy is different from the test plan since it identifies QA's testing vision for the project. The test strategy should contain the following:

1. Introduction
   (Provide Organizational information)

2. Purpose of Test Strategy and the intended audience
   (Describe what the document is trying to establish, proposed readers, marketing, development, management and QA)

3. Identification of proposed software and hardware to be used
   (Software, operating system, database and hardware)

4. Objective of Testing Effort
   (Current methodology)

5. Web Product Overview
   (Describe the product under development)

6. Related Documents
   (List Requirements, design documents, test plans such as functional, integration and acceptance test plans)

7. Overall Project Organization and Personnel/Contact Information
   (List key team members and QA personnel)

8. Dependencies, Deliverables and Risks
   (List all QA deliverables and dependencies along with possible risks)

9. Testing Priorities and Focus
   (Describe how tests will be prioritized and where tests will have heaviest focus)

10. Test Scope and Limitation
    (List the possible risks, test case generation/maintenance, white box test activity, report generation and the sharing of information)

11. Test Approach

(Describe the testing approach, how QA will assist at stabilization of code, regression tests, etc.)

12. Test Environment
    (Describe and list hardware, operating systems, other required software, data base, relationship to development, staging and production environment, configuration management, build process and delivery to QA)

13. Test Data Setup Requirements
    (Describe how test data setup will be defined in test cases)

14. Database Setup Requirements
    (Describe database setup requirements)

15. Defect Tracking
    (Describe how defects will be tracked, system to be used, reports generated and frequency)

16. Test Automation Tools
    (Describe automation tools to be used, maintenance process, etc.)

17. Test Script/Test Code Maintenance Process and Version Control
    (Describe how test scripts will be archived, method of change and version control, etc.)

18. Regression Tests
    (Describe regression test strategy, frequency of regression test execution, regression test pass/fail acceptance criteria)

19. Open Issues
    (List any open issues not addressed in test strategy)

20. Test Schedule
    (Define test schedule)

21. Appendix
    (Glossary of Terms, acronyms, etc.)

Understand and Diversify the Quality Assurance Team

Its important that the individual managing the QA team understands the skill set of each of its participants. Many times, QA personnel are hired as entry-level Information Technology personnel. There is nothing wrong with hiring entry level test engineers yet in rapid development environments, they may quickly become lost in the hurried pace of the project's expansion and may have difficulty providing useful contributions due to lack of experience. Their lack of experience can be costly to the organization, not only in miscommunication but also in lost test execution and unreported system

failures. Therefore, the QA team should be diversified with a variety of technical skills.

Skill sets in the QA team needs to be as broad as the requirement and development activity. Rapid development activities necessitate the QA team to be astute, therefore requiring knowledge in areas of requirement analysis and development.

In addition to traditional QA procedures (test plans, test case, test scripts and test execution) web development QA departments need a mix of individuals that can perform requirement analysis, unit and white box test activities, database and hardware configurations along with technical document preparation. This diversity allows the QA department to keep pace with the development process and helps eliminate any dependencies on other groups within the organization. When staffing the QA department, the manager should diversify with the following QA personnel.

DBA Test Engineer – DBA(s) in a QA environment are skillful at working with the marketing and product groups in order to drill down the needed requirement specifications for the QA department. Their participation in this area assists with the much-needed test plans and test cases that will be utilized by the QA test engineers. Having the QA DBA generate the high-level test plans and cases before development starts, avoids the project from defining ambiguous system functionality that may be inappropriately developed and changed at a later date in the project.

QA Developer – Stabilizing the project's components at the earliest stage is essential for rapid development and web systems. Testing the components at their infant state provides a swift method for stabilization. QA should employ developers who will work in the shadows of the development group. The QA developer will assist development with unit and white box test activities. This type of test activity assures that components are sound and stable as they are integrated with other project components. Identifying issues and defects at the development level significantly reduces the development effort as the system begins configuration with other segments within the project.

Senior Test Engineers – The QA Department should employ senior test engineers with extensive web system test experience. These individuals are accustomed to web system methodology and have experience with the pressures of rapid system development.

Communication with QA Team is Essential

Meeting with your QA teams on a regular basis assures that critical issues are reported at an early state. On many occasions, communications relating to early issue detection and possible problems are assumed within QA and rarely re-directed to the appropriate managers until the issue has escalated to a critical level. Frequent communications and project status reporting assists all participants of the project to immediately redirect the issue before it becomes a showstopper. Many times, QA engineers will make assumptions that others have identified and communicated problems when in fact they have not. QA personnel should generate a 'brief' daily status of the project/component that they are testing. These daily reports should be rolled up to a weekly report that can be reviewed by the QA manager.

Establish Obtainable Test Schedules and Priorities

It's always fascinating when project and development managers forecast QA schedules and time estimates, especially when QA was not consulted. Project schedules will display test time expectations that are, in most cases, ambitious and unrealistic. Therefore, its the responsibility of the QA department to immediately review the project schedules in order to rapidly communicate with the other project leads that the allocated test schedules are or are not obtainable. If the revisions in the test schedules are not communicated, then they are usually permanent and difficult to change. This problem becomes further impacted when development schedules begin to slip and QA test timetables are further compressed.

Since schedule slips inevitably require overlaps in QA's time slots, the QA manager will need to carefully forecast and factor the time needed for:

Test Plan and Test Case Generation
Test Execution (White/Black Box, Integration, Interface, System, etc.)
Marketing/Product Acceptance Test Activity
Customer Service Test and Training

Control and Configuration Management – The Hidden Ally for Web Site Development and Quality Assurance

A lost concept during the rapid development process of many Internet Sites, is the omitted technique of establishing control and accountability during development. Even some of the largest, most successful companies have Intranets and External Internet sites that are, for all intents and purposes, out of control. Propelled by competitive and customer pressures, many Web Development companies may have unintentionally rushed to build their Internet site without fully documenting

or controlling their build process, therefore loosing sight of how it actually works.

These Internet sites then expand as new features are required, and over time, become mazes of directories, applications, and scripts that the site may or may not still use. The Internet site becomes cluttered with the debris of past system iterations. The team that built it may move on, and no one really knows how the entire site works.

Since many sites are considered to be a full-featured Web application, it commands that its employees keep a pace of change that increases exponentially. Many sites, along with their QA department do not realize that a hidden ally is simply controlling and implementing a Configuration Management procedures that specializes in keeping the build process in order and recoverable. This configuration management implementation is a critical supporter for the QA department. Proper configuration management assures that accountability regarding the sites design and development teams are consistent with the constant need to integrate to the site scores of technology changes that incorporates hundreds of new releases each year, with proper change control or system documentation.

The Importance of Building and Regression Testing the Internet Site Daily

Many Internet sites do not understand the value of building their site on a daily basis and performing regression test activities. The daily build and regression test activity rapidly provides stability for the individual components as they come together in integration. In the daily build and regression test, the web site is built every day. The software is then put through two types of regression tests; one to test overall functionality and the other to verify that all closed defects have not been re-opened. These tests should be automated and are relatively simple to execute in order to see how stable the site is. The daily build and regression test significantly reduces the likelihood of one of the greatest risks that a web project faces: the risk that when the different components combine or integrate the code that they have been developing separately, the code doesn't work well together. This practice also addresses the risk of low quality. By at least minimally testing the full scope of the web site every day, quality problems are reduced from taking control of the project. The project team brings the site to a known good state and then keeps it there. The site is simply not allowed to deteriorate to the point where time-consuming quality problems can occur.

Performing daily builds also makes it easier to monitor a site's progress. When the project team builds the system every day, the status of both complete and incomplete features is visible; both technical and non-

technical parties can simply exercise the site to get a sense of how close it is to completion. In addition, the marketing and product groups can see progress without waiting until completion to identify key issues.

The daily build and regression test practice is especially important on large and complex sites since the risk of unsuccessful interface integration is so significant.

Control, Control and More Control

It's important that web development projects maintain control over the project. This control can be easily accomplished with standard configuration management procedures. These procedure include basics such as:

1. Establishing a Configuration Management Department
2. Checking Code in and out of CM Libraries
3. Establish Change Control Team Members
4. Frequent Meetings Regarding Releases to QA, Staging and Production
5. Daily Builds with Regression Test Activities
6. Published Meeting Minutes in Order to Communicate Configuration Board Meetings
7. Using Defect Tracking System to Monitor Critical Issues Reported Daily
8. Using Tracking System to Monitor Requested Changes to Requirements and Functionality

The Power of Buy-In –

Without Buy-In from the other team members, QA and the web project will be powerless. Confusion, breakage, instability and dissatisfaction will surely prevail if the minority groups are allowed to circumvent project understandings and goals relating to the delivery of a quality project. Convincing arguments from upper management and QA need to demonstrate the importance of specialized QA activities for web application development. Antiquated and conventional quality assurance methodologies used on non-web applications need to be adapted for rapid development and short-shelf life projects. Buy-in and acceptance of new QA techniques and processes that allow for rapid test activity need to be communicated to all parties of the web development process and quality assurances to succeed. Exclusion of Buy-in and changes in QA test practices will have extreme difficulty demonstrating to the designated user that the web site released to production is stable and reliable.

Reference:

Lyon, David. *Practical CM Best Configuration Management Practices*. MA: Butterworth-Heinemann, 1999

McConnel, Steve. *Rapid Development.* Redmond, Washington: Microsoft Press

Carroll, Paul B. *Creating New Software Was Agonizing Task for Mitch Kapor Firm.* The Wall Street Journal, May 11, 1990

Gibbs, W. Wayt. *Software's Chronic Crisis, Scientific American.* September 1994, 86-95

Jones, Capers. *Assessment and Control of SoftwareRisks.* Englewood Cliffs, N.J.: Yourdon Press, 1994

McConnel, Steve. *Code Complete.* Redmond, Washington: Microsoft Press, 1993.

The Patricia Seybold Group, *Managing the Mutable Web Site: Orderly Web Site Progression,* © 1999

Forrester Research, Business Trade & Technology Strategic Report, *Resizing On-Line Business Trade,* November 1, 1998

Humphrey, Patricia. *Testing and Controlling Changes to Web Development,* © 1999, Englewood Cliffs, N.J.: Yourdon Press, 1999, (submitted for publication)

International Data Corporation, 1999.

Murugasen, S., Deshpande, Y.: *Proceedings of ICSE99 Workshop on Web Engineering*. International Conference on Software Engineering, Los Angeles, USA (May 1999)

Merrill Lynch Co., 1999.

Bloomberg News: Net Shares Battered Amid Signals That Web's Expansion Is Slowing. *Wall Street Journal* (June 15, 1999)

Dart, S: The Dawn of Document Management. *Application Development Trends* (Aug. 1997)

Hutcheson, M.: The NT Application That Wouldn't Die (NASDAQ.COM). *Enterprise Development*. 1,1 (Dec. 1998)

Sliwa, C: Maverick Intranets: A Challenge for IT. *Computerworld* (March 15, 1999)

Dart, S.: To Change Or Not To Change. *Application Development Trends* (June 1997)

Gellerson, H. Gaedke, M.: Object-oriented Web Application Development. *IEEE Internet Computing* (Jan/Feb 1999) 60-68

Lockwood, L.: Taming Web Development. *Software Development Magazine* (April 1999)

Iyengar et al.: Techniques for Designing High-Performance Web Sites. *IBM Research* (March 1999) 17pp

Powell, T.: *Web Site Engineering*. Prentice Hall, NJ, (1998)

Dart, S.: The Agony and Ecstasy of CM. A half-day tutorial given at 8th International Workshop on Software CM, Brussels Belgium (July 20-21 1998)

Dart, S.: Not All Tools are Created Equal. *Application Development Trends* (Oct. 1996)

Dart, S.: Content Change Management: Problems for Web Systems (Aug. 1999)

Siegel, D: *Secrets of Successful Web Sites : Project Management on the World Wide Web*. Haydn Books, Indianapolis, Ind. (1997)

QW2001 Standby Paper 9W2

Mr. Alexey Kerov
(Amphora Quality Technologies)

Iterative Approach as Basis For Effective Testing

Key Points

- Iterative Approach to the development of software as an alternative to the method of “waterfall”
- Iterative Approach advantages and disadvantages from the testing point of view
- Ways of optimal use of Iterative Approach to testing
- Testing Outsourcing

Presentation Abstract

The advance of Internet economy requires employment of more economical and at the same time more effective software quality assurance methods. Typical WEB application, portal, e-shop require 100% reliability of server functioning 24 hours 7 days a week and the developer is obliged to react immediately to market demand, to modernize site at a high pace, risking to disturb the application functioning, to introduce a new security hole or to make the performance unacceptable for the end user.

The iterative approach to the software development and in particular to testing gives the opportunity to reduce the risk of arising such problems. In terms of time shortage and lack of human resources iterative approach allows dividing of whole process to several parallel tasks, to manage the development effectively and accurately, to detect program defects and design failures on time as a result increasing the product quality and reducing production costs.

At the present time many companies are just starting the trial use of iterative methodologies that is why it is very important to realize the fact that besides advantages such an approach has also a series of problems that have to be studied necessarily. In general the questions of qualitative personnel management and of development process control have a decisive role in the implementation and employment of the described approach. The question of particular interest is interoperability between an organization which uses the iterative approach and a subcontractor that means companies performing outsourcing, for example in the field of testing. There is no doubt that the experience of Amphora Quality Technologies in the area of implementation and employment of iterative testing will be very useful for all the companies which are engaged in optimization of functioning of its testing and quality assurance departments.

About the Author

Alexey Kerov is a Marketing director of Amphora Quality Technologies, the very first Russian private company that has become leading Russian SQA services provider since last year. Within 10 years he made career from a finance programmer to Top Management positions at line of the leading Russian Software Development Companies. On the one hand, Alexey has profound IT experience and is well conversant with the problems of this branch. On the other hand, in order to provide professional solutions to business tasks, he received a good education on management and led the promotion of the services of the first-string Russian development companies in international market.

Few years ago Alexey faced the problem of quality and felt an interest in tasks of organizing of the software engineering process and Quality Assurance methods. Although he was working intensely at marketing, Alexey managed to become familiar with several cutting-edge QA methodologies and tools and started his work on problems of QA industry. He has a number of publications devoted to software development process. Perceptions of necessity of profound approach to testing and quality assurance led him to the joining the AQT Company.

Alexey’s high weight among software development community has allowed him to make AQT known as a reliable partner not only in Russia but also in the USA, UK and Europe.

Iterative Approach as Basis for Effective Testing

Alexey Kerov
Marketing Director

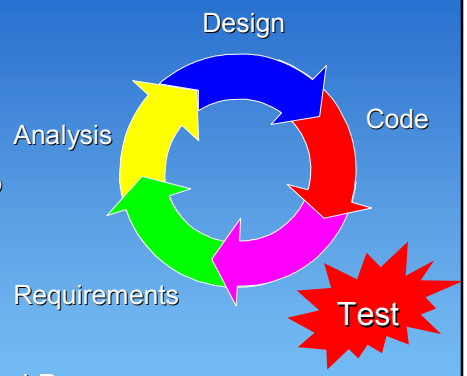
amphora Quality Technologies

Why Speak About Iterations?

- Waterfall process does not meet "The Internet time" needs
- We are constantly searching for new progressive technique of software development and testing
- Understanding Software Lifecycle – key point to success
- All the modern Software engineering processes have "cycles" as their basis

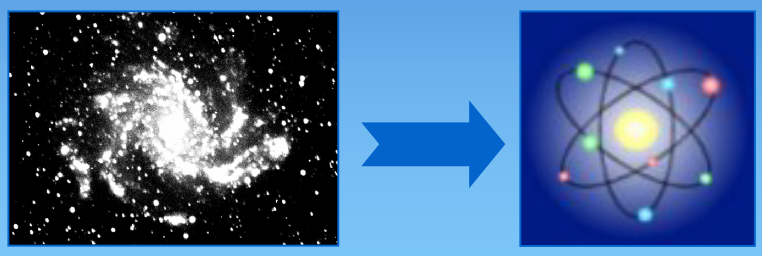
Software Lifecycles and Models

- NEN
- TIM
- TMM
- TSM
- IEEE
- AQAP
- TOM
- ESPITI
- PSP
- SEI/CMM
- ISO 9000
- TQM
- BOOTSTRAP
- TickIT
- SPIDER
- SPICE
- Rational Unified Process
- Many more – name yourself…



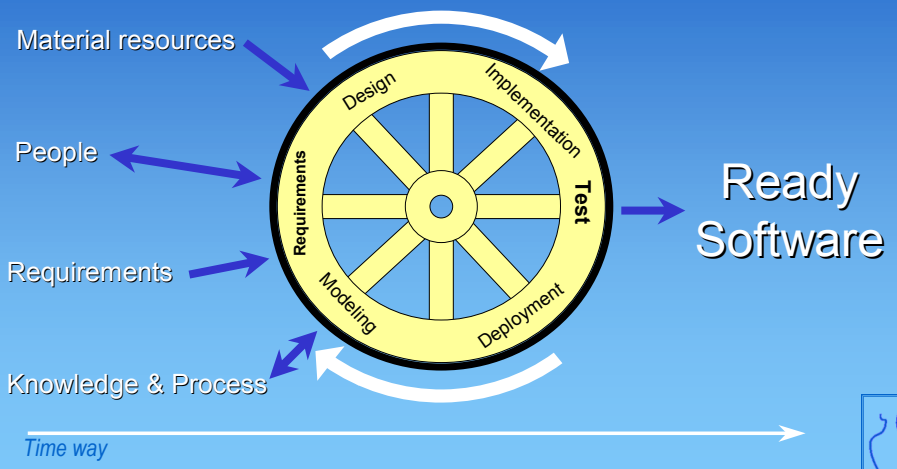
We Need More Common Knowledge

- Understanding of “hidden” process
- Finding most common set of software lifecycle phases
- In fact all processes have “cycle” or “spiral” nature



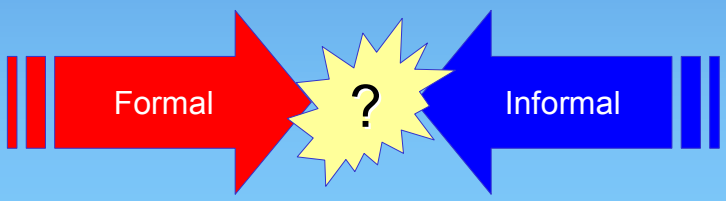
Galaxy Atom

The Wheel of Software Process



Contradiction

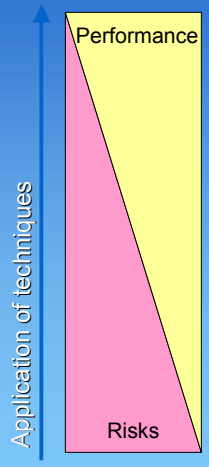
- Formal methods good for long term development
- The Internet Time projects mostly use informal methods
- We need time optimized and quickly adopted technique



How To Solve This Problem?

- Achieve equal understanding of development life cycle among all involved persons
- Identify most time consuming phases
- Outline ways of life cycle optimization
- Design accurate assess criteria

Ways Of Optimization



- Requirements management
- Risk based management
- Universal language – UML
- Best practices dissemination
- Use or study as less as one formal methodology or standard
- Constant improvement of process

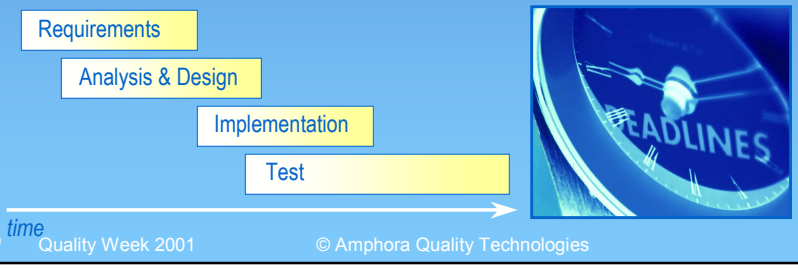
Iterative Approach to Software Development

- Early beginning of Test process
- Reduction of development risks
- Lower pressure on QA departments
- Effective requirements management

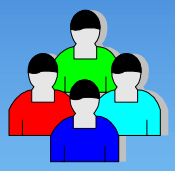
An *"iteration"* is a sequence of activities with an established plan and evaluation criteria, resulting in an executable release

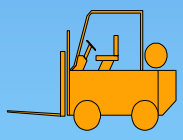
Classical Iterative Testing Approach Risks

- A lot of iterations may lead to management problem
- Long testing phase in each iteration slows down development cycle
- Reduced quality of tests due to short time of iterations



Iteration Specific Optimization

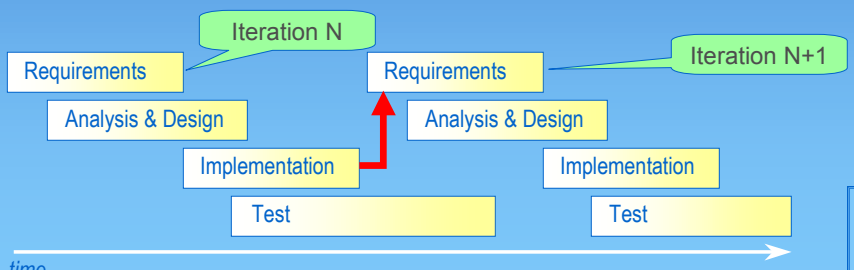




- Advanced test planning
  - Advance beginning of iterations
  - Test phases synchronization
- Flexible management
  - All-knowing test manager
  - Mix testers with developers
  - Test outsourcing
- Optimized automation
  - Test execution automation
  - Requirements tracking system
  - Self-testing procedures

Advance Beginning of Iterations

- Next iteration starts before finishing of all tests
- Testing in next iteration starts after completion of all tests in previous iteration
- Two releases can be made in parallel
- Testers not under severe pressure
- Significantly reduced time to market

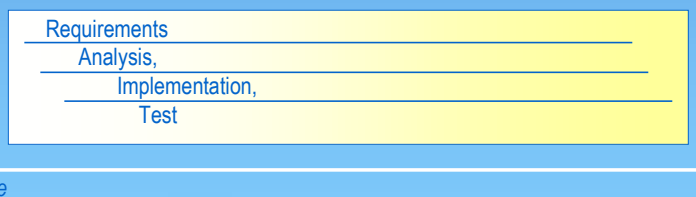


Challenges of "Advance beginning of Iterations" Approach

- Insufficient resources can slow down process
- Having more than 2 concurrent iterations is hard to manage
- Doubling of resources gives less than 70% increase in performance
- Change of requirements can possibly bring to naught the effect of concurrently executed works
- Old-fashioned automation systems do not support "overlapped" approach

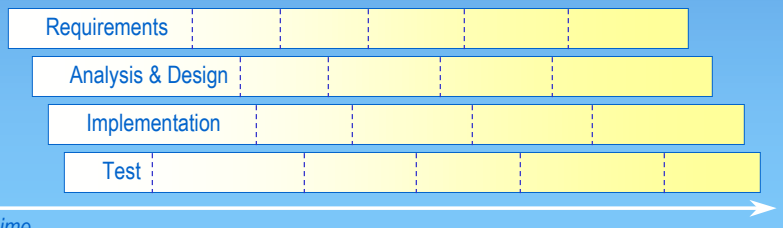
Internet-Speed Project

- Informal process
- Mixed process phases
- Weak quality management
- High time pressure
- Absence of Requirements



Internet Time Management

- Test manager should be aware of all development
- Mix testers with Web designers and coders
- Sub-contract and outsource effectively
- Make "Good enough" software
- Preserve "dynamic balance" of the production system



SQA Consulting & Outsourcing Address Proposed Approach

- SQA Consulting
  - Independent risk assessment
  - Third party analysis of software development process in your organization
  - SQA Consultants use modern and improved software engineering methodologies

- Test outsourcing
  - New look at your software
  - Extensive experience of testers
  - Modern tools of test automation
  - Flexible schedule and scalable resources

Amphora Quality Technologies

- Advantages
  - Cutting-edge methodologies and tools
  - Fundamental background, Extensive experience
  - Flexible schedule, Scalable resources
  - Challenging application testing
  - Customized solutions
- Laboratories
  - Internet applications lab
  - Functionality lab
  - Performance lab
  - Research department

AQT Expertise in Outsourcing

- Web site content and Load testing
- N-tier client-server solutions, Middleware servers
- Biometric scanners and security software
- Front end C++ compiler compliance to standard
- And many others…

"We devoted a great deal of attention to the problem of product quality in respect of Informix Gateway to the Future, but we decided to entrust final testing to the professionals at Amphora Quality Technologies – they performed some serious work and gave a new impetus to this migration technology." declared Valerii Dutchak, manager of the professional services division, Informix Russia.

Let's Meet Again!

We would be glad to see you at our booth or Contact us:

Russian Federation

Address: Office 701, 17, Presnensky val, Moscow, 123557, Russia
Phone: +7 (095) 737-0225, +7 (095) 784-7496
Fax: +7 (095) 737-0224

E-Mail: *aqt@in-amphora.com*

WWW: *www.in-amphora.com/aqt*

Iterative Approach as Basis for Effective Testing

Alexey Kerov
Marketing Director
Amphora Quality Technologies

Abstract

The advance of Internet economy requires employment of more economical and at the same time more effective software quality assurance methods. Typical WEB application, portal, e-shop require 100% reliability of server functioning 24 hours 7 days a week and the developer is obliged to react immediately to market demand, to modernize site at a high pace, risking to disturb the application functioning, to introduce a new security hole or to make the performance unacceptable for the end user.

The iterative approach to the software development and in particular to testing gives the opportunity to reduce the risk of arising such problems. In terms of time shortage and lack of human resources iterative approach allows dividing of whole process to several parallel tasks, to manage the development effectively and accurately, to detect program defects and design failures on time as a result increasing the product quality and reducing production costs.

At the present time many companies are just starting the trial use of iterative methodologies that is why it is very important to realize the fact that besides advantages such an approach has also a series of problems that have to be studied necessarily. In general the questions of qualitative personnel management and of development process control have a decisive role in the implementation and employment of the described approach. The question of particular interest is interoperability between an organization which uses the iterative approach and a subcontractor that means companies performing outsourcing, for example in the field of testing.

Software lifecycle

The present report is devoted to Iterative Testing Optimization. But preparatory to considering the given subject it is necessary to give definition for the iterative testing and iteration in general. Why is it necessary to speak namely about iterative testing?

Considering growth of software industry during the last decade we can state with confidence that it was in the name of Internet. Namely Internet technologies had a great deal to do with development processes and, in particular, with quality assurance and software testing. A classic approach to software development, so-called "waterfall" method is well known from manuals. In brief its essence is in consideration of software development process as a linear sequence of actions from intention to develop a program to testing and implementation of a new program product. This concept of development process has been formed rather long ago, probably at the age of first lamp computing systems. But with the advent of the Internet economy age we begin to feel that this approach is defective and incomplete. There is a gap between theory of software development and practice of development of Internet systems and software in general. Most likely at fault is development time shortage, radical complication of software systems, availability of component, object programming, creation of multilevel architecture of software complexes.

Software processes

As a result we see a sharp rise of interest to problems of software development methodologies and, in particular, to quality assurance as one of the most important features of a program product. It looks like each developed country of the world has its own institute for software engineering and software quality assurance. The last decade was the most fruitful in creation of different software technologies and standards. International quality standards of series ISO 9000, 15504, developed by Software Engineering Institute – CMM, PSP are among the most well-known. But these recognized methods are not the only available. In actual practice of present time in spite of availability of international standards and authorities in the field of software quality there appears a lot of new technologies, processes, ways of assessment and software quality assurance. Many large companies and state organizations in spite of availability of standards take the road of development of their own technologies adapted to local conditions. The companies engaged in software testing and quality assurance also do not stand aside and not always are guided by standards creating again their own software technologies. By now there are known at least twenty such processes which came in "great life" from organizations, where they have been developed and got their own famous names and abbreviations. It looks like the given process is connected with both difficulty of the problem to be solved and absence to a great extent of fundamental knowledge of software engineering nature. As a result in real life we can use only adapted technologies, so called tailor-made quality, which as a rule reflects only subjective opinion about quality assurance process of one specialist or a group of specialists. In their turn available standards as a rule provide only a possibility to carry out standardized assessment of quality assurance system, process maturity level, but to a lesser extent they describe its essence, which to all appearances forms the basis for creation of few tens of software technologies and quality assurance processes.

Turning back to necessity of fundamental knowledge I would like to mention one feature typical for many technologies. This is availability of both evident and hidden cyclicity in software development process. Since revealing the most common features of processes permits us to get the most complete and authentic knowledge, as well as to reveal the laws of outgrowth, it is appropriate to consider this fact in more detail. As we can see software development process consists of several stages or phases familiar for us from the "waterfall" process, namely, business modeling, requirements specification, requirements analysis and system architecture design, development and coding, testing and implementation. Different terminology and graphical presentation of the development process are used in different technologies, which for some specialists makes understanding easy, but for other specialists difficult. In our opinion it is important to reveal the essence of these processes, which reduces to the fact that software development should not be considered as a linear process consisting of 3 or 10 stages. In fact it is of cyclic nature. Our experience in different software projects tells the same.

Nature of cycles

In the course of work we practically always should turn back to earlier stages of the process. It is generally known that testing is a cyclic process. Error searching – coding – debugging and so on till reaching the required product quality level. If there appear serious and conceptual errors, it is necessary to enlarge the given cycle and to turn back to stages of design and development, and in the worst case to revision of task. So it is apparent that several cycles of iterations are necessary to produce a ready software version. They can be both complete iterations including all the considered development stages, and local iterations, for instance: testing – error correction. In the limiting case even work of a programmer can be considered as an iterative process because on receiving a requirement description he, as a rule, does not write the whole program for its

further testing. He moves forward gradually, i.e. develops a code fragment, carries out its testing and debugging, then increases functionality, again carries out its testing and repeats this several times and only then passes the code for external testing.

It is evident that this way of work is in full correspondence with the ideas of dialectical materialism indicating progressive and spiral development of nature. The given law is universal in nature: from development of the Galaxy to atom structure. It is not surprising, that software development process obeys the same principles, but at different organizational level. The given statement permits us in a new fashion to look at organizational problems of software companies and software quality assurance. What's more the developed in time cyclic process can be considered as harmonic oscillations. So in case of sufficiently formal description of software development process by means of a number of harmonic functions it will be possible to use all the accumulated body of mathematics for more detail analysis of software development process, search for optimum management.

Development process as open system

So an understanding of cyclic nature of software development process provides a fundamental basis for the further studies in optimization of functioning of software companies. Now let's consider a software development cycle in general and identify external factors influencing the process of software development. It is worth noting that due to difficulty of the problem considered, there is a good reason to consider software development process as a dynamically stable open system using the corresponding methods of system analysis. As discussed above the general cycle consists of the phases familiar to us from the "waterfall" method. Among the external factors defining functioning of development process special attention should be paid to the following: availability and accessibility of material and human resources as the main resources of a software project; knowledge, experience and techniques accumulated and used in the given organization. Customer requirements for development of a certain software are applied to the system input. At the system output we obtain a new software, gained experience and knowledge. Amount and structure of personnel also can be changed in the course of development. Systematization of acquired knowledge expressed in form of the best practices to be used in realization of the following software projects is of special importance.

Formal vs. Informal methods

From history of science and technology it is well known that knowledge development was the most stimulated when physical or mathematical model describing some process came into conflict with practical experience. The efforts to solve the problem resulted in creation of new hypotheses, theories, discoveries enriching our knowledge of nature and ourselves. At present similar situation is in the field of software development. The technologies created many years ago, for instance the "waterfall" method, that looked stable and reliable before, now begin to loose their stand. Technologies which were the best solution 20 years ago now are no good at all. Modern Internet projects can serve as an example. Extremely short time for development of program products dictates priorities in the development process. Old techniques can't any more answer the challenges of time. In practice they are simply ignored. Formalism is replaced by absolutely informal relations, experiments in development process. Success of one or another software project as never before depends on personal qualities of managers and programmers. Management risks became higher. It looks like crisis of management is one of causes of present recession in high-tech industry. So it is high time to consider intently the problems appeared. At the very first sight it is evident that now shortage of time is the main problem for specialists engaged in software development. That is why optimization of development process should be directed to its modernization aimed at minimization of development cycle time.

Approach to problem solving

What steps are to be made in this way? In our opinion first of all it is necessary to make sure that the company staff has a clear and well-defined understanding of problems, goals and tasks to be solved. Until this problem is not solved, all further actions will either not bring the expected result at all, or will be a cause of one more disappointment. Namely at this stage an understanding of fundamental cyclicity and iterative character of processes can be of immeasurable service.

When the first goal is accomplished, you can go to consideration and description (!) of the process accepted in your organization. The given stage will make your mutual understanding even deeper and will permit you to reveal the most difficult stages of work requiring maximum time and other resources. Then you can start developing methods and approaches to solution of the available problems. These can be both fundamental studies and "trying on" the best practices or taking advice of a consulting company.

Finally, when determining ways of your development process modernization don't fail to bear in mind assessment of the results obtained. It is necessary to clearly determine the goals and criteria of success. Only in this case your material, moral and physical efforts will be not for nothing.

Ways of lifecycle optimization

In practice even without complicated mathematical and statistical methods for analysis of work of organizations there are a lot of well known approaches to improvement of development process that in most cases are simply ignored consciously or unconsciously.

No matter how short are time limits of your project, time for assessment of risks is always available. This probably is one of the most efficient methods to obtain the best product quality with minimum resources, first of all time resources. As a result software quality will be not absolute, but "rather good". Just this quality in most cases is expected by the Customer. Moreover, in most cases assessment of risks makes it possible to do work within time and budget limits. This concerns both the development process as a whole, and testing process in particular. May be this is especially true for testing because of lack of time for testing in "short-term" projects, and extremely high responsibility of testers.

Requirements management is no less important method supplementing risks management. What information can be in practice a basis for conclusion about risk of one or another way? Only documented and clearly stated requirements to the system. They can be in any form, but electronic document is preferable. A number of software systems have been developed for this purpose. Important conditions for requirements are clear interpretation, completeness and consistency. In real life it is difficult to satisfy all the above conditions. But the more accurate is system description, the higher are chances for success! It is not recommended to go deep in details missing some important requirement of a higher level. In description of requirements it is also recommended to apply risks assessment.

There is no question that the language is the basis for mutual understanding of people. Even using the same terminology specialists can imply different meanings in it. This is connected with large amount of methodologies with different meanings of the same words. As a result it looks like the members of one team speak different languages. Coming to common point of view and working out any solution turns into long and painful process. So it is not worth to ignore training of all team members in any one software technology. As a rule such technologies have a glossary whose terminology is equally understandable to all team members. In addition to terminology the important element of communication consists in exchange of imagery and graphical

information such as charts, diagrams. Adopt one of graphical notations and use it. In our opinion the most promising now is the Unified Modeling Language. Our experience indicates that observing so evident rules saves you a lot of time and efforts.

System growth is the guarantee of its long and successful functioning. Termination of growth means unavoidable death of the system. You should find possibilities and resources for growth of your development process. If there is no sufficiently experienced person in the team, who can analyze the current status, errors, successful solutions for introduction of this knowledge at the further stages of work, take advice of consultants, carry out seminars, training. Collect the best practices and inform of them all the interested team members. A tester or a programmer working under pressure of time has no possibilities for investigations, so inform him of a new method of memory overflow errors by e-mail, or organize a short lecture.

Iterative approach

Let's consider one of special techniques intended for iterative testing optimization. It looks like namely iterative approach will permit us to eliminate the conflict appeared in software development and provide maximum possible use of formal technologies in modern software projects. Let me give a definition of the term “iteration” to clarify its backbone and create a good background to understanding how iterations can be optimized. Webster’s desk dictionary of the English language provides the following definition of the word “iteration”: «a procedure in which repetition of a sequence of operations yields results successively closer to a desired result».

When the iterative approach is applied to software engineering a team of developers is supposed to perform a sequence of iterations when developing software. Each iteration, in this case, is viewed as a compact complete cycle of software development containing stages of Modeling, Requirements Specification, Analysis, Design, Coding and Testing. At the early stages of a project most of the time of a cycle is spent on Modeling, Analysis and Design, later on Coding and finally on Testing.

It should be noted that the iterative approach implies creating of testing scenarios at the early stages of system development on the basis of existing Technical Requirements or Use Cases. Now the first prototype of future program developed within one of the first iterations involving coding appears together with the first tests that helps to find conceptual defects at early stages. Subsequently, further evolution of the tests goes simultaneously with the development of whole system. Every new iteration implies regression testing and also design of tests for newly created or modified functionality. The iterative approach is exceptionally good for the quality of a released product because it allows you to control the quality of the product and its development process itself from the very beginning till its commercial release is delivered to the customer.

Such an approach is employed in modern methodologies of software engineering such as Rational Unified Process, Microsoft Solution Frame and many others.

This is the way we will understand the meaning of the term “iteration” to avoid any ambiguity. And now let’s see what drawbacks we may face when using classical iterative approach.

Iterative approach risks

The iterative approach provides really fruitful results when iterations are short and rather numerous. Short duration of iterations allows team to control project execution more frequently and make amends more operatively. As we can judge from our own experience and appraisals of many independent experts you should estimate 5-6 iterations as a minimum to achieve more or

less significant effect and it may take you more than 10 iterations to enjoy all the advantages of the iterative approach. And only in this case required efficiency of change management can be achieved.

But in fact, in real life it's not that easy. An increased amount of iterations requires sound management that may cause a problem itself because of the personnel inexperience or lack of automation tools. Moreover, you will face some additional difficulties when considering the organization of testing in detail.

When the iterative approach is applied to software development the team is supposed to execute a sequence of iterations. Each iteration implies Quality Assurance activities including Testing.

As a rule, testing begins since the very start of iteration and lasts till the iteration is complete. The testing goes simultaneously with analysis, design and development of the system. Most of the time of this process is spent on planning, development of the test strategy and method of testing, design and implementation of the testing scenarios. Execution of the tests itself and analysis of their results usually take place at the final stage of an iteration falling behind the other works. If you still have time on schedule, you can eliminate all the bugs and defects of design and architecture within the current iteration delaying its completion, but if you don't, you have to postpone it for the next one.

On the one hand this approach pays its way because it allows developers to conduct the processes of development and quality assurance simultaneously providing quality control for each release of the product apiece and also allows you to discover some amount of bugs and eliminate them within the current iteration before the coding is complete. On the other hand efficient full-range testing as an inevitable part of the sequence of iterations may cause considerable delays of product releases. This problem becomes extremely important when a project has a very tight schedule of releases and lacks resources.

The point is that you can't finish the iteration till the full-range testing of the current release is complete; hence you are unable to start the next iteration. Of course it may look reasonable that the development shouldn't be continued till you are absolutely sure that the achieved results fully meet your requirements. However as our practice shows this approach works well in some ideal conditions, which in majority of real projects cannot be provided.

We always find ourselves under an extreme pressure of time and other recourses shortage so we either have to delay the deadline of release, or reduce the full-range testing and its quality, or leave some bugs unfixed and so on. In any case you see project risks increase, software quality downgrade, time shortage and problems mounting. It looks like there's no way out.

However it's not true. The truth is that the classical iterative approach of software engineering needs to and can be successfully upgraded. It is necessary to extract testing from the general sequence of iterations and organize it in a special way. Testing is a very serious and complicated procedure. It's as difficult as it is in the software development itself to forecast at the beginning what difficulties you may face and how much time it'll take you to discover and locate all possible bugs. That's why we should eliminate any correlation between the deadline of the full-range testing of the current release and further continuation of works on the project as a whole.

What steps should be taken to achieve this goal?

Iterations optimizing

Evidently there are several ways or directions to optimize iterative approach to software development and testing in particular. It makes sense to divide them into groups according to their type:

- Better planning of work
- Use of optimized techniques and technologies
- Flexible management

What is the Better planning of work? It is worth noting here that we should thoroughly determine and extract all the processes of software development that can go concurrently to reach our target of efficient optimization of work on the project. This is the first step to efficiency and it is interesting for us in the first place in the view of possibility to conduct testing procedures simultaneously with analysis, design, coding and other aspects of the work. All these phases should be coordinated in time to avoid both dead time and overwork. It should be noted that iterative methodologies of software engineering have been accentuating the necessity of concurrent execution of works within one iteration for a long time.

According to the methodologies the QA group is supposed to work simultaneously with the groups of analysis, design and development and other subdivisions. It allows you to divide the whole process into several concurrent processes without leaving all the QA procedures for the stage directly preceding the product release. In theory due to this approach the full-range testing is to fall behind insignificantly as compared to the other works within the iteration without creating any problems. Unfortunately, the theory is too far from the reality.

The point is that concurrent execution of works within one iteration itself can really be very helpful because it allows you to optimize the work on the project and provides you with the possibility of more efficient use of personnel, funds and time. But anyway we consider this approach as a preliminary condition to the next level of optimization - the advance beginning of iterations, which will be described later on.

The Flexible Management is close the Better planning of work, but in the given case it is considered not for planning of phases and deadlines of work, but for information exchange between participants of the project. As is known in intensive projects the requirements management is poor, if available. Even when available, the reading of designing documents can take a lot of time. From this point of view it is useful to provide managers of testers and directly testers with information in advance. The best way to do this consists in temporary work of the testers in direct contact with the programmers, for instance, in the same room. For managers it is sufficient to take part in all conferences of programmers, which allows them to know "how the project is getting on". The given approach will permit you at sufficiently informal level to fill a gap in communications and to partially compensate the lack of formal specifications. However this approach should not be considered as the only way of information exchange. Otherwise there is a risk to loose control over the situation, because knowledge and ideas of testers about a program can be erroneous and not coinciding with the opinion of programmers and designers.

Testing outsourcing is a good way to speed up execution of work with proper quality level, but the problem of communications and mutual understanding is brought again to the fore. To solve this problem the managers should work with due efficiency and accuracy.

And finally the last group is the use of the optimized testing techniques and technologies. It is not a secret that various programs and packages for testing automation available in the market are not always used in work. Nevertheless, the use of special packages can provide a considerable gain in time even without complex automation. Let's say the use of the system for

requirements management support in addition to systematization of requirements storage introduces a certain discipline in work allowing us to self-organize the process. Special attention should be paid to automatic testing packages that allows you to reduce time for regression testing by a factor of few tens and hundreds. Execution of load tests without them can be insoluble problem in principle. It is not worth neglecting introduction of self-diagnosis facilities in software systems. As regards to time resources self-diagnosis is much cheaper than the following manual revealing of errors.

Advance beginning of iterations

As it was mentioned above the method of advance beginning of iterations follows just after selection of concurrent processes and phases. We think that this technique is the most efficient way to use formal methods in execution of modern Internet projects under hard pressure of time.

What is the advance beginning of iterations? Let's consider this taking as an example the process of quality assurance of a software system in the course of its development.

Planning, design and implementation of tests are executed in the usual way simultaneously with the other works within iteration. Then the proper testing process begins and first of all we are supposed to conduct top-priority investigations that most sufficiently reduce the risks and control over the essential functionality and characteristics of the system. We don't conduct the full range of tests but only marginal ones that should be enough for completion of the iteration. When the analysis of the testing results is complete and we either decide to make amends or postpone them, this iteration is actually over and the next one begins. Using the proper set of test-cases will allow us to find the most critical defects first and we expect to discover about 80% /eighty percent/of all the errors during the first 20% of the testing time. It is obvious that in some cases defect detection rates in the process of testing may differ, but anyway, our suggestion is a good precondition for the advance beginning of the new iteration and it may promise a good benefit in the future from concurrent execution of work.

So the analysts, designers and developers start the next stage without wasting a minute on waiting for the full-range testing results, not applying pressure to a QA department The selected part of QA department staff continues the full-range testing of the current release at the same time. The defects discovered are sent to the development department without a delay and depending on their priority and complexity they can either be eliminated within the new iteration or cause creation of a new build of the preceding release. It's vital here that the full-range testing of the preceding release be over before the preliminary testing of the new release is complete, otherwise we will be unable to start its full-range testing on time.

So it looks like both iterations cross in time. While the full-range testing of the previous release continues the bigger part of the testing team is working on preliminary testing of the next release.

As our experience reveals this approach to testing considerably increases efficiency of labor and lowers risks as compared to the traditional iterative schemes. We can draw an analogy between our approach and the algorithms of command execution in the modern processors when they start parallel execution of the next commands in the sequence while the previous ones are still being executed.

Many of software companies have intuitively been using this approach or its analogs for quite a long time enjoying its efficiency. However, even most advanced object-oriented methodologies of software engineering don't have any recommendations on implementation of this approach. In fact, today you can't find any serious research works on the concurrent executions of iterations. But we do believe that this approach will be developing.

Now let's discuss the requirements and limitations that should be taken into consideration for efficient functioning of the testing optimization model we offer.

Advance beginning of iterations challenges

There's nothing absolutely perfect in the world, and, of course, the model for iterative testing optimizing we offer has its own advantages and limitations. These are conditions and limitations connected with concurrent execution of work in several iterations.

Firstly, this approach requires the use of additional personnel and equipment when working under pressure of time shortage. Time saving is the main goal of our optimization. However when you experience shortage of resources but time is not the most critical problem, for instance in an exactly planned project, our approach will be not only useless but may even create some difficulties.

Secondly, when you have enough resources at your disposal you should avoid using extreme forms of our approach. Even if you have a well-balanced team of professionals you will hardly be able to speed up the process for more than 50% by adding one concurrent testing iteration. It happens because the more concurrent iterations you have the more dependent they come on each other and also because you have to deal with a more complicated management and increased data exchange rates that, in its turn, causes additional operational and time expenses. Adding one more concurrent testing iteration is even less efficient. As our practice reveals, you can enjoy all the advantages of our approach having not more than 2 concurrent testing iterations at a time. The third iteration would make the control over change management extremely complicated and would slowdown the whole work. However this situation may be changed once automated systems of control are employed.

Thirdly, in some rare cases, the system requirements significantly changed within one of the development iterations may partly bring to naught the effect of concurrently executed works. On the one hand works on analysis, design or the requirements management performed within the new iteration may make obsolete a series of tests being concurrently executed. On the contrary defects detected in the process of concurrent testing may influence the requirements within the new iteration. It should be taken into consideration that all the key managers of the project including the managers of concurrent iterative testing should participate in the process of the project requirements management.

As it follows from what I've just said, the approach of advance beginning of the new iteration in the worst case may not only fail to boost the work but even somewhat slows it down!

What should be taken into account in the first place when planning the optimization to avoid it? There is a straight correlation between the efficiency of the optimization, sound management and quality of information exchange between the concurrent iterations.

Those companies that wish to use this new approach may face some difficulties because old-fashioned automation systems of development process are inflexible and do not support the new approaches for the latter are still under development and not formalized yet. Conservatism of top managers may also be a deterrent factor. In real life small teams using simple tools without inflexible management system to be upgraded can be in better situation. They can implement our approach painlessly, fast and cheap providing themselves with more mobility for the future.

Internet Time challenges

Practically all types of problems and their solutions discussed before can be applied to modern Internet projects. The whole complex of conflicts in development process of Internet projects served as a catalyst to search for new approaches to development management and organization. Now we are equipped with new knowledge and iterative approach. Let's consider once more how many Internet projects looks from this point of view.

Informal management process plays first fiddle in an Internet project. The situation depends on charisma of a project leader, «genius» of programmers and their capability to work 24 hours a day. Formal methods hardly can be used in such projects. There is such a shortage of time in the project that the first version with minimum functionality and content is immediately available for the users. The testing is often limited by sanity check or postponed to the moment, when the users have problems, for instance, with long time of site response or safety. Since requirements are not kept at all, or kept in greatly reduced and general form, it is very difficult to carry out functional testing. From the point of view of iterative approach in a typical Internet project there are no clearly selected iterations, they are mixed and exist only in heads of managers in the best case. Hence it follows that results of work are unpredictable and risks are high as it was mentioned before. How to struggle against this and what can be recommended to managers and workers of such projects?

And solutions…

It is worth noting that in our opinion the optimized iterative approach suits in the best way to execution of Internet projects. The point is that iterative nature of the process is recognized by the managers long ago, and they use it not by intention but subconsciously, intuitively. The same is true for the advance beginning of iterations. The given ways of optimization will help to introduce the required part of formalism in development process making it realized and controlled. So risks of unsuccessful project execution will be brought to minimum.

There is no doubt that revision of project management activity can take more than one day or even a month, but it is important to move gradually forward to the target. So first of all you should focus on the following ideas.

First of all, I want to wish managers to gain an understanding of development process, to try to select stages, even if very short, and then to "superimpose" iterative approach to software development. When this goal is reached, you can go to the use of all the suggested above optimization techniques of iterative development and testing. Special attention, evidently, should be paid to risks management, because for Internet projects there is no alternative to development of «rather good» software, so it is recommended to avoid maximalism in making decisions.

Anyway no matter whether you are an experienced company or just an Internet start-up, if you have a real need for testing optimization it makes sense to do it together with professionals. That’s why I’d like to dedicate the rest of my report to outsourcing of quality assurance and testing as one of the best solutions to the problems I’ve just been speaking above.

SQA Outsourcing

It should be noted that top managers share cautious attitude to the testing outsourcing. Despite the fact that in the classical textbooks independent testing is recognized as the best solution to assure a better quality of the product, project administrators often don’t make use of it. They worry that the testing company’s quality standards are low, its specialists are not aware of all the peculiarities of the product being developed and the internal technology of the developing

company, difficulties in communication between the specialists of both companies, and finally additional expenses low efficiency.

Anyway testing centers are spawning all over the world. The rationale? The factors I have just been talking about should be enough for a company to never use any external testing services. But as our practice reveals they are not.

I can provide you with some factors leading independent QA companies to success. The key motivation of project administrators to retain an independent tester's services is their wish to get some additional guarantee of their project successful completion. No matter how good their own testers might be they as a rule have one-side opinion on the system being developed and work under pressure of time and their supervisors and as a result they are unable to adequately estimate the situation. Testing outsourcing is like audit in accounting. It provides you with the comfortable feeling that everything is under control.

However in addition to subjective factors there exist objective factors for using testing outsourcing, namely, qualification of personnel engaged in complex of testing works. In most cases professional level, experience and erudition of the specialists of an independent company is higher than in your organization simply because they are permanently dealing with testing of a wide range of software and have techniques for localization of larger amount of errors in software. It is unlikely that testers of your company are as much motivated. Alternative view of independent testers will help you to see what was hidden from you before. I am more than sure that they will offer you an interesting set of tests and software analysis methods that would hardly ever occur to you. Combination of internal testing with testing outsourcing provides you not only with excellent results but also with valuable experience. Namely high qualification of the specialists of a SQA company will permit you to reduce time for the next iteration of software development.

References

Spitsnadel V.N.(2000) Fundamentals of system analysis. «VOENMEH» University, St. Petersburg, Russia

Spitsnadel V.N. (2000) Quality assurance systems. «VOENMEH» University, St. Petersburg, Russia

Lipaev V.V. (1999) System design of complex software facilities for information systems. Institute of system programming of RAS. Moscow, Russia.

Prangishvili I.V., Abramova N.A., Spiridonova V.F., Kovriga S.V., Razbegin V.P. (1999) Search for approaches to solution of problems. Institute of management problems of RAS. Moscow, Russia.

Quality Week Europe 2000 Conference Materials (2000) Brussels, Belgium.

Euro Software Testing Analysis & Review 2000 Conference Materials (2000) Copenhagen, Denmark.

QW2001 Paper 2M1

Mr. Scott Jefferies
(Technology Builders, Inc.)

A Requirements-Based Approach To Delivering
E-Business And Enterprise Applications

Key Points

- Minimize application failure and revenue loss and maintain loyal customers
- Shorten development timeframe, decrease costs and improve application quality
- Integrated requirements-based approach that enables project teams to optimize initiatives

Presentation Abstract

This presentation will demonstrate to attendees how a requirements-based approach to delivering E-business and enterprise applications will minimize application failure, avoid revenue loss and maintain loyal customers. And with a proven process in place, organizations can make an even bigger impact on the development cycle, shortening the development timeframe, decreasing costs and significantly improving application quality.

This presentation describes an integrated requirements-based approach that enables project teams to:
* Gather and define requirements
* Analyze the requirements to eliminate ambiguities, conflicts and other errors
* Manage requirements and their evolution throughout the development cycle
* Define test completion criteria
* Design and build test cases based upon requirements
* Review test cases with stakeholders
* Execute tests and verify results
* Verify test coverage
* Track defects and
* Manage the test repository

Attendees will gain an understanding of How To:
* Fully document requirements to know exactly what the end users need
* Resolve ambiguities, conflicts and other errors so to ensure the right requirements are met
* Define the test completion criteria to guarantee that the application is ready for release
* Design, build and execute the minimum number of test cases required to fully test the application

* Track defects to determine where and why errors occurred, allowing to reduce the probability of future errors
* Manage test libraries to provide a set of reliable, repeatable tests

About the Author

Scott Jefferies is a TBI Technology Engineering Manager with over 25 years' experience in providing business and information management solutions to Fortune 1000 companies. Scott is experienced in the installation and implementation of automated software quality (ASQ) tools on a variety of platforms.

Since joining TBI, Scott has been responsible for implementation of both requirements management and automated testing tools, with a focus on the integration of those tools. Scott was also a key player in establishing TBI's requirements-based approach to delivering e-business and enterprise applications, which includes requirements management, test case design, test and defect management, and automated testing.

A Requirements-Based Approach to Delivering E-business and Enterprise Applications

Scott Jefferies
Technology Engineering Manager
Starbase Corporation

Agenda

- Gathering and defining requirements
- Performing ambiguity reviews
- Managing requirements
- Designing test cases
- Defining test completion criteria
- Other steps in the process

Gathering and Defining Requirements

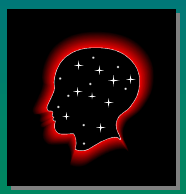
- Who should be involved:
  - Business analysts
  - Users
  - Other project stakeholders

starbase

3

Gathering and Defining Requirements

- How to gather requirements:
  - User interviews
  - Brainstorming
  - Facilitated sessions
  - Project specification



starbase

4

Gathering and Defining Requirements

- **What** to gather:
  - The requirement itself
    - "the system shall…"
  - Attributes
    - Priority, need, precedence, relationships with other requirements
  - Supporting information
    - Graphics, object models, regulations

Performing Ambiguity Reviews

- **Who** should be involved:
  - Newest member of the team
  - Business analysts
  - Users
  - Other project stakeholders

Performing Ambiguity Reviews

- Why we should perform ambiguity reviews:
  - Eliminate ambiguities
  - Identify conflicts and logic errors
  - Reorganize requirements for clarity

7

starbase

Performing Ambiguity Reviews

- What we are looking for and how to find them:
  - Traceability and inconsistency errors
    - Requirement traces to nothing (orphans)
      - Use Traceability Matrix to identify
    - Terms used inconsistently

8

starbase

Performing Ambiguity Reviews

- What we are looking for and how to find them:
  - Imprecise terminology
    - Acronyms, company- or industry-specific jargon, non-explicit terms ("quickly," "user-friendly")
      - Use outside resource to help identify
      - Create project glossary to explain terms used

Imprecise Terminology Exercise

How many examples of imprecise terminology can you find in the following:

The ATM shall respond quickly and in a user-friendly manner to any user action, and print a TR when the transaction is completed.

Imprecise Terminology Exercise

How many examples of imprecise terminology can you find in the following:

The ATM shall respond quickly and in a user-friendly manner to any user action, and print a TR when the transaction is completed.

Performing Ambiguity Reviews

- What we are looking for and how to find them:
  - Ambiguities
    - "If the table is next to the chair, then move it."
      - Look for exceptions, use outside resource
  - Logical errors
    - "If A and B then C" … "If A and B then Not C"
      - Use cause-effect graphs, rearrange requirements

Performing Ambiguity Reviews

- What we are looking for and how to find them:
  - Undocumented assumptions
    - "The system shall perform the calculation in the usual way."
      - In-depth interview by business analyst
      - Make inferences

Performing Ambiguity Reviews

- How the requirements are improved:
  - Become testable
    - Deterministic, unambiguous, complete, non-redundant, traceable, explicit and feasible
  - Become easier for developers to work with
  - Become easier to manage

Transportation Device Example

- Transport one person at a time
- Over hard flat surfaces
- At speeds not to exceed 20 miles per hour
- For distances up to 2 miles
- Using only person power for locomotion
- Personal comfort is not important

???

Managing Requirements

- What is involved:
  - Manage changes (errors, new requirements and user requests)
    - Reduces scope creep due to unnecessary changes
  - Establish priorities
    - Focuses development on core set of requirements

Managing Requirements

- What is involved:
  - Assign responsibilities
    - Assists in communicating changes
  - Document rationale
    - Allows project team to understand why certain decisions were made or changes not made

Managing Requirements

- What is involved:
  - Trace requirement relationships
    - Allows impact analysis for more informed decisions
  - Communicate changes
    - Allows entire project team to understand current project status and scope

Managing Requirements

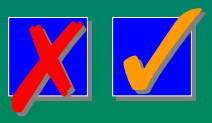
- What is involved:
  - Establish baselines
    - Tracks scope creep and changes for management
  - Track requirement histories
    - Ensures audit trail

19

starbase

Designing Test Cases

- What factors should be considered:
  - Test cases should be based on requirements
  - Test data should be designed to provide maximum coverage with minimum number of tests



20

starbase

Designing Test Cases

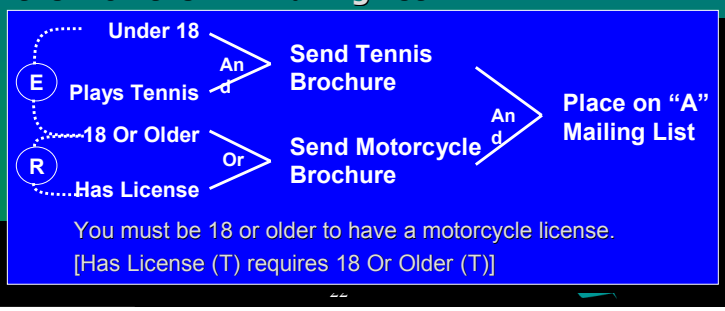
- What factors should be considered:
  - Use cause-effect graphing to clarify requirement relationships and testing needs
  - Group test cases into test sets for ease of execution/organization

starbase

Cause-Effect Graph Example

Criteria

- If the person is under 18 and plays tennis, then send them a tennis club brochure.
- If the person is 18 or older, or has a motorcycle license, then send them a motorcycle club brochure.
- If the person was sent both brochures, then put them on the "A" mailing list.



Designing Test Cases

- How testers are currently designing tests:
  - "Gut feel"
  - Live data
  - Brute force combinations

starbase

Designing Test Cases

- Why current methods fall short:
  - Too many tests, too little time
  - Varying skill levels and experience
  - Not enough coverage (functional or code) to ensure system integrity

starbase

Designing Test Cases

- The solution:
  - Use scientific methods to produce the minimum number of test cases that will validate all of the functional requirements
  - Result: 100% functionality coverage and 85%-90% code coverage on the first pass

Reviewing Test Cases

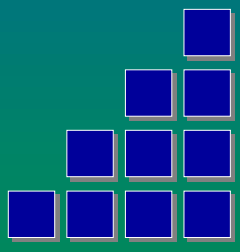
- Who should be involved:
  - Spec writer
  - User/domain experts
  - Developers
- Why this is important:
  - Minimizes misunderstandings

Defining Test Completion Criteria

- Why this is important:
  - Sets policy for when software will be considered for release
- What should be specified:
  - Which tests must have been performed
  - Which tests must have passed
  - How many iterations of the testing cycle need to be clean

Other Steps in the Process

- Build tests
- Execute tests and verify results
- Verify test and functional coverage
- Track defects
- Manage repositories



Summary

- Gather and define requirements
- Analyze them to eliminate ambiguities, conflicts and other errors
- Manage requirements and their evolution throughout the development cycle
- Design and build test cases based upon the requirements
- Review test cases with spec writer, user/domain experts, developers

29

Summary

- Define test completion criteria
- Execute tests and verify results
- Verify test coverage
- Track defects
- Manage the requirements, test, code and defect repositories

30


starbase
Questions?

QW2001 Paper 2M2

Mr. Robert Benjamin, Ms. Ruth Pennoyer & Ms. Karen Law (Spherion Corporation)

Pre-Defining Success: Incorporating e-Metrics Into Business And Technical Requirements For Web And e-Business Solutions

Key Points

- Requirements Management
- eBusiness
- Risk Managemnet

Presentation Abstract

For over twenty years, quality management professionals have been saying that understanding of requirements, and effective, end-to-end management of those requirements, are the most critical determinants of success or failure in information systems. Several widely-quoted studies, including, but not only, the Standish Group's CHAOS Report, back this view. Drilling down further, many believe that a significant number of projects fail, over 16% in some studies, when the requirements process fails to include measurable improvements in business value as the key determinants of success. In other words, projects that come in on time, under budget, and with little or no serious defects can still fail if they do not return any real business value on their investment.

Although business requirements have crept into the consciousness of business sponsors and developers alike over the past decade, the explosion of Web-enabled and eBusiness applications seems to have halted, maybe even reversed this trend. Quality management professionals, supported by credible industry observers, report a decline in understanding of, or concern for the importance of end-to-end quality management among their customers, especially those under intense pressure to deliver solutions against unrealistic deadlines. Moreover, stories about "dot.com" failures abound, with many calling those failures preordained due to faulty business models with no realistic business metrics. They fail because they start out as bad ideas.

Ensuring that Web-enabled and eBusiness applications start out as GOOD ideas demands new metrics for success, based on an understanding of what makes these applications either successes or failures, and the inclusion of these metrics into business requirements. Incorporating e-Metrics into the requirements for a Web or eBusiness system can enable project teams to produce more effective design, development, test and deployment plans that are based on requirements and business risk.

This presentation will describe how e-Metrics are incorporated into the elicitation and validation of Web and eBusiness system requirements.

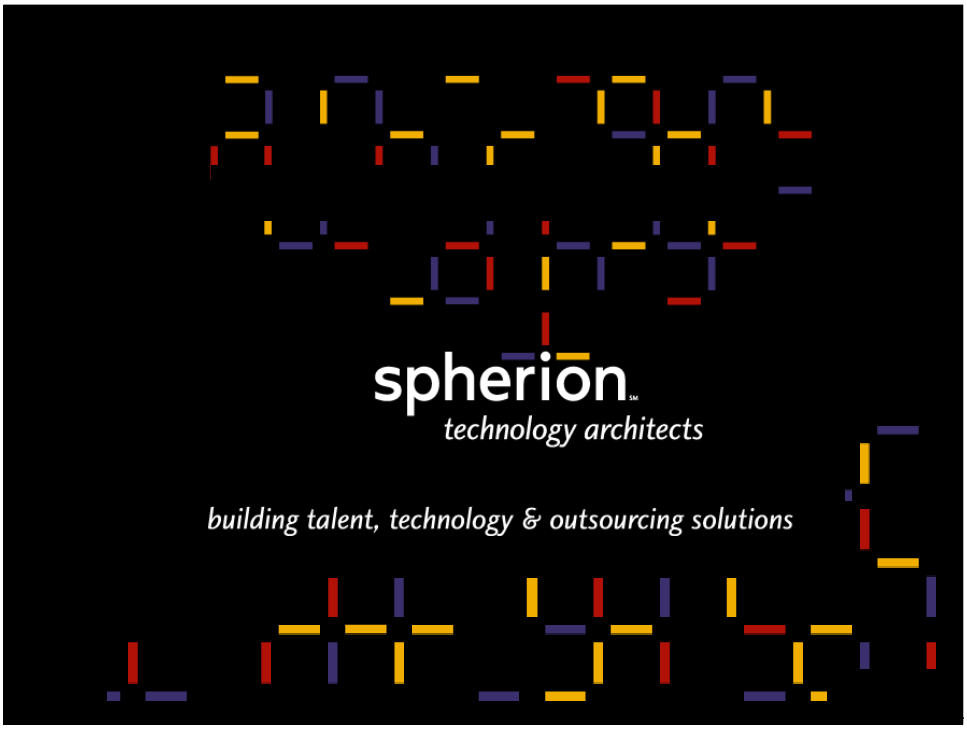
About the Author

Robert Benjamin, Author
Mr. Benjamin is a Certified Quality Analyst (through the Quality Assurance Institute) with over three decades experience in Information Technology consulting, sales, and marketing, the last twelve specializing in Software Quality Management. He has designed new process lifecycles based on the Software Capability Maturity Model for three major IT development organizations, co-founded the New York City Software Process Improvement Network, supported major software process improvement initiatives, and facilitated business and technology strategy planning projects. He has also led testing and development teams in successful project recovery efforts. He is currently a regional Software Quality Management Practice Director for Spherion Technology Architects, Spherion Corporation.

Ruth Pennoyer, Co-Author
Ms. Pennoyer is a Certified Quality Analyst (through the Quality Assurance Institute), and a Certified Software Quality Engineer (through the American Society for Quality). She has over twenty-eight years experience in Information Systems and is a principal author of Spherion Technology Architects' requirements and risk-based testing process. She has experience in quality assurance, quality process assessment, project and testing management, corporate management, methodology development, and training. Areas of special expertise include risk management, quality program implementation, organization assessment and planning, and management staff development. She is currently Managing Consultant for Spherion’s Software Quality Management Practice in New Jersey and Project Manager for a major Software Quality Assurance project for the New York City Government.

Karen Law, Co-Author
Ms. Law is a Certified Quality Analyst and Certified Software Test Engineer (through the Quality Assurance Institute), and a Certified Software Quality Engineer (through the American Society for Quality). She is a principal contributor to Spherion Technology Architects’ training courses in Software Quality Management for Web and eBusiness systems. Ms. Law has led recent strategic quality management projects for Internet software companies, managed a Web application test laboratory, including support and testing of its production site,

evaluation of automated test tools, and development of a software testing methodology and Software Quality Assurance process. She is currently Deputy Project Manager for a major Software Quality Assurance project for the New York City Government.



Pre-Defining Success:

Incorporating e-Metrics into Business and Technical Requirements for Web and e-Business Solutions

What we will cover…

- When failure looks like success
- eBusiness and the changing meaning of Metrics
- Traditional metrics meet emerging Metrics
- Customer satisfaction eMetrics
- Internal business improvement eMetrics
- Incorporating eMetrics into Requirements practices
- Simple tools

spherion

When Failure Looks Like Success…

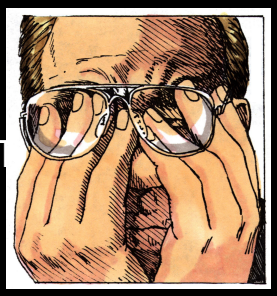
- The phenomenon of the long-fuse failure
- The problem with “Great Ideas”
- Where do they fail
- IT Metrics and Great Ideas

spherion

The Phenomenon of the Long-Fuse Failure…

- A GREAT IDEA!
- On time
- Within budget
- Meeting customer requirements
- Zero defects, but…

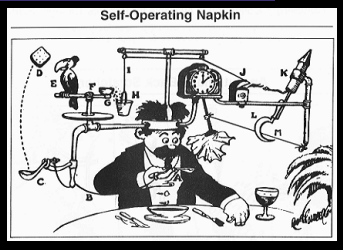
- NO BUSINESS IMPROVEMENT



spherion

The Problem with Great Ideas…

Most aren't.



the MICHAEL RICHARDS show

spherion

Where They Fail…

- Not tied to *any* business strategies
- Not tied to the *right* business strategies
- Not measurable against actual business improvements
- Tied to faulty business cases
- Take resources away from REALLY great ideas

spherion

IT Metrics and "Great Ideas"

- IT Metrics validate – or invalidate – Great Ideas
  - Provide the foundation for a v
  - Are inherently measurable

spherion

eBusiness and the Changing Meaning of Metrics…

- eBusiness models and IT Metrics
- What still applies, what doesn't

spherion

eBusiness Models and IT Metrics …

- eBusiness models are redefining both the meaning and the uses of traditional metrics
- eMetrics come from a variety of sources
  - Financial models
  - Non-financial models
  - Commonly accepted performance models

spherion

What Still Applies, What Doesn't…

- Traditional metrics for IT projects are still valid for eBusiness projects.
- eBusiness projects also need eBusiness metrics.
- Many emerging eBusiness metrics may also apply to NON-eBusiness projects
- Metrics – Traditional or Emerging – are the highest level of Information Technology requirements we can define

spherion

Traditional Metrics Meet Emerging Metrics…

- Traditional metrics
  - Project metrics
  - Financial metrics
  - NON-financial metrics
- Emerging metrics
  - Commonly-accepted performance models
  - Business modeling methods
  - Drivers of change

spherion

Traditional Metrics…

- *Project* Metrics…
  - Time
  - Size
  - Cost
  - Effort
  - Defects found and fixed

spherion

Traditional Metrics…

- *Financial* Metrics: liquidity, leverage, operating and performance ratios:
  - Current ratio (current assets/current liabilities)
  - Debt ratio (total liabilities/total assets)
  - Average Inventory Turnover (cost of goods sold annually/average annual inventory)
  - Average collection period ratio
  - Return on Sales (net profit after taxes/net sales)
  - Quarter-to-quarter sales growth

spherion

Traditional Metrics…

- *Non-Financial* Metrics:
  - Employee turnover
  - Account turnover
  - Trends in market share
  - Trends in customer retention

spherion

Emerging Metrics…

- Commonly-accepted performance models
  - Balanced Scorecard
    - Financial perspective, plus:
      - Customer perspective
      - Internal Business perspective
      - Innovation and Learning perspective
  - Baldrige Performance Award
- Business modeling methods
- Drivers of change
  - Business drivers
  - Technology drivers

spherion

Customer Perspective Metrics …

- Things that indicate how well we meet customer needs
  - Customer loyalty and retention
  - Drivers of overall customer satisfaction and value
  - Customer/Consumer Satisfaction
  - Partnering Index

spherion

What They Tell Us…

- Customer behavior indicates customer satisfaction
- eBusiness functionality can track customer behavior data and organize it into customer satisfaction metrics

spherion

Examples…

- Visitor-to-customer conversion percentages
- Abandonment rates for shopping carts
- Completion rates for checkouts
- Conversions of wish list items to shopping cart items
- Positive and negative email
- Trends in new accounts per month
- Trends in cancelled accounts per month
- Trends in median customer revenue per month

spherion

Internal Business Perspective Metrics…

- Measures of process efficiency, effectiveness, and adaptability
- Measures of asset utilization
- Based on operations data

spherion

What They Tell Us…

- How well we use our assets
- What we should be benchmarking against external norms

NOT whom we should punish!

Examples…

- Late or incorrect orders
- Reworked products
- Submitted orders
- Approved orders
- Planned throughputs
- Actual throughputs
- Equipment utilization

spherion

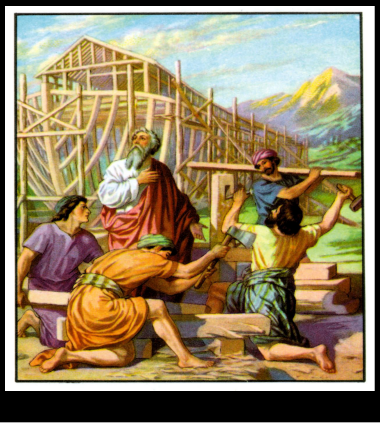
Innovation and Learning Perspective Metrics…

- Measures of activities, investments, and results related to the organization's people and infrastructure, such as
  - Investment in and effectiveness of R&D
  - Investment in and effectiveness of training
  - Investment in and effectiveness of compensation

spherion

What They Tell Us…

- How well we are positioning for the future



spherion

Examples…

- Patents awarded
- Employee professional certifications earned
- Organizational awards and certifications
- New products and services launched
- New product and service lead times
- Employee suggestions

spherion

Baldrige Performance Award Criteria…

- Leadership
- Human resources focus
- Strategic planning
- Process management
- Customer and market focus
- Business results
- Information and analysis

spherion

Baldrige Performance Award Criteria - Example…

- Strategic planning
  - "…customer-driven quality is a strategic view of quality. The focus is on the drivers of customer satisfaction, customer retention, new markets, and market share — key factors in competitiveness, profitability, and business success…"
  - Implied metrics:
    - Customers retained vs. customers lost
    - Growth in new customers in new market segments
    - Industry estimates of market share

spherion

Baldrige Award Criteria…

- Leadership
- Strategic Planning
- Customer and Market Focus
- Information and Analysis
- Human Resource Development and Management
- Process Management
- Business Results

spherion

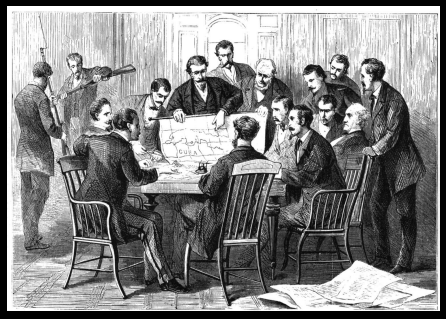
Business Modeling - Five Forces Model…

- Supplier metrics
- Buyer metrics
- Barriers to new entrants
- Substitution metrics
- Competitive rivalry metrics

(They can really gang up on you)

Business Modeling - SWOT Analysis…

- Strengths
- Weaknesses
- Opportunities
- Threats



spherion

Drivers of Change…

- Examples of business drivers
  - Trends toward part-time workforce
  - Trends in public policy on privacy
- Examples of technology drivers
  - Limits to Moore's Law with current technology
  - Trends in energy and environment costs

spherion

Metrics Specific To eBusiness Models…

- First-generation eMetrics focus on ACTIVITY
  - Number of hits
  - Length of stay
  - Source of referral
  - Return ratio
- Second-generation eMetrics focus on RESULTS
  - Trend in average customer revenue
  - Trend in completed transactions
  - Trend in repeat sales, PLUS
  - All other traditional metrics adaptable to eBusiness

spherion

Measurable Business Improvements…

- Every metric identifies an improvable business attribute
- Every improvable business attribute represents a potential project goal or objective
- Each project goal or objective is a top-level requirement

spherion

Incorporating eMetrics Into Requirements Practices…

- Commonly-accepted templates can help
  - Example: Volere Requirements Specifications Template, available from the Atlantic Systems Guild at http://www.atlsysguild.com/Site/Robs/Template.html
- Begin with a Statement of Principal Drivers or Business Purposes, stated in terms of business metrics.

spherion

Example 1…

We need to reduce the cost of sales ratio for products that generate repeat sales.

To achieve this, we must use web technology to have customers initiate, process, complete, and track orders without the intervention of a salesperson. Further, we must enable customers to create their own reorder process and manage it online.

spherion

Example 2…

We can increase revenue per customer by promoting addition of complementary products as part of on-line sales.

To achieve this, we must use web and data base technologies to cross-link complementary products, track customer purchasing patterns and provide incentives for adding complementary products to each sale.

spherion

Prioritizing Business Improvement Goals…

- Key questions
  - What are my critical business and technology drivers?
  - What business improvement goals best address my critical business drivers?
  - What is the *true cost* to achieve them?
  - When and how can I achieve them?
  - How do I need to manage the risks?

spherion

Simple Tools…

- Net present value analysis
- Pareto analysis
- Force Field Analysis
- Root-Cause analysis

Repeat: SIMPLE tools…


Questions…
spherion

QW2001 Paper 3M1

Mr. Timothy Kelliher, Dr. Daniel Blezek, Mr. William Lorensen & Dr. James Miller
(GE Research & Development)

Six-Sigma Meets Extreme Programming: Changing the Way We Work

Key Points

- Six Sigma Applied To Software
- Extreme Programming in Practice
- Changing Group Behavior

Presentation Abstract

Introduction

For the past 5 years GE has been following and enhancing the Six Sigma quality model. During this same period Extreme Programming has emerged as an alternative to the traditional software development process. In the past year the authors have been following a process which merges elements from each of these disciplines. This paper gives the background and describes their current approach.

Six Sigma Basics

Behavior is a function of values. We measure what we value. Therefore it follows that to change behavior we have to change what we measure. This simple truth is the core of the Six Sigma quality program at GE We must adapt our measurement systems to look at true life cycle costs and reward behavior that improves on these costs. Basic Six Sigma focuses on improving existing processes and understanding the key control factors for these processes. Within GE this has been stretched far beyond manufacturing to include processes throughout the company including sales, services, and engineering. In this basic approach GE has followed the Mikel Harry's Six Sigma Breakthrough Cookbook [Harry 94]. This approach breaks down any quality problem into four steps: Measure, Analyze, Improve, and Control. In the measure phase the practitioner characterizes the process in terms of Critical To Quality, CTQ, characteristics. These are the elements that the customer considers as the key factors for process success. Specific, measurable targets are established for these CTQs both for mean performance and statistical variation. In the analyze phase the CTQs are broken down to understand the factors that influence their performance. These are divided into two group, key control parameters and noise parameters. In the improve phase measures of the key control parameters are further collected and studied to learn how their performance can be

tuned to optimize the effected CTQ's performance. In the final phase, control, a method is developed to ensure that the CTQ performance will remain at its improved value. This reliance on gathering data from actual measures is another of the fundamental aspects of the Six Sigma approach. Instead of relying on intuition to guide changes Six Sigma demands that data be used to make decisions.

Extreme Programming

Extreme Programming, XP, is a emerging approach to software development that focuses attention on the granularity of work elements from concept to implementation, the testing of the work products, the reduction of 'overhead' activities and the involvement of the customer in the development process. XP is successful because it emphasizes customer satisfaction and promotes teamwork. The most surprising aspect of XP is its simple rules and practices. They seem awkward and perhaps even naive at first, but soon become a welcome change to developers who adopt the XP model. Many customers enjoy being partners in the software process and developers actively contribute regardless of experience level. The rules and practices must support each other. Together they work to form a development methodology. Unproductive activities have been trimmed to reduce costs and frustration. This approach to refining the development process is in keeping with the Six Sigma tenet of behavior being driven be values and measurements. In general people value their time and as such are unwilling to spend time on 'unproductive' activities. Thus all activities that are elements of XP need to be transparently productive or they will soon fall into disuse. XP, as defined by Kent Beck[Beck 2000], has a number of elements.

Six Sigma Applied to Software Development

Green Belt Projects

At GE every person in the company was challenged to understand and apply Six Sigma methods to their job. To back up this challenge every GE employee has been trained in Six Sigma tools and practices. Every professional within the company is expected to demonstrate use of the tools to perform their job. The demonstration takes the form of green belt projects. For people just undergoing training these are generally small projects that look to make demonstrable improvements to quality in some aspect of their work and to demonstrate proficiency with elements of the Six Sigma approach. The projects are selected by the trainee and mentored by a more experienced Six Sigma leader. In software groups the training projects have generally focused on some measurable aspect of the coding or testing process. Topics such as regression test coverage, memory usage, and code style are all projects that have been done for training. The general experience with these projects has been that they do a reasonable job of defining CTQ's, e.g. all files shall have a mean of 80% of their lines of code exercised by regression tests with a standard deviation of no more than 5%. The analysis and improve phases uniformly yield an improvement in CTQ performance, generally as the result of personal effort on the part of the trainee, e.g. they implemented a sufficient number of tests to drive the coverage up. The control phase is then where

problems arise. In this phase, the trainee reports on a plan to monitor CTQ performance on a periodic basis. Having gotten their training completed the trainees then go back to working exactly as they have in the past. The problem is in the control step. The quality gain, once achieved, was a lone effort and remains that way in the control stage. Thus the rest of the team ends up with little concern for the gain and eventually it is forgotten.

Changing the Way We Work

As the same a group of workers that has gone through or is going through green belt training began to adopt elements of extreme programming an interesting phenomenon occurred. The group has a five year history of developing an open source package so they had been early adopters of elements of the approach. The idea of collective code ownership, refactoring, simple design, and coding standards were all part of the group ethic to a greater or lesser degree. One of the new elements of extreme programming that the group decided to experiment with as it was embarking on Six Sigma was a modified form of pair programming. This has made all the difference in the impact made by Six Sigma.

In the local version of pair programming the concept is extended to be paired, or trippled, work. Not only is time programming spent together but other work time is also spent together. This includes time spent on green belt projects. The time being jointly directed as in pair programming. This means that each member of the work group ends up having to buy into the value of the green belt project before it begins. Given this expanded ownership of note just the code output but also of the quality outputs the control phase got a renewed emphasis. With additional owners comes additional pressure to achieve a suitable performance on the CTQ. After the first round of projects the CTQ performance had risen noticeably and any backsliding was more rapidly noticed. The group then moved to the next element of extreme programming, continuous integration. Measuring the quality results entailed running a test of sorts for each aspect of quality that was being checked. Running these tests by hand became cumbersome so the group developed a nightly test harness that runs each of the quality tests and collects all of the data to present a single comprehensive view of measured quality. It was a small move from this point to start a continuous test process that runs a slimmed down set of the quality tests whenever code is checked into the source code repository.

The continuous measure of quality then is the final blending of extreme programming with Six Sigma. Behavior and measurement are well aligned. System performance against external, customer defined CTQs, expressed through regression tests, and internal, team defined CTQs, expressed through quality tests are available on a continuous basis. The teams are not able to move forward unless the results of their work is a clean dashboard as measured by both continuous and nightly dashboards. Extreme Six Sigma has become the way we work.

About the Author

Timothy P. Kelliher is a Computer Scientist at GE's Corporate Research and

Development Center in Schenectady, NY. He has over 15 years experience in systems and software engineering. At the center he has worked on Software Engineering CASE tools, Human Computer Interaction, and Software Quality systems. He is a Six Sigma Master Black Belt and spends much of his time instructing and mentoring the GE software development community. He is co-author of “Engineering Complex Systems with Models and Objects” published by McGraw-Hill, 1997.

Daniel Blezek is a Computer Scientist in the Visualization and Computer Vision Program at GE's Corporate Research and Development center. He holds a Ph.D. from the Mayo Graduate School in Biomedical Engineering. His research interests include medical image segmentation, advanced rendering algorithms, and practical software quality techniques.

William Lorensen is a Graphics Engineer at GE's Corporate Research and Development Center in Schenectady, NY. He has over 30 years of experience in computer graphics and software engineering. William is currently working on algorithms for 3D medical graphics and scientific visualization. William is the author or co-author of over 60 technical articles on topics ranging from finite element pre and postprocessing, 3D medical imaging, computer animation and object-oriented design. He is a co-author of "Object-Oriented Modeling and Design" published by Prentice Hall, 1991. He is also co-author with Will Schroeder and Ken Martin of the book "The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics" published by Prentice Hall in November 1997. Mr. Lorensen holds twenty seven US Patents on medical and visualization algorithms.

James Miller is a Computer Scientist at GE's Corporate Research and Development Center in Schenectady, NY. He joined GE after receiving his PhD from Rensselaer Polytechnic Institute in 1997. His thesis topic was in Computer Vision. At GE James has become a primary contributor to vtk software algorithm development and testing. For the past year, he has been the project leader on a research project for Lockheed Martin involving the inspection of large airframes using laser ultrasonic techniques.

Six Sigma Meets Extreme Programming

Timothy P. Kelliher
General Electric
Corporate Research and Development
kelliher@crd.ge.com

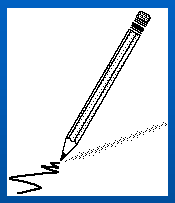
Outline

- Introduction
  - Six Sigma at GE
  - Design for Six Sigma
  - Extreme Programming
- Similarity
- Differences
- Results In Practice

Six Sigma - Cutting to the Core

Behavior is a function of Values

$$B = f(V)$$



Behavior

The way in which a person or group of people responds.

Values

The complex of beliefs, ideals, or standards, which characterizes a person or group of people.

. . . What are the "common beliefs" which characterizes our organization ?

Changing Focus from Output to Process

$$Y = f(X)$$

Y	Effect
Dependent	Symptom
Output	Monitor

$X_1 \ldots X_N$	Cause
Independent	Problem
Input-Process	Control

Identifying and fixing root causes will help us obtain the desired output

Critical-To-Quality (CTQ) Characteristics

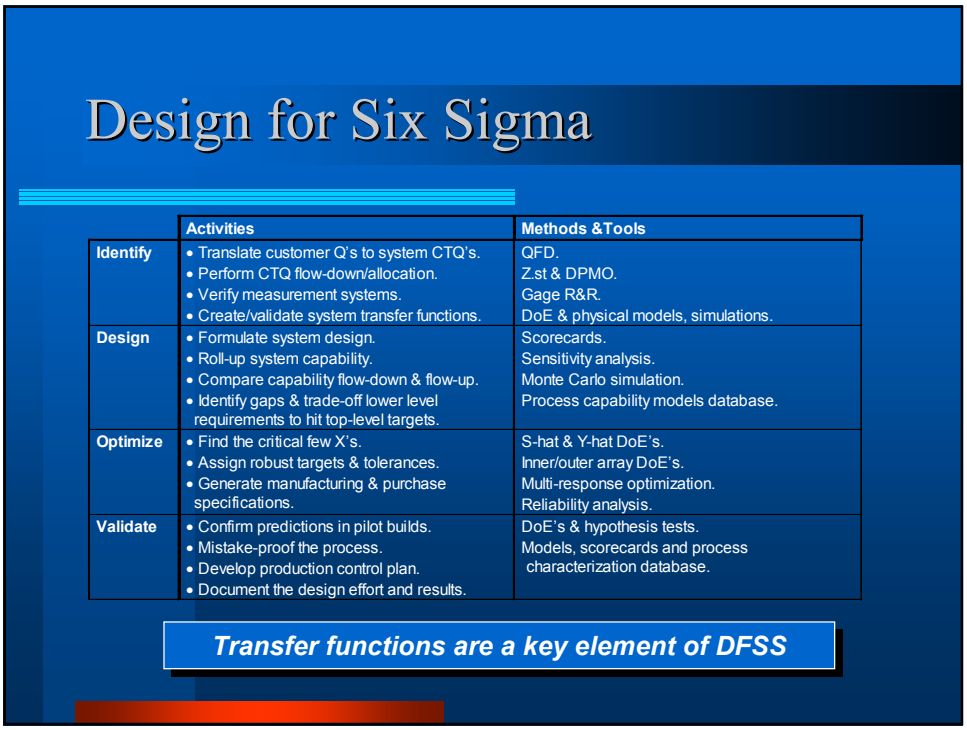
- **Customer states as critical to quality through a survey, Quality Function Deployment result, or by question / inspection**
- **High combined risk priority factor, as from a Failure Modes and Effects Analysis (FMEA)**
- **Sufficient economic benefit from defect reduction**
- **Regulatory or safety-related requirement**

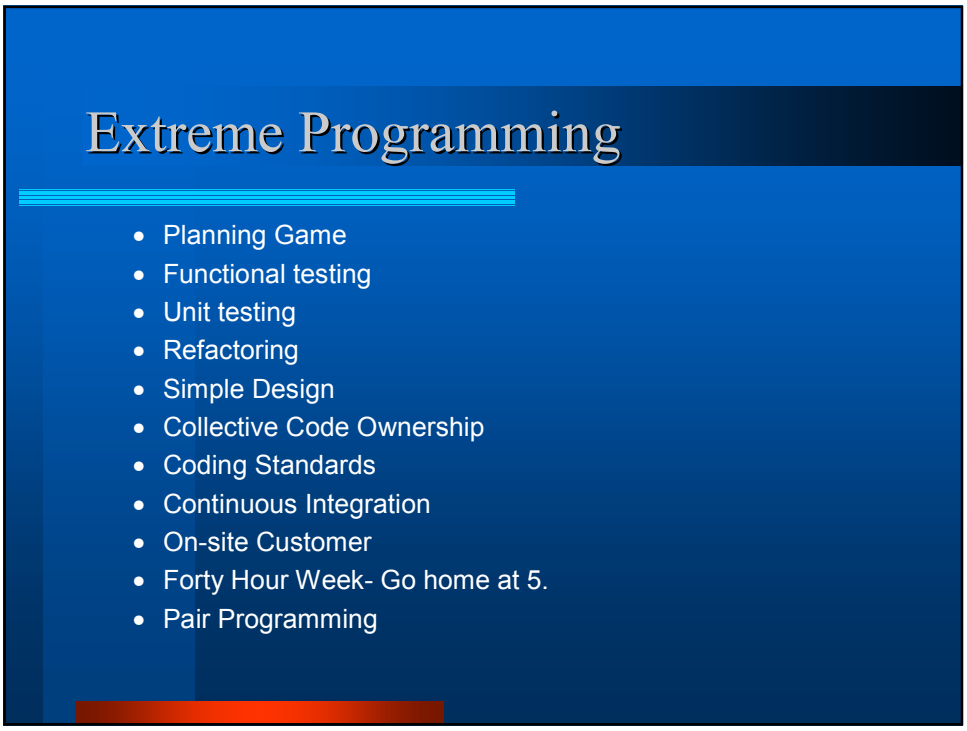
Most CTQ's are customer-driven; but risk, economics, and regulation may drive others

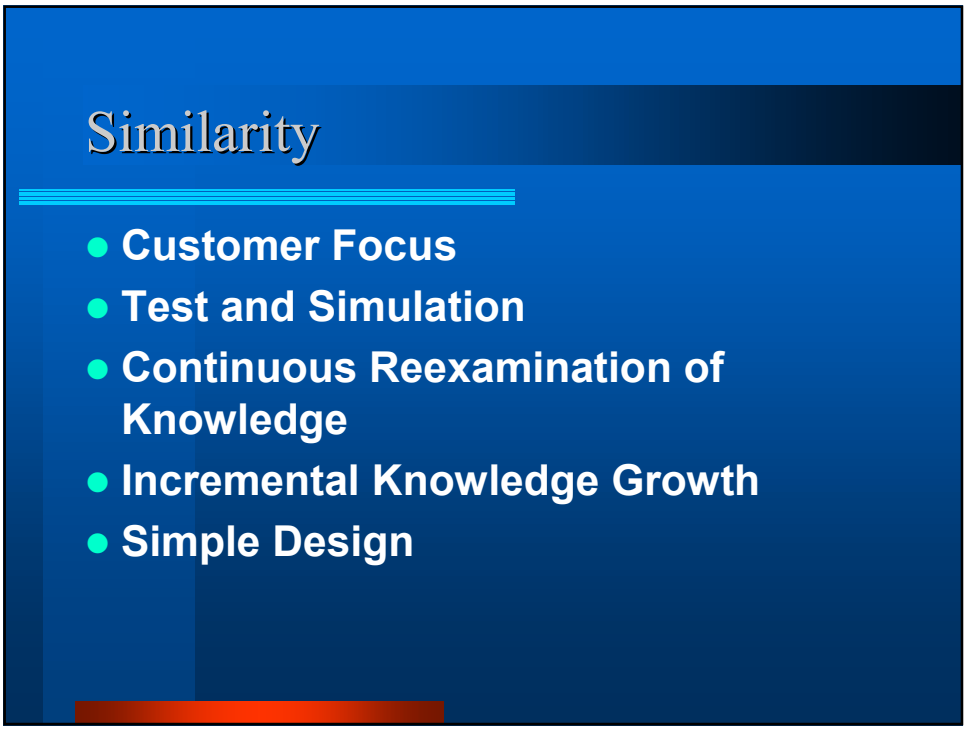
The Breakthrough Cookbook

Step	Description	Focus	Tool
Measure			
1	Select CTQ characteristic	Y	Customer, QFD
2	Define performance standards	Y	Customer, blueprints
3	Validate measurement system	Y	Gauge study
4	Establish product capability	Y	Capability indices
Analyze			
5	Define performance objective	Y	Team
6	Identify variation sources	X	Multi-Vari
Improve			
7	Screen potential causes	X	DOE-Fraction
8	Discover variable relationships	X	DOE-Full
9	Establish operating tolerances	X	DFM
Control			
10	Validate measurement system	X	Gauge study
11	Determine process capability	X	Capability studies
12	Implement process control system	X	SPC


Design for Six Sigma: The Goals
Reactive Design Quality
Transition to ...
Predictive Design Quality
FROM
TO
Intuitive Scheduling
Random Improvements
Evolving Requirements
Design rework/tweaking
Build & Test
Measurement
"Test in" Quality
Repeatable Predictive Process
Data Driven Decisions
QFD, CTQ Flowdown
Control Critical Design Parameters
Modeling/Simulation with Design/Process Capability Flowup
Statistical Quality Prediction
"Design in" Quality


Design for Six Sigma
Activities
Methods &Tools
Identify
• Translate customer Q's to system CTQ's.
• Perform CTQ flow-down/allocation.
• Verify measurement systems.
• Create/validate system transfer functions.
QFD.
Z.st & DPMO.
Gage R&R.
DoE & physical models, simulations.
Design
• Formulate system design.
• Roll-up system capability.
• Compare capability flow-down & flow-up.
• Identify gaps & trade-off lower level requirements to hit top-level targets.
Scorecards.
Sensitivity analysis.
Monte Carlo simulation.
Process capability models database.
Optimize
• Find the critical few X's.
• Assign robust targets & tolerances.
• Generate manufacturing & purchase specifications.
S-hat & Y-hat DoE's.
Inner/outer array DoE's.
Multi-response optimization.
Reliability analysis.
Validate
• Confirm predictions in pilot builds.
• Mistake-proof the process.
• Develop production control plan.
• Document the design effort and results.
DoE's & hypothesis tests.
Models, scorecards and process characterization database.
Transfer functions are a key element of DFSS

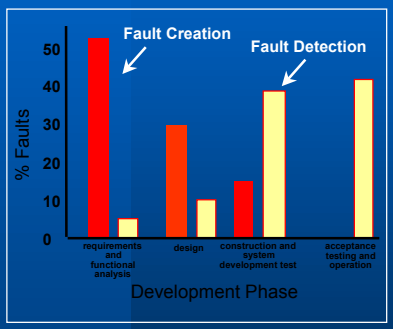

Extreme Programming
Planning Game
Functional testing
Unit testing
Refactoring
Simple Design
Collective Code Ownership
Coding Standards
Continuous Integration
On-site Customer
Forty Hour Week- Go home at 5.
Pair Programming

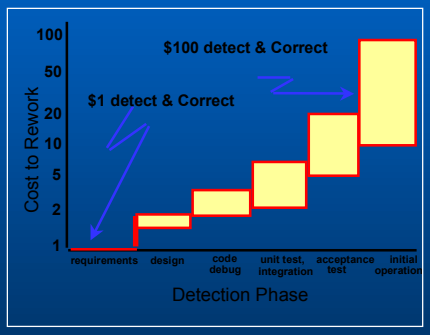

Similarity
Customer Focus
Test and Simulation
Continuous Reexamination of Knowledge
Incremental Knowledge Growth
Simple Design

Differences

- Technical / Psychological Split
  - Pair Programming
  - 40 hour week
  - Code Ownership
- Project Management
- Knowledge Framework
  - Simulation
  - Evolving Core

Impact of Defects





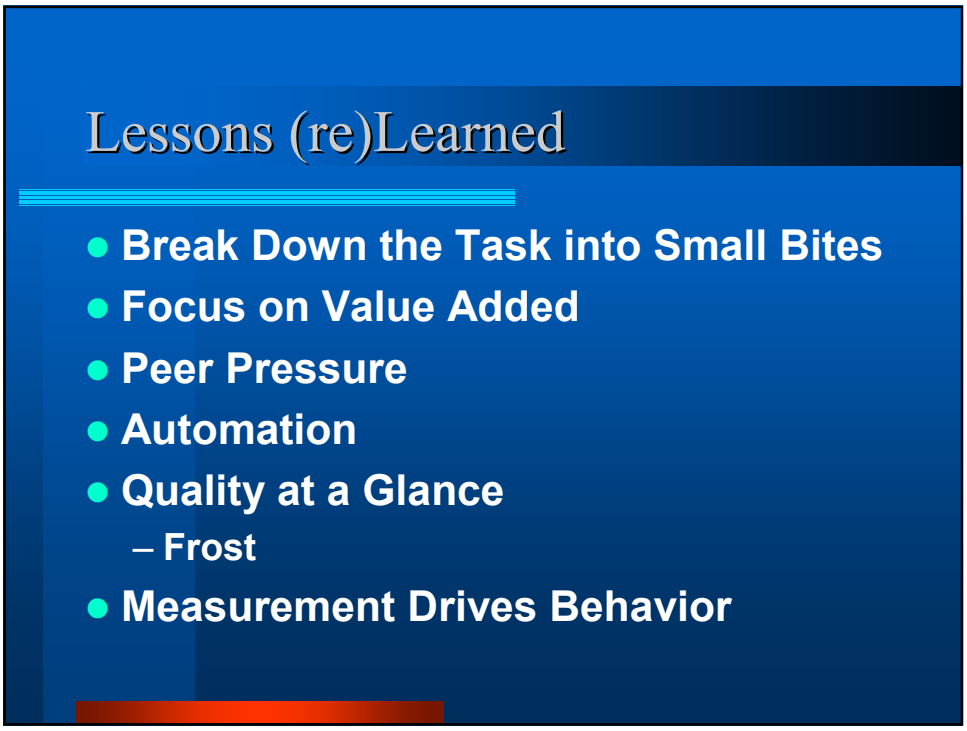
Huge Financial and Opportunity Cost

Green Belts: Common Results

- Individual Effort
- Focus on Getting "Green Belt" stamp
- Temporary Gain
- Little long term impact

A Different Outcome

- Collective Project Ownership
  - less willing to do poor work as a team
- Concentrated Effort of Many Projects
- Framework for Incremental Addition
- Quality as "The Way We Work"


Lessons (re)Learned
Break Down the Task into Small Bites
Focus on Value Added
Peer Pressure
Automation
Quality at a Glance
– Frost
Measurement Drives Behavior

Six Sigma Meets Extreme Programming

Changing the Way We Work

Timothy P. Kelliher
518-387-6691, fax 518-387-6981, kelliher@crd.ge.com

Daniel J. Blezek
518-387-5481, fax 518-387-6981, blezek@crd.ge.com

William E. Lorensen
518-387-6744, fax: 518-387-6981, lorensen@crd.ge.com

James V. Miller
518-387-4005, fax: 518-387-6981, millerjv@crd.ge.com

GE Corporate R&D
KW-C211A
1 Research Circle
Niskayuna, NY 12309

Introduction

For the past 5 years GE has been following and enhancing the Six Sigma quality model. During this same period Extreme Programming has emerged as an alternative to the traditional software development process. Although they are described in very different terms and come from different communities these two disciplines share some common threads. In this past year the authors have been following a process which draws on these common threads to create a blended process which draws on the strengths of each discipline and fit the needs of their development environment. In some cases the choices of how the development process would evolve were deliberate, in other cases chaotic forces were employed. This paper gives the background of six sigma and extreme programming, shows where they are similar, and where they differ. It then puts these thoughts into the context of day to day practice and describes their current implementation and how it has changed the way we work.

Six Sigma Basics

Behavior is a function of values. We measure what we value. Therefore it follows that to change behavior we have to change what we measure. This simple truth is the core of the Six Sigma quality program at GE. We must adapt our measurement systems to look at true life cycle costs and reward behavior that improves on these costs. This means understanding the true cost of quality. That means we must understand the impacts of decisions and process from product conception through design through development straight through to product end of life. Separate but similar disciplines have been developed to tackle each of these areas. At the product conception end *Design for Six Sigma Innovation* is used. For product design *Design for Six Sigma Product* is the applicable approach, a sub discipline with this area is *Design for Six Sigma Software.* On the manufacturing end, the basic *Six Sigma* concepts as practiced in other companies are directly applicable.

Basic Six Sigma focuses on improving existing processes and understanding the key control factors for these processes. Within GE this has been stretched far beyond manufacturing to include processes throughout the company including sales, services, and engineering. In this basic approach GE has followed Mikel Harry's Six Sigma Breakthrough Cookbook [Harry 94]. This approach breaks down any quality problem into four steps: Measure, Analyze, Improve, and Control. In the measure phase the practitioner characterizes the process in terms of Critical to Quality, CTQ, characteristics. These are the elements that the customer considers as the key factors for process success. Specific, measurable targets are established for these CTQs both for mean performance and statistical variation. In the analyze phase the CTQs are broken down to understand the factors that influence their performance. These factors are divided into two groups, key control parameters and noise parameters. In the improve phase measures of the key control parameters are further collected and studied to learn how their performance can be tuned to optimize the effected CTQ's performance. In the final phase, control, a method is developed to maintain the key control parameters at their tuned settings. Often statistical process control is used at this stage to track and maintain focus on the key control parameters. This ensures that CTQ performance will remain at its improved value.

This reliance on gathering data from actual measures is another of the fundamental aspects of the Six Sigma approach. Instead of relying on intuition to guide changes, Six Sigma demands that data be used to make decisions.

Practitioners of Six Sigma understand that there are limits to how far the Breakthrough Cookbook can go toward improving quality. There is only a finite amount of improvement that can be made to an existing process before it reaches its quality entitlement. To improve quality

beyond this point requires redesigning or changing the process so that a leap in quality can be had. This is where Designing for Six Sigma comes in.
Design for Six Sigma, DFSS, begins at the same place as basic six sigma, understanding the customer's CTQs. Once these are established, however, the two approaches diverge. Six sigma moves from this point to analyze the existing solution. In the DFSS case that solution does not yet exist. Each of the discipline areas for DFSS - Innovation, Product, Software, Commercial Quality – takes the same basic approach to developing an understanding of the CTQ's and evolving design within the context of the discipline. In place of analyzing a solution as in basic Six Sigma, models and simulations of the to be built product or process are constructed and used. These simulations range from simple monte carlo simulations to complex simulations of basic physics depending on the depth of understanding necessary to make informed design decisions.

In software modeling UML is the most prevalent modeling language. These models are employed to understand the basic structures of the solution and to transform the CTQs into a solution approach in a way that the customer can see and react to. Simulations in the software realm are most often used to understand and predict performance or human factors issues. In other disciplines MCAD or ECAD models are used to evaluate performance, Thermo and hydro dynamic performance is simulated to yield insight into the design parameters.

DFSS uses these simulations and models to predict the values of the CTQs early in the design cycle and to understand the factors that influence the CTQs. The goal of this work is to change from a reactive quality stance to a design quality stance. Throughout the design lifecycle attention to the CTQ's is not allowed to vary. The acceptable values are established up front with the customer, They are entered into a quality tracking scorecard along with the established design goals. As design begins to evolve the best estimates for the result values are filled I the scorecard and compared against the design goals. Adjustments to the design are made such that a suitable level of quality is achieved. The more information that is gathered from simulation and models the better the estimates of end product quality. As the product comes into being the estimates from the scorecard are replaced with actual values measured directly from the product. These are again compared to the design goals and the estimates. Differences are noted and corrective action taken where necessary. Where the results are different from that which was predicted we learn where to invest in improved models for the next generation.

Extreme Programming

Extreme Programming, XP, is a emerging approach to software development that focuses attention on the granularity of work elements from concept to implementation, the testing of the work products, the reduction of 'overhead' activities and the involvement of the customer in the development process. XP is successful because it emphasizes customer satisfaction and promotes teamwork. The most surprising aspect of XP is its simple rules and practices. They seem awkward and perhaps even naive at first, but soon become a welcome change to developers who adopt the XP model. Many customers enjoy being partners in the software process and developers actively contribute regardless of experience level. The rules and practices must support each other. Together they work to form a development methodology. Unproductive activities have been trimmed to reduce costs and frustration. This approach to refining the development process is in keeping with the Six Sigma tenet of behavior being driven be values and measurements. In general people value their time and as such are unwilling to spend time on 'unproductive' activities. Thus all activities that are elements of XP need to be transparently productive or they will soon fall into disuse.

XP, as defined by Kent Beck[Beck 2000], has a number of elements.

- Planning Game- Stories, lightweight use cases, are the starting point for beginning production coding.

- Functional testing- You can't continue development until the functional test scores are acceptable to the customer.
- Unit testing- You can't release until the unit tests are 100%. The unit tests enable refactoring, they drive the simple design
- Refactoring- You can't just leave duplicate or uncommunicative code around. The long term value is that reusable components emerge from this process, further speeding development.
- Simple Design- The right design for the system at any moment is the design that runs all the tests, says everything worth saying (only once), and contains the fewest possible classes and methods.
- Collective Code Ownership- If you run across some code that could be improved, you have to stop and improve it.
- Coding Standards- Everyone chooses class names and variable names in the same style. They format code in exactly the same way.
- Continuous Integration- Code additions and changes are integrated with the baseline after a few hours, a day at most.
- On-site Customer- Instead, you are in hourly contact with a customer who can resolve ambiguities, set priorities, set scope, and provide test scenarios.
- Forty Hour Week- Go home at 5. Have a nice weekend. Once or twice a year, you can work overtime for a week, but the need for a second week of overtime in a row is a clear signal that something else is wrong with the project.
- Pair Programming- This is the master feedback loop that ensures that all the other feedback loops stay in place. The pairs shift around a lot (two, three, four times a day), so any important information is soon known by everyone on the team.

Six Sigma related to Extreme Programming

From these brief descriptions it is apparent that both six sigma and XP place a great deal of emphasis on getting to know the customer and understanding the customers definition of quality. While this is certainly not a novel idea, the degree to which each of these approaches focus on the customer sets them apart from some other software quality models such as CMM. Not that the CMM ignores the customer but it stresses other, internal, aspects of the process.

The stated approach of the two disciplines is different, yet both are geared toward developing an early understanding of performance against customer CTQs and maintaining that understanding throughout the development process. The two methods differ in how they ask for practitioners to express their growing knowledge of the system. In DFSS the knowledge is expressed as model and simulations, in XP the knowledge is embedded in a growing framework of the solution. While outwardly these may seem at odds with each other, the intent in both cases is the same: capture the knowledge in a way that it can be viewed and reviewed by all of the stakeholders. Each approach puts faith in the power of exposing ideas to inspection and analysis.

Part of the apparent difference in the approaches can be bridged by reflection on testing, simulation, and modeling. When considered in the abstract a simulation is really just a test of the design and the source code in nothing more than a complete model of the solution. As the distinction between these areas is blurred, we think of all activities that evaluate our work products as a form of testing. Thus simulation are one form of test. Design reviews are another form of test. Compiling the code is still another test. Traditional regression tests are also still used. Extreme DFSS Programming holds that all of these tests should be reevaluated continuously throughout the design process. The feedback gained from knowing immediately that something has changed in the result set is empowering.

An area in which there is a large departure between XP and six sigma is the softer side of development. In the technical aspects of project development there is significant overlap of ideas between the two approaches. In the psychological aspects, however, six sigma and DFSS are essentially silent. Beyond recognizing the need for close customer interactions to define

CTQs, six sigma treats design and implementation as a purely technical task. XP goes beyond the mere technical to consider the psychology of design and how that psychology plays out in group interactions. This psychological side turns out to be at least fifty percent of the challenge to building a lasting improvement. It is a necessary, but not sufficient, condition to provide the technical support tools and guidance for quality improvement. Without addressing the human motivational side of improvement, however, the technical effort is a waste.

Changing The Way We Work

Prior to GE's thrust into six sigma our group's development process was not something we paid much attention to. The practices we followed were typical to a homegrown software effort. We had a core software product, which had been reengineered from its predecessor product, when that had grown too large and its technical foundation had become out of date. The new core product was initially developed as a demonstration of how this kind of software should operate, to serve as an example for a textbook on the subject. From there it had grown to displace the previous generation product. As versions of the textbook were produced the accompanying software also went through versions. The examples in the textbook served as the test cases for the software. These test cases would be run when we were preparing a new release. The result was that the quality of our product went up and down based on how long since the previous release. The problem with this approach was that we actually were making continuous releases to our internal customer in between the major external releases, thus their quality suffered.

Green Belt Projects

At GE every person in the company was challenged to understand and apply Six Sigma methods to their job. To back up this challenge every GE employee has been trained in Six Sigma tools and practices. Every professional within the company is expected to demonstrate use of the tools to perform their job. The demonstration takes the form of green belt projects. For people just undergoing training these are generally small projects that look to make demonstrable improvements to quality in some aspect of their work and to demonstrate proficiency with elements of the Six Sigma approach. The projects are selected by the trainee and mentored by a more experienced Six Sigma leader.

Six sigma takes a statistical look at understanding where a product or process fails to meet its CTQs. Failure is characterized as a number of defects per million opportunities for making the defect. Since six sigma is a data intensive discipline most people looking for training projects first considered where they could find some data, preferably continuous data that is easily produced in large quantities and easy to see the influence of changing parameters in the defect rate. The sort of process data that you might find in a manufacturing line that produces thousands or millions of parts per day is ideally suited. This quest left software groups at a disadvantage. The software "manufacturing" line produces single products over the course of months. Software process data comes about slowly and often is not highly repeatable from one application to the next.

To counter this the training projects for software have generally focused on some measurable aspect of the coding or testing process for which a tool exists for collecting data. Topics such as regression test coverage, memory usage, and code style are all projects that have been done for training. These were picked because they had some ability to cast the data as looking for a small number of defects within a large number of opportunities for making the defect.

The general experience with these projects has been that they do a reasonable job of defining CTQ's, e.g. all files within an application source code shall have a mean of 80% of their lines of code exercised by regression tests with a standard deviation of no more than 5%. The analysis and improve phases uniformly yield an improvement in CTQ performance, generally as the result of personal effort on the part of the trainee, e.g. they implemented a sufficient number of tests to drive the coverage up. The control phase is then where problems arise. In this phase, the trainee reports on a plan to monitor CTQ performance on a periodic basis. Having gotten their training completed the trainees then go back to working exactly as they

have in the past. The CTQ, which was often somewhat artificial is no longer being measured and as a result no longer valued. The problem is in the control step and in the initial definition of CTQ. The quality gain, once achieved, was a lone effort and remains that way in the control stage. Thus the rest of the team ends up with little concern for the gain and eventually it is forgotten.

Extreme Six Sigma Programming

Once a green belt project is successfully completed, and reported out to senior management, a trainee is noted as 'green belt trained.' Completion of a second project, presumably a little more ambitious in scope than the initial training project, leads to the designation of an individual as 'green belt certified.' At the research and development center becoming green belt certified is a condition of employment.

Many groups, and most software groups, that have gone through six sigma training end up in a similar state. The training does some good; people make small adjustments to their work habits; a few new skills are added to employee's toolkits; old habits are dress up in new vernacular; work goes on. In the case of our group, however, we took the second round of six sigma projects as a chance to change. The result is our blended "Extreme Six Sigma Programming" method of work.

At the same time as the second round of green belt projects started several members of the group began working more closely than they had in the past. This close work resulted partially from the desire to commiserate over having to do more green belt projects and partially from a desire to see a lasting improvement from the projects. The results from the first round, although temporary, had been enough to inspire some of the group to see that there was room for improving the way we worked and the quality of our output. From the short lived gains we understood that the overall quality of our software was better when we took the time and effort to check performance against the CTQs from many of the training projects. The challenge for the second round became how to put a control mechanism in place that could survive beyond the completion of the project with limited resources.

From the teams that were working closely came a secondary motivational force, if the team was going to be spending time working on something it became very important that the work would result in value. Thus the team became a quality self-enforcing unit. These forces, which were somewhat chaotic, coupled with members of the team reading and learning about XP resulted in a drive to build an extreme testing environment. XP also validated what they were experiencing – pair programming leads to better productivity.

The result of the second round project was an automated "extreme" framework for running the various tests and tools that had been built for individual green belt projects. The lessons learned from the failed control efforts guided the team to create a process for compiling the results into a dashboard that makes the results plainly visible. This framework is set up to run the tests overnight and to have the html dashboard ready in the morning for any of the developers or customers to view. Thus, in keeping with XP, our customers could now observe the state of the groups quality at any time. They could also view the quality trends over the course of a project.

Lessons Learned

From the building of the framework we learned a few lessons. There are several key elements to the success of our quality efforts. Take away any one of them and the overall result will be diminished significantly. Automation of the process is one of the keys to its success. Without the automation we would have to rely upon individual developers to run the tests. While this could work for a small set of tests over a small period of time the lessons of the first round of green belt projects showed it wasn't going to be sustained without the automation. Consolidation of the results into a single dashboard is another key. The consolidation is really just a second call for automation. Even when the tests are run and results produced automatically, without the consolidation people do not make the time to search out the results routinely and adjust as necessary to maintain the quality. Careful attention needs to be given to the consolidation. Even today, with the system having been in

place for two years, those quality items that are most prominent on the quality dashboard get the most attention. Those which appear as more of a footnote are often neglected. This behavior relates back to the basic six sigma principle; we value what we measure. In this case we value more that for which we highlight the measurement.

Third, and perhaps the most important lesson, is team buy-in and ownership. Not everybody has to 100% on board but there have to be some of the alpha-developers for whom this becomes a way of working. From their influence comes the change in the way the rest of the group behaves.

Conclusions

As a group of workers that has gone through green belt training began to adopt elements of extreme programming an interesting phenomenon occurred. The group has a five year history of developing an open source package so they had been early adopters of elements of the approach. The idea of collective code ownership, refactoring, simple design, and coding standards were all part of the group ethic to a greater or lesser degree. One of the new elements of extreme programming that the group decided to experiment with as it was embarking on Six Sigma was a modified form of pair programming. This has made all the difference in the impact made by Six Sigma.

In the local version of pair programming the concept is extended to be paired, or trippled, work. Not only is time programming spent together but other work time is also spent together. This includes time spent on green belt projects. The time being jointly directed as in pair programming. This means that each member of the work group ends up having to buy into the value of the green belt project before it begins. Given this expanded ownership of not just the code output but also of the quality outputs the control phase got a renewed emphasis. With additional owners comes additional pressure to achieve a suitable performance on the CTQ. After the first round of projects the CTQ performance had risen noticeably and any backsliding was more rapidly noticed. The group then moved to the next element of extreme programming, continuous integration. Measuring the quality results entailed running a test of sorts for each aspect of quality that was being checked. Running these tests by hand became cumbersome so the group developed a nightly test harness that runs each of the quality tests and collects all of the data to present a single comprehensive view of measured quality. It was a small move from this point to start a continuous test process that runs a slimmed down set of the quality tests whenever code is checked into the source code repository.

The continuous measure of quality then is the final blending of extreme programming with Six Sigma. Behavior and measurement are well aligned. System performance against external, customer defined CTQs, expressed through regression tests, and internal, team defined CTQs, expressed through quality tests are available on a continuous basis. The teams are not able to move forward unless the results of their work is a clean dashboard as measured by both continuous and nightly dashboards. Extreme Six Sigma has become the way we work.

Reflections

We did not set out to change the way we worked. In fact we were pretty happy with the state of our developments. Our group was seen as one of the exemplar groups for software development within our larger organization. Why then did we change? Once we were forced to do some self-examination it became clear how much better things could be. We learned from the green belt projects and came to see the worth of six sigma, even if it was, and is, a top down initiative. At the same time we did not take the whole package. Parts of six sigma and DFSS are still not a part of our daily routine. The parts we have taken, however, are now deeply ingrained in our process.

XP had an advantage to adoption. It did not come with a top down mandate. It did agree with some of our intuition and some of what we were being told to do. XP's testing fit well with the testing we had arrived at through six sigma. Other parts of XP, likewise, fit with our needs. As with six sigma, we have not taken the whole XP package, instead picking the elements that work for us. As we learn perhaps we will understand the benefit to some of the elements we

have not yet chosen to try. We can say, however, that as a result of our experiences we have changed the way we work and none of us will go back to a less results conscious mode of work even as we enter new project areas.

[Harry 1994] Harry, Mikel J.,*The Vision of Six Sigma:Tools and Methods for Breakthrough*, Sigma Publishing Company, 1994.

[Beck 2000] Beck, Kent, *Extreme programming explained: embrace change*, Addison-Wesley, 2000

Author Biographies

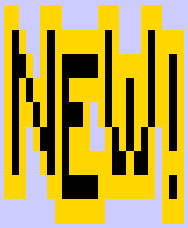
Timothy P. Kelliher is a Computer Scientist at GE's Corporate Research and Development Center in Schenectady, NY. He has over 15 years experience in systems and software engineering. At the center he has worked on Software Engineering CASE tools, Human Computer Interaction, and Software Quality systems. He is a Six Sigma Master Black Belt and spends much of his time instructing and mentoring the GE software development community. He is co-author of “Engineering Complex Systems with Models and Objects” published by McGraw-Hill, 1997.

Blezek is a Computer Scientist in the Visualization and Computer Vision Program at GE's Corporate Research and Development center. He holds a Ph.D. from the Mayo Graduate School in Biomedical Engineering. His research interests include medical image segmentation, advanced rendering algorithms, and practical software quality techniques

William Lorensen is a Graphics Engineer at GE's Corporate Research and Development Center in Schenectady, NY. He has over 30 years of experience in computer graphics and software engineering. William is currently working on algorithms for 3D medical graphics and scientific visualization. William is the author or co-author of over 60 technical articles on topics ranging from finite element pre and postprocessing, 3D medical imaging, computer animation and object-oriented design. He is a co-author of "Object-Oriented Modeling and Design" published by Prentice Hall, 1991. He is also a co-author of "The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics" published by Prentice Hall in November 1997. Mr. Lorensen holds twenty seven US Patents on medical and visualization algorithms.

James Miller is a Computer Scientist at GE's Corporate Research and Development Center in Schenectady, NY. He joined GE after receiving his PhD from Rensselaer Polytechnic Institute in 1997. His thesis topic was in Computer Vision. At GE James has become a primary contributor to vtk software algorithm development and testing. For the past year, he has been the project leader on a research project for Lockheed Martin involving the inspection of large airframes using laser ultrasonic techniques.

QW2001 Paper 3M2



Ms. Elli Georgiadou (Middlesex University)
&
Ms. Naomi Barbor (University of North London)

Investigating The Applicability Of The Taguchi Method To Software Development

Key Points

- Quality Engineering and Experimental Methods
- Taguchi made simple through novel vizualisations
- Applicability of Taguchi Method to Software Development

Presentation Abstract

The purpose of this paper is to investigate the possibility of applying the Taguchi Method to software production. It is well recognized that we need to ensure the quality of the end product early in the software life cycle. Attempts to introduce quality at the later stages increase cost. Dr. Taguchi's philosophy is now well practiced in the manufacturing industry. In Japan the Taguchi Method is called 'hinshitsu kougaku'. It literally means 'quality engineering'. The method has ensured the significant reduction of manufacturing costs together with increased product quality. In this paper we present a suite of visual representations of the major components of the Taguchi method. These visualisations aid the understanding of both the Taguchi's philosophy and the techniques. The investigation concludes with a set of guidelines for improving software quality through statistical analysis methods, which are practiced in the Taguchi Method.

About the Author

Elli Georgiadou is a Principal Lecturer in Software Engineering at Middlesex University, London. Her teaching includes Software Metrics, Methodologies, CASE and Project Management. She is engaged in research in Software Measurement for Product and Process Improvement, Methodologies, Metamodelling and Software Quality Management. She has extensive experience in academia and industry and has been active in organising conferences and workshops under the auspices of the British Computer Society and the ACM British Chapter.

Investigating the Applicability of the Taguchi Method to Software Development

NAOMI BARBOR[1] and ELLI GEORGIADOU[2]
n.barbor@unl.ac.uk, e.georgiadou@mdx.ac.uk

[1]*School of Informatics and Multimedia Technology, University of North London, 2-16 Eden Grove, London N7 8EA*, UK

2*School of Computing Science, Middlesex University*
Trent Perk Campus, Bramley Rd, London N14 4YZ, UK

Agenda

- **The need to ensure quality**
- **Quality and Experiments in Software Engineering**
- **The Taguchi Method**
- **Software Development and Taguchi**
- **Guidelines for Applying the Taguchi Method in Software Development**
- **Conclusions & Future Research**
- **Acknowledgements**
- **Q& A + Contact Details**

The need to ensure quality

- Safety / Reliability ?
- High Performance ?
- Cost-effectiveness ?
- User satisfaction ?

Software Quality is….

- Fenton (Fenton *et al*, 1995) states that software quality is "The totality of features and characteristics of the software product that bear on its ability to satisfy stated or implied needs ".

Software Quality is….

- 'ISO9126-Software Product Evaluation: Quality Characteristics and Guidance for their Use' is the first international standard to attempt to define a framework for evaluating software quality (Azuma, 1993).

Software Quality is….

- *Reliability : The software should maintain its level of performance under stated conditions for a stated period of time6*
- *Efficiency:* The software should provide a solution to the problem in an efficient (time, accuracy) manner.
- *Usability:* The software should be easy to use.
- *Maintainability:* The software should be easily maintained after its shipping.
- *Portability*: The software should have the capability to be transferred from one environment to another.

Quality and Experiments in Software Engineering

- Traditional sciences have always recognised and used formal, controlled experiments for testing hypotheses.
- However, in software engineering very few controlled experiments have been carried out (Law *et al.,* 1992; Basili *et al*., 1984; Card *et al*., 1987; QUANTUM, 1992; Georgiadou *et al*., 1993; Georgiadou *et al*., 1994; Shepperd *et a*l., 1997; Georgiadou e*t al*., 1999 & 2001) to examine specific aspects of quality such as complexity. understandablity and maintainability.

Laboratory Experiments (from Galliers)

- - are designed to be precise
- - provide clear distinction between variables
- - use quantitative analytical techniques
- - support a view for generalisation
- Strengths:
  - keep control of a few variables which may be studied intensively later
- Weaknesses:
  - they are often very simplified;
  - what about real life?

Four problem areas

- Schack (Schack, 1987) identified four problem areas facing formal experimentation in Software Engineering namely:
  - (1) the prohibitive costs,
  - (2) the difficulties in controlling differences in developers' and users' ability,
  - (3) the effects of development methods and tools used in part of the development process on other parts, and
  - (4) the evolutionary nature of software development environments.

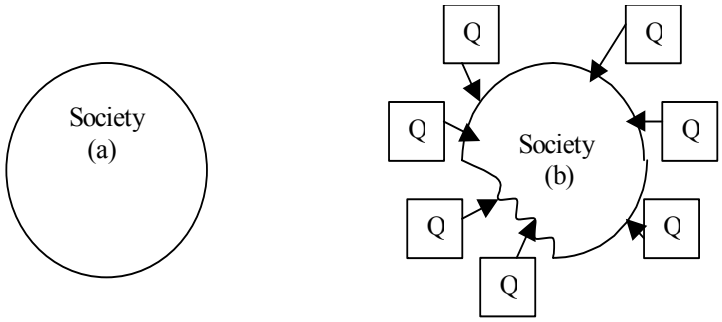
The Taguchi Method

Dr. Genichi Taguchi is director of the Japanese Academy of Quality. He is credited with having started the Robust Design movement in Japan more than 30 years ago. Dr. Taguchi's philosophy began taking shape in the early 1950s when he was recruited to help correct postwar Japan's crippled telephone system. Finding deficiencies in traditional trial-and-error approaches to identifying design problems, he eventually developed his own complete, integrated methodology for designing experiments (American Supplier Institute, 1999).

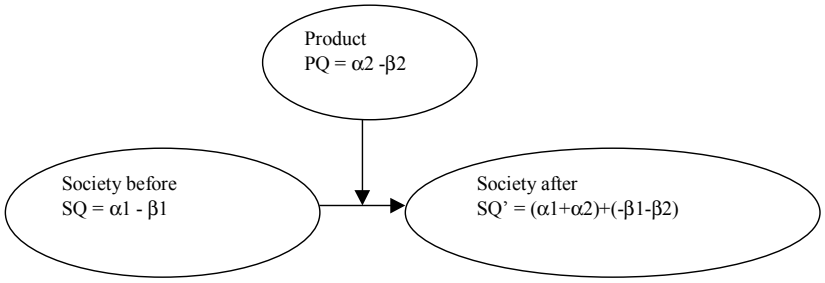
A new view of Quality

- "If quality is high, our society will get benefit from the product. If the quality is low, our society's current standard will decrease to cope with those bad products. That is, the smaller the loss, the higher the desirability" (Baba, 1999). The term 'social loss' implies:
- .losses due to poor and varied performance of a product;
- .failure to meet the customer's requirements of fitness for use or for prompt delivery;
- .harmful side-effects caused by the product.

Visualization of the Taguchi's Philosophy



Taguchi's philosophy -2



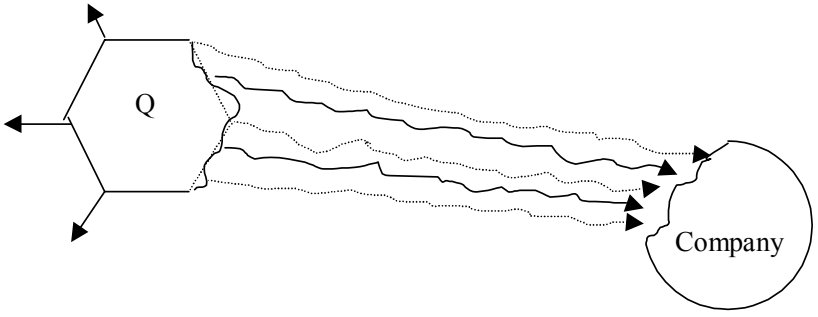
Robust Design

- Robust Design is an important methodology for improving product manufacturability and life span, and for increasing the stability of the manufacturing process. In Taguchi's philosophy, the use of experimental design is critical. This experimental design aims to minimize the *variability* of a product so that the quality of a product does not vary unpredictably

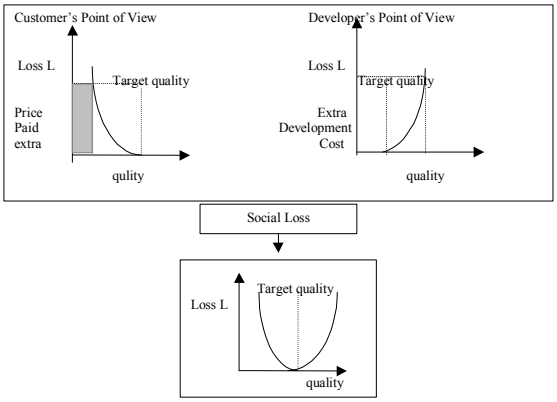
Loss Function

- In the Taguchi Method it is believed that the variability of a product quality results in financial loss. To assess the amount of loss in Taguchi's quality definition, the *Loss Function* was suggested.

Visual representation of the relationship of product quality and loss function

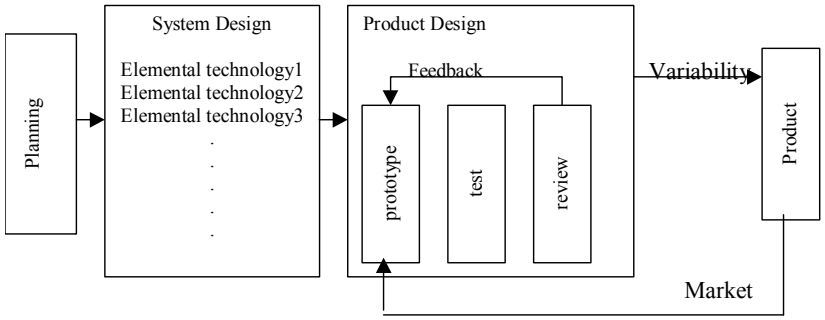


Loss Function-2

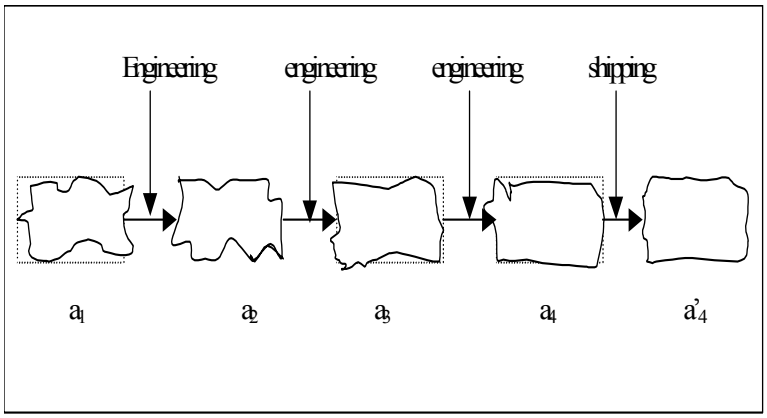


Manufacturing Process

[adapted from website-2]

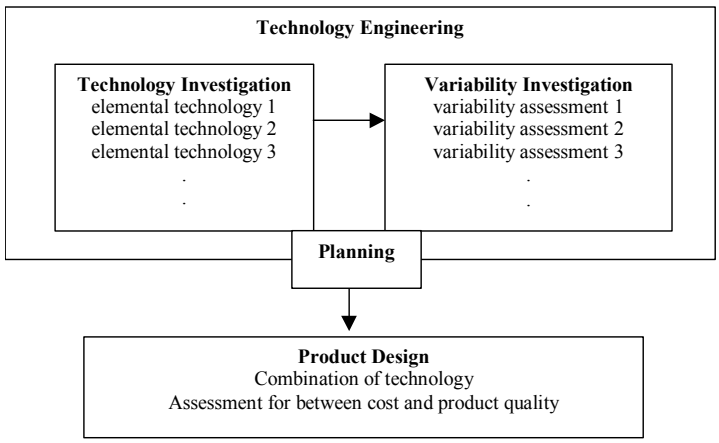


Diagrammatic representation of Target Quality Value



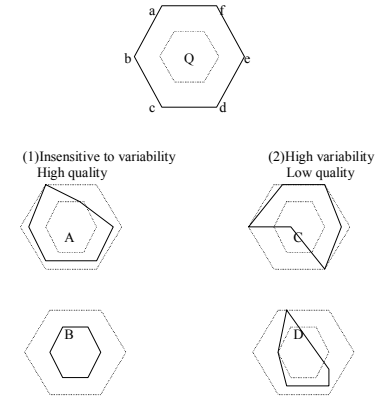
Off-line Process Control

[adapted from website-2]

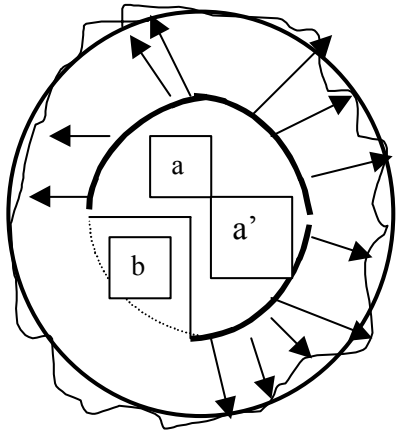


Product Quality

Cultural, Legal Shape of the Problem



Noise Factors and their Effect on Quality



Software Development and Taguchi

- We need to conduct experiments before actually producing code since Taguchi's experimental design and experiments are set before production. However, in software development it is not realistic to do this. Therefore for each project the guidance of coding process may be suggested such as " The target number of lines of code per module is less than 150".

Significant features/parameters

- According to Adrian Burr (Burr *et al*., 1996) the number of lines of code per module, target complexity, McCabe's cyclomatic complexity and the number of parameters are considered to have significance for software quality. These factors can be used in designing experiments.

Design parameters in the software development environment

- Machine, Operating System and Languages
- The efficiency of a software product such as execution time has high priority as a software quality factor. To maximize the performance of a product, the choice of machines, operating systems and programming language has to be included in the list of parameters.

Human Factors

- We must mention an additional design parameter which the Taguchi Method does not mention explicitly in the Robust Design. That is, performance variability in a human being such as his/her experience and communication skills needed in a software development team. The developers' performance has an effect on producing quality software products in a similar way to the effect of machines etc.

Orthogonal arrays

- It is important to choose the design parameters which are independent of each other.
- Orthogonal arrays are used to record information about the design of the experiment. The advantage of using an orthogonal array is that the effect of several parameters can be determined whilst also minimizing the number of experiments.

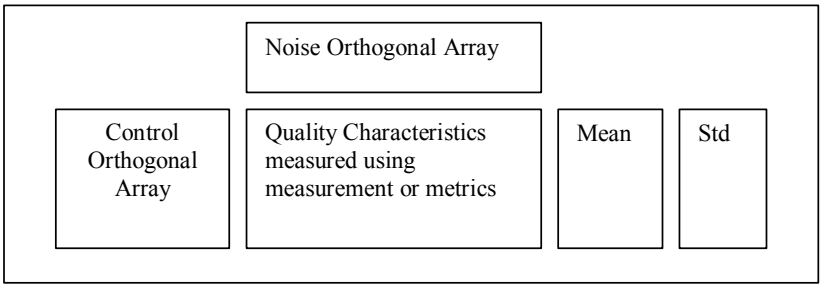
Selection of Parameters and their settings

- The following list contains some of the widely-used orthogonal arrays which use two and three level design parameters. L18 is most commonly used in the Taguchi Method.
  - L4: Accommodates three two-level design variables
  - L8: Accommodates seven two-level design variables
  - L9: Accommodates four three-level design variables
  - L12: Accommodates eleven two-level design variables
  - L18: Accommodates one two-level and seven three-level design variables

An illustrative example

Design Parameters		Settings	Comment
A	Lines of code per module	(50, 150, unspecified)	This setting is based on suggestions by Adrian Burr who and information found in Beizer (Beizer, 1990) and Georgiadou (Georgiadou, 1993).
B	Nesting Level according to the design	(3, 5, 7)	Cantata indicates McCabe's complexity 1 to 10 is 'pass' and greater than this results in 'fail'. At the design stage the flow diagram or Pseudo Code can be used to indicate the nesting level before coding.
C	Number of statements per module	(30, 40, 50)	It is difficult to control this parameter but it is not impossible. As is commonly known, the larger the number of statements in a module the harder it is to achieve 100% test coverage because of the cost involved.

Matrix Experiment



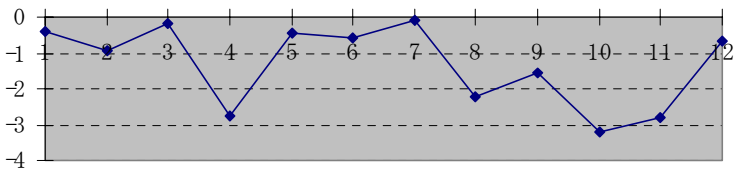
Quality characteristics and the S/N Ratio

- The traditional Experimental Design uses S/N Ratio to observe the variance of data. The Taguchi Method aims to analyse S/N Ratio, and to use it to reduce variability of the final product. An example is shown to demonstrate variability of possible design parameters' in a software product followed by suggested quality characteristics in three categories using three equations.

Static Analysis, Metrics and Taguchi

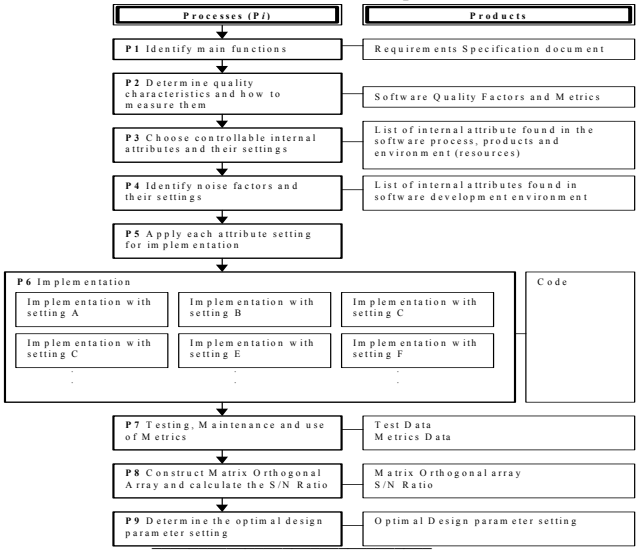
- Eight GNU applications were downloaded from the internet (GNU, 1984) and analysed. Cantata (IPL, 1994) instrumented these files, calculated a number of metrics and produced files which gave the static analysis of each module in each application. These data was used to apply the Loss Function in order to see the variability of the listed measurements for the modules (Barbor, 1999).

Variability



The numbers labeled for each dot in the graph are:
1: Expression Statements
2: For Loop Statements
3: While Loop Statements
4: Do Loop Statements
5: If Statements
6: Switch Statements
7: Return Statements
8: GOTO Statements
9: Break Statements
10: Continue Statements
11: Null Statements
12: INT Statements

Guidelines for Applying the Taguchi Method in Software Development



Difficulties

- The difference between mass-production and the production of a unique piece of software, the importance of customer evaluation, and unavoidable changes of requirements at any stage of software life cycle make it difficult to apply the Taguchi Method as it stands to software development. However, the authors believe that the underlying concept of Robust Design for product quality and minimization of production cost can be applied in software development since it enhances the probability of controlling software production towards high quality software.

More difficulties

- Difficulties are also experienced in carrying out the experiment since certain information that is needed when applying the Taguchi Method is often difficult and sometimes impossible to obtain. This includes in-house information such as time and cost of product development, the company's policy or guidelines for producing the code (if these exist), the environment such as the machine used or experience of the programmers.

Conclusions -1

- The purpose of this paper was to investigate the possibility of adapting the Taguchi Method for Software Quality Improvements.
- A number of suggestions are made to adapt the Taguchi Method from ensuring quality in manufactured products to the development of quality software.
- Adjustments are necessary because of the fundamental differences between tangible manufactured products and software artifacts.

Conclusions -2

- Taguchi's philosophy "The better the quality, the less the production cost" in the manufacturing industry is equally valid in the software industry because quality software must also have a good design to enable a software company to minimise the cost of repetitive development processes (redesigning, re-coding and re-testing) and the severity of testing.

Problems

- It is not straightforward to conduct the Robust Design in software development mainly because software products are intangible. A problem specific to design and execute experiments was that there were vast numbers of design factors for which it was not possible to obtain values such as cost and time spent, the methodology used, the experience of the developers, types of machines, the office environments, and whether the programmers have developed a similar type of product before or not.

Software Metrics & Taguchi

- The use of metrics is particularly useful in software development if what needs to be done is clearly identified and when and how statistical analysis activities should be carried out in the software development procedure is established. The authors believe that product variability can be measured, monitored and controlled with a high degree of confidence if the Taguchi Method is applied to software production.

Further work

- Further work will initially concentrate on using the proposed guidelines for carrying out a number of experiments in order to validate the adapted model. Secondly, we will work on identifying a correct and unambiguous set of controllable design parameters which have significance for software quality employing empirical data and its analysis under a systematic measuring operation.
- Industrial/Academic Funding required

Acknowledgements

- The authors would like to thank Adrian Burr for the origin of the idea and his practical help. Sincere thanks are also due to Aasma Saadia for her incisive and useful comments, corrections and suggestions for the completion of this paper. Finally many thanks to IPL for providing the Cantata tool free of charge.

How you can contact us

- Naomi Barbor
- *School of Informatics and Multimedia Technology, University of North London,*
- *2-16 Eden Grove,*
- *London N7 8EA, UK*

- **Tel: +44 207 607 2789**
- **Fax: +44 207 753 7009**
- **email: n.barbor@unl.ac.uk**

- Elli Georgiadou
- *School of Computing Science, Middlesex University Trent Perk Campus,*
- *Bramley Rd,*
- *London N14 4YZ, UK*

- **Tel: +44 208 411 4331**
- **Fax: +44 208 411 5924**
- **email: e.georgiadou@mdx. Ac.uk**

Investigating the applicability of the Taguchi Method to Software Development

NAOMI BARBOR and ELLI GEORGIADOU

School of Informatics and Multimedia Technology, University of North London, 2-16 Eden Grove, London N7 8EA

Tel: +44 171 753 3142 Fax: +44 171 753 7009

email: barborn@unl.ac.uk / e.georgiadou@unl.ac.uk

The purpose of this paper is to investigate the possibility of applying the Taguchi Method to software production. It is well recognized that we need to ensure the quality of the end product early in the software life cycle. Attempts to introduce quality at the later stages increase cost. Dr. Taguchi's philosophy is now well practiced in the manufacturing industry. In Japan the Taguchi Method is called 'hinshitsu kougaku'. It literally means 'quality engineering'. The method has ensured the significant reduction of manufacturing costs together with increased product quality. In this paper we present a suite of visual representations of the major components of the Taguchi method. These visualisations aid the understanding of both the Taguchi's philosophy and the techniques. The investigation concludes with a set of guidelines for improving software quality through statistical analysis methods, which are practiced in the Taguchi Method.

Investigating the applicability of the Taguchi Method to Software Development

The purpose of this paper is to investigate the possibility of applying the Taguchi Method to software production. It is well recognized that we need to ensure the quality of the end product early in the software life cycle. Attempts to introduce quality at the later stages increase cost. Dr. Taguchi's philosophy is now well practiced in the manufacturing industry. In Japan the Taguchi Method is called 'hinshitsu kougaku'. It literally means 'quality engineering'. The method has ensured the significant reduction of manufacturing costs together with increased product quality. In this paper we present a suite of visual representations of the major components of the Taguchi method. These visualisations aid the understanding of both the Taguchi's philosophy and the techniques. The investigation concludes with a set of guidelines for improving software quality through statistical analysis methods, which are practiced in the Taguchi Method.

1. Introduction

In modern life, a computer has become one of the essential appliances like a telephone or a television. The advantages of using computers such as accuracy and time efficiency are now well recognized, and so is the demand for software which has high performance and achieves user satisfaction. Here the word 'satisfaction' might be affirmation that money was well spent at the consumer side and 'profit' will be the term for it for software developers. Statistical Process Control (SPC) has been well practiced in the manufacturing industry. In the Japanese industry SPC has been widely applied resulting in the success of manufacturing technological products. The big names such as Sony, Sharp or Pioneer have gained good reputation for their products from customers all over the world. It is true that the economic crisis in 1998 has brought the Japanese industry financial decline. The Japanese Yen has become cheep in the financial market and manufacturing products costs more than before because mainly the industry and fundamentally the social life itself depend on import materials from all over the world. If products become more expensive, the only way to maintain possible markets might be to produce higher quality products using cheap materials which may have high variability. For this reason Japanese companies make use of Statistics to control the quality of products.

In the software industry, the Deming or Shewhart's quality improvement cycles or the W model have led to a systematic software process which increases the quality of the end product. However, traditionally the quality of software depended on whether it was produced by highly knowledgeable, skilled programmers or not. In such cases, there will normally be variability in the end products. Here, the adoption of SPC is a new weapon for ensuring the quality of software. It is desirable to reduce product variability early in the software lifecycle, and to minimise the cost of production.

Dr. Genichi Taguchi's Experimental Design incorporates the collection of statistics as a fundamental activity in the manufacturing process. To conduct these experiments, it is necessary to fully understand the software development procedures, where all the design parameters required in the operation of the Taguchi Method originate from, and the measurements proposed to measure those design attributes and variables.

Even though the Taguchi Method has been popular throughout the world, there are criticisms. Greenfield (Greenfield, 1995), a statistician, warns that the adoption of the Taguchi Method itself does not ensure successful Quality Improvements. He argues that the followers of the method are in danger of only fulfilling the procedure of the experiments without fully understanding or investigating the vast number of response variables to a product quality.

The choice of the required design parameters and their correct leveling is the key of the Taguchi Method. Even if the improved design suggests the improvement of product quality, the setting of these quality factors may be wrong unless the designers fully understand what they are doing.

There are difficulties to adopting the method into software production because of the difference in the nature of the software artifacts. For example the manufacturing industry tries to maximise the profit by selling more while minimizing the production cost. On the other hand, a software company might have two situations. One is producing one product and selling a lot so that profit from warranty increases such as in computer game production. The other is when a single customer wants a specific single product which is unique to the customer's need, and often the required system is something new. In the worst case if that single software product does not meet the customer's requirements, the loss is unrecoverable. At least the cost of overtime until the final product is satisfactory to the customer will bring the company financial loss.

Furthermore there are differences in the nature of materials for use. In manufacturing physical sciences are the key to analyzing the product performance. The quality of the product is good if all the input energy, such as gas, electricity or fuel, is converted to the product function consistently under a wide range of environments. However in the software industry, materials equate to logic, and during the integration process, there has not been a formal method for ensuring that the product is bug free.

2. Experiments in Software Engineering

The final goal of Software Quality Assurance is producing 100% bug free software. However, it is commonly recognised that so far it has been impossible to produce bug free software. According to Beizer (Beizer, 1990), it is impossible to test software thoroughly. In addition, there might be the possibility of invisible bugs in software. Therefore it seems practical that instead of trying to test software as much as possible we tackle those parts of the software which are more likely to contain bugs. The next question is which parts of a system we should look into. Through experience they would be the parts which handle more variables, have more function and communicate with the other modules more. So if we can identify those parts and concentrate on making them bug-free, it can be said that we increase the overall quality of the software.

Fenton (Fenton *et al*, 1995) suggests that software quality is "The totality of features and characteristics of software product that bear on its ability to satisfy stated or implied needs ".

ISO9126-Software product evaluation : quality characteristics and guidance for their use is the first international standard to attempt to define a framework for evaluating software quality (Azuma, 1993).

The standard specifies six characteristics for evaluating software quality:

Functionality: The software should meet all the user's requirements

Reliability: The software should maintain its level of performance under stated condition for a stated period of time.

Efficiency: The software should provide a solution to the problem in an efficient (time, accuracy) manner.

Usability: The software should be easy to use.

Maintainability: The software should be easily maintained after its shipping.

Portability: The software should have the capability to be transferred from one environment to another.

2.1 *Formal Experiments in Software Engineering*

Traditional sciences have always recognised and used formal, controlled experiments for testing hypotheses. However, only a few controlled experiments have been carried out in Software Engineering (Law *et al.,* 1992; Basil *et al*., 1984; Card *et al*., 1987; QUANTUM, 1992; Georgiadou *et al*., 1993; Georgiadou *et al*., 1994; Shepperd *et a*l., 1997; Georgiadou e*t al*., 1999) examining specific aspects of the software product or process.

Hetzel (Hetzel, 1993) warns that " …a few experiments have produced important results, their overall legacy and influence on software practice has been limited. One reason is inherent with the experimental approach. No one is willing to fund multiple repeated development of complex systems just so that method and individual differences can be systematically controlled and analysed. Consequently most experimental results have been obtained on very small problems in artificial

environments and practitioners and managers are properly unconvinced that the results will scale up and apply to 'real' world work ".

Schack (Schach, 1987) identified four problem areas facing formal experimentation in Software Engineering namely the prohibitive costs, the difficulties in controlling differences in developers' and users' ability, the effects of development methods and tools used in part of the development process on other parts, and the evolutionary nature of software development environments.

Planning, designing and executing an experiment within an academic environment avoids the problem of cost due to the availability of physical and human resources. Additionally, the ability, prior knowledge and skill of the experimental subjects can be controlled. Galliers (Galliers, 1992) summarises the key features, strengths and weaknesses of the laboratory experiment as shown in table 1.

Table 1. Laboratory Experiments

Research Approach	**Key Features**	**Strengths**	**Weaknesses**
laboratory experiments	- designed to be precise - provide clear distinction between variables - use quantitative analytical techniques - support a view for generalisation	keep control of a few variables which may be studied intensively later	very simplified; what about real life?

For the evaluation of the Graphical Query Language GOQL (Keramopoulos, 1997) the formal laboratory based controlled experiment was selected because it was possible to resource the activity in terms of available laboratories, experimental subjects and time for designing and executing the experiment.

The authors believe that product variability can be measured, monitored and controlled with a high degree of confidence if the Taguchi Method is applied to software production.

2.1 *Importance of Software Measurement*

Fenton and Pfleeger (Fenton *et al*., 1997) provide the following definition of measurement:

"Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterise them according to clearly defined rules. The numeral assignment is called the measure."

The theory provides the rigorous framework for determining when a proposed measure characterises an attribute and provides rules for determining what statistical analysis are relevant and meaningful.

To understand the definition of measurement in the software context, we need to identify the relevant entities and the attributes of those that we are interested in characterising numerically.

2. 2 *Internal attributes and external attributes*

According to Fenton (Fenton, 1994) internal attributes are the key to improving software quality and can be measured in terms of the code. Software engineering methods provide rules, tools and heuristics for producing software products. They show how to provide structure in both development process and the products themselves such as documents and code which have properties (internal attributes). These properties are modularity, re-use, coupling, cohesiveness, redundancy, D-structuredness and hierarchy. They assure reliability, maintainability and usability for users and also assure productivity and cost-effectiveness for managers.

Brooks (Brooks, 1995) states that a good top-down design avoiding bugs can be achieved in the four ways listed below.

1. The clarity of structure and presentation of a design make it easier to understand

the precise statement of requirements and functions of the modules.

2. The partitioning and independence of modules helps to avoid system bugs.
3. The suppression of detail makes flows in the structure more apparent.
4. Testing can be easier because the proper level of detail will be shown at each step.

Why might we be especially interested in measurements for early life-cycle products?

Because we would like to predict attributes of the eventual implemented system such as cost, effort, size, complexity and quality. Complexity means the totality of all internal attributes and we aim to control it in software products.

We are interested in both internal and external attributes because for example reliability of a program is dependent not just on the program itself, but on the compiler, machine and user. And productivity is dependent on people and management of a project.

It is often necessary to use surrogate measures for complex external attributes (Fenton *et al*., 1996; Kitchenham, 1996). For example time taken to carry out specified maintenance tasks might be used to provide an indication of the maintainability of software (Georgiadou *et al*., 1994).

2.3 *Software Quality Assurance (SQA) Metrics*

The use of software quality metrics within an organisation or project is expected to have a beneficial effect by making software quality more visible. Data collection methods and the idea of validation of metrics are employed. A software quality survey gives some information as to the state of practice with software quality assurance metrics. Metrics gives some help to the inaccurate area of software estimation. All metrics are not an SQA measure, however they deserve some special notice for overall software success.

There are a number of examples given of various practical implementations of SQA metrics. For example, Hitachi's quality measurement was described by Tajima (Tajima *et al*., 1981). Tajima described 'quality' improvements at Hitachi in terms of spoilage, being the time to fix post-release defects.

What choice of a set of data is an important and difficult task. As Fenton (Fenton et al., 1997) and Shepperd (Shepperd, 1995) emphasizes that it is meaningless to collect figures without purpose We are thus interested in the question which property (attribute) of software has significance to the quality of the end product. Realistically to achieve this aim, it would be necessary to collect data empirically. However it is useful to develop the skill to examine the data and understand the program's constructs and complexity. That will help the programming capability of a programmer.

3. The Taguchi Method

3.1 *Japanese quality control*

According to Logothetis (Logothetis, 1989), the Japanese approach for quality control was founded by W. A. Shewhart who is the founder of modern quality control. His philosophy is based on the motto "the better the quality, the lower the cost".

Logothetis also comments that Kaoru Ishikawa is the father of 'Total Quality Control' and received the 'Deming Prize' for the philosophy of:

1. Statistics becoming a common language which can be used at all levels in the organization and provide the information to anticipate, identify and correct mistakes.
2. The purpose is to reduce wasteful variability in the system by 'doing it right the

first time'.

Hagime Karatsu (Logothetis, 1989) explains the manufacturing process as follows;

"If it is aimed to produce quality products, there will be great financial benefits. Withdrawal and return of products are reduced. Higher productivity will be achieved because it will be less frequent to stop machines for replacing materials. That means it is possible to reduce the operation rate. As the manufacturing system itself improves in quality, the cost will be minimized. That will give rise to the company's reputation and will increase its sales."

Dr. W. E. Deming was Chairman and President of Nashua Corporation and he is regarded as the founder of the third wave of the Industrial Revolution. He claimed that if a company tries to obtain shorter term profit, it would lead to business failure. Deming suggested ceasing dependence on inspection in order to achieve quality, and eliminating the need for mass inspection by building quality into the product in the first place. This emphasizes the importance of the stage of design of a product, which is common from Deming to Taguchi.

3.2 *Taguchi's philosophy*

Dr. Genichi Taguchi is director of the Japanese Academy of Quality. He is credited for starting the Robust Design movement in Japan more than 30 years ago. Dr. Taguchi's philosophy began taking shape in the early 1950s when he was recruited to help correct postwar Japan's crippled telephone system. Finding deficiencies in traditional trial-and-error approaches to identifying design problems, he eventually developed his own complete, integrated methodology for designing experiments (American Supplier Institute, 1999)

3.3 *Definition of Quality*

"If quality is high, our society will get benefit from the product. If the quality is low, our society's current standard will decrease to cope with those bad products. That is 'the smaller the loss, the higher the desirability" (Baba, 1999). The term 'social loss' implies:

1. losses due to poor and varied performance of a product;
2. failure to meet the customer's requirements of fitness for use or for prompt delivery;
3. harmful side-effects caused by the product.

3.4 *Visualization of the Taguchi's philosophy*

In this investigation we propose the visualization described in Fig 1 and 2 of Taguchi's philosophy.

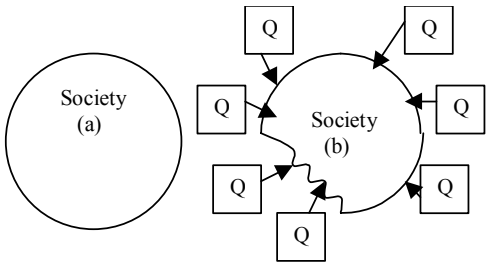


Fig. 1. Diagramatic represention of Taguchi's Philosophy - 1

Here the ideal society shape is represented by a perfect circle. In the Taguchi method, the variability of product quality Q causes social loss. Society (b) cannot remain the shape of the circle. This affects not only the customer, but also the companies which supplied these products. The loss should be minimised. Thus low loss is emphasised to mean high quality in society.

Fig 2 demonstrates the effect of product quality on society. We represent Taguchi's social philosophy by using plus and minus symbols. Quality includes the idea of the loss caused to society as a result of poor quality. "Quality-loss'' of a product has an effect on anything which involves and reduces the quality at the other end. This is equivalent to the Physics principle of energy consumption where energy moves from higher to lower levels.

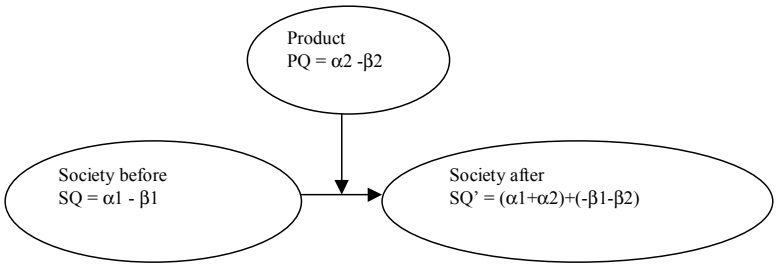


Fig. 2. Taguchi's Philosophy - 2

The current value (SQ) in society is altered by the value (α for positive and/or β for negative) of the product (PQ) after its shipping as shown in formula (1). The assumptions are made that society already contains loss (SQ=α1-β) and Product is not perfect (PQ=α2-β2).

$$\sum(Societyaftervalue\mathrm{SQ}') = \sum(societybeforevalue\mathrm{SQ}) + \sum(productvalue\mathrm{PQ}) \quad (1)$$

3.5 *Robust Design and Loss Function*

Robust Design is an important methodology for improving product manufacturability and life span, and for increasing the stability of the manufacturing process. In Taguchi's philosophy, the use of experimental design is critical. This experimental design aims to minimize the variability of a product so that the quality of a product should not vary unpredictably. In other words, the Robust Design is used to ensure the

performance of a product is steady. The use of experimental design in off-line quality improvement requests active use of scientific method and statistics. Since its introduction in 1980, Taguchi's approach to quality engineering and robust design has received much attention from designers, manufacturers, statisticians and quality professionals.

The central aim is minimizing the *variability* of a product. The quality of product should not be variable unpredictably. It is important to ensure the performance of a product is steady.

In the Taguchi Method it is believed that the variability of a product quality results in financial loss. To assess the amount of loss in Taguchi's quality definition, the Loss function was suggested.

The Loss Function is the function which describes financial loss both the part of the company and of the customer. When a product does not satisfy a customer, he may think that he wasted money or it was too expensive. When a company developed the product which requires any redesign or reproduction after its shipping, costs will exceed the estimation. Loss to a company can be categorized in these two ways.

Direct loss: warranty, increased service cost, dissatisfied customers.

Indirect loss: market share loss, need to increase efforts to overcome lack of competitiveness.

3. 5.1 Product Quality and Company Losses

We propose to illustrate the relationship of product quality and loss function visually is shown in Fig. 3.

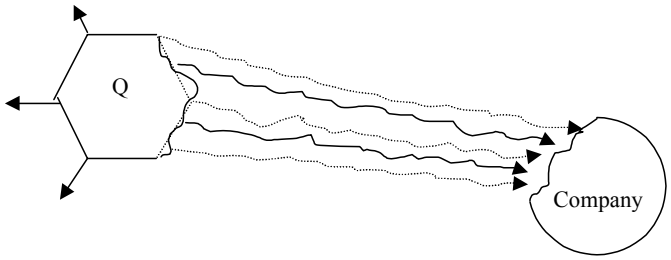


Fig. 3. Product Quality and Company Losses

The product quality ideally a complete hexagon shape brings both direct and indirect loss to the company. The curving line represents direct loss and the dotted line represent indirect loss.

These relationships can be represented in Fig.4 (Yano, 1995).

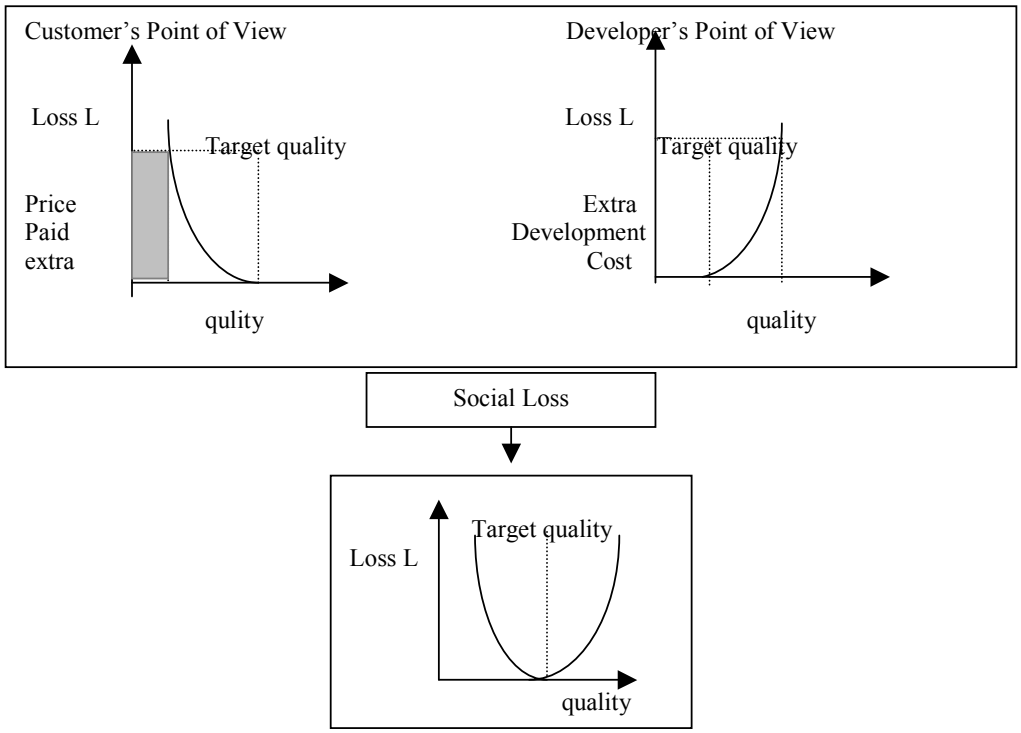


Fig. 4. Loss Function-1

The parabola in Fig. 4 shows the Loss Function (Yano, 1995). This graph shows that the further the variability of the product is shifted from the target quality value, the larger financial loss. The value of quality must be continuous.

3.6 *On-line Process Control in Manufacturing*

The manufacturing process shown in Fig. 5 represents the uncovering of problems/defects by testing a number of quality factors, and by subsequently trying to modify the product.

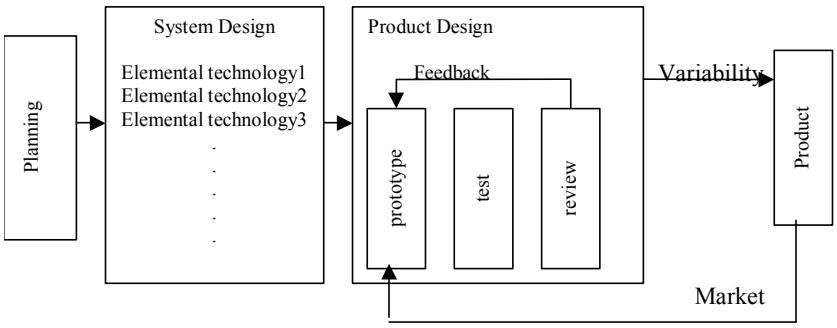


Fig. 5. Manufacturing Process [adapted from website-2]

The problems of this procedure are:

1. The development period tends to exceed its estimated length because even a single modification often leads previously acceptable quality factors to become worse and requires them to be further modified.
2. Problems occur because of the limitation of the testing process. It is uncertain that product quality is consistent under conditions which were not included in the testing before shipping. Therefore those products which did not provide expected functions expected will be rejected and the manufacturer has to compensate these problems. That leads to the further extension of the development process.

Overall, the extended development process means the extension of development costs and also a reduction in profit for the company because a longer development procedure will shorten the product's life span in the market in the fast development of modern technology.

In this investigation we propose to represent the Target Quality Value by a rectangle (shown in dotted lines in Fig. 6.).

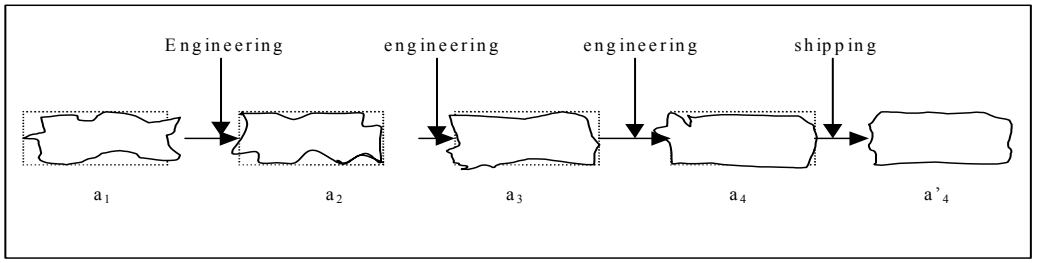


Fig. 6. Diagrammatic representation of Target Quality Value

In the traditional method of production, attempts to innovate the product quality on-line tends to cause additional variability in the product and it is difficult to shape the product quality to match the desired quality. Thus a_1 , a_2 and a_3 are engineered to reduce variability but it is difficult to achieve the target quality value in the shape of the required rectangle. Therefore, shipping the product will be made leaving uncertainty in the variability of the product after exceeding overtime and development cost. At shipping point, the product may appear to be satisfactory on the basis of tests performed. However, there is a possibility that a danger of unknown variability in the product might exist. This is shown as a'_4 in the diagram.

3.7 *The new method for reducing development cost in manufacturing*

To minimise the cost of development caused by variability of products, a new type of developing process was suggested by Taguchi. It emphasises the importance of testing and quality assurance in the early product life cycle, that is detailed assessment for the development of elemental technology. Fig. 7 depicts the Off-line Process Control.

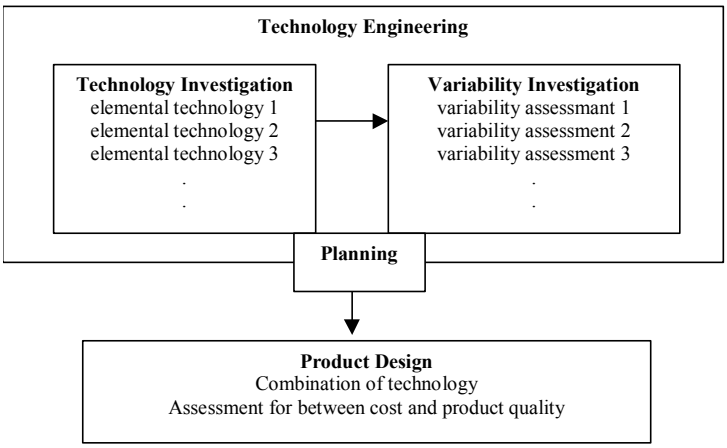


Fig. 7. Off-line Process Control [adapted from website-2]

3.8 *The advantages of the new procedure*

The characteristics of this procedure are:

Before planning, not only the feasibility of functionality required are investigated but also insensitivity to large variations (Noise) in the input condition is investigated. This is called Robust Design.

The advantages of adopting the Taguchi Method in the manufacturing process can be summarised as follows: "If Robust Design of elemental technology is well established, then developing products requires simply combining these elemental technologies and assessing required quality. Therefore there is no need to have a review group for feed back as in the traditional procedure and it is possible to reduce the number of product developing processes. Thus minimisation of cost is assured."

Furthermore, only when it is found that Robust Design does not solve the variability of a product, replacement of material (more expensive one) or the introduction of additional backup circuits considered. Thus minimisation of cost-up is assured.

3. 9 *Product Quality*

In addition to the visualization of Taguchi's Social Philosophy, we propose to describe product quality as a shape.

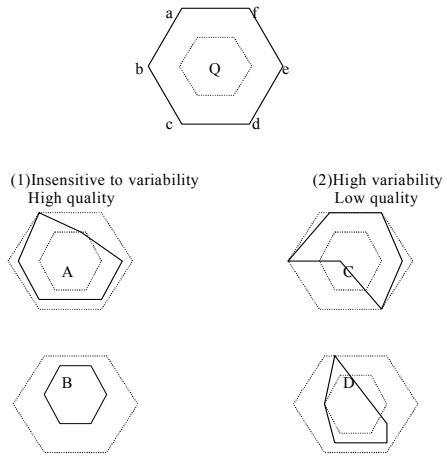


Fig. 8. Cultural Legal Shape of the Problem

The smooth line used to construct the first hexagon given above represents complete achievement of the specification of the product which requires six performance criteria a, b, c, d, e and f. The inner hexagon drawn using a dotted line shows the lower limits of acceptable performance which were specified in the requirement.

In the real world the performance of a product will vary. Products can be divided into two categories, such as products which are insensitive to variability in other words those which have high quality as defined in the Taguchi Method (category 1 above), and products which have high variability therefore have low quality (category 2) respectively. In this diagram, product B has higher quality than the almost perfect

product C, because it satisfies the lower limit of acceptable performance for all the performance criteria whereas product C does not. Therefore the quality is $A \geq B \geq C \geq D$.

3. 10 S/N Ratio

The S/N ratio (Signal-to-Noise ratio) is a measure of how sensitive a system is to noise, or variance. An insensitive (robust) system will have a high S/N ratio. Finding a correct objective function in an engineering design is very important. Failure to do so can lead to considerable inefficiencies in experimentation and even to wrong conclusions about the optimum levels.

There are common types of static problems. Three equations (2, 3 and 4) for calculating S/N Ratio are shown for type 1, 2 and 3 where η is S/N Ratio, δ is variance , μ is mean and y is quality factor value calculated in experiments.

1) Smaller-the-better type problem

The most desired value for a response or performance index of a product or process is zero. Such problems include minimization of surface defect count in manufacturing computer wafers, minimization of the pollution from a power plant and minimization of leakage current in integrated circuits.

$$\eta = 10\log\frac{\mu^2}{\delta^2} \quad \ldots\ldots\ldots\ldots (2)$$

2) Nominal-the-better type problem

In these types of problems, such as achieving target thickness in polysilicon deposition, the quality characteristics are continuous and non-negative and their target value is nonzero and finite. For these problems, when the mean becomes zero, the

variance also becomes zero.

$$\eta = -10\log\left\{\frac{1}{n}\sum y^2\right\} \quad \text{.....(3)}$$

3) Larger-the-better type problem

The quality characteristic is continuous and non-negative, and it is desirable that the value is as large as possible. Examples of such problems are the mechanical strength of a wire per unit cross-section area, the miles driven per gallon of fuel for an automatic vehicle carrying a certain amount of load. Maximization of a larger-the-better S/N ratio type of problem can easily be converted to maximization of a smaller-the-better type problem by considering the reciprocal of the quality characteristics.

$$\eta = -\log\left\{\frac{1}{n}\sum\left(\frac{1}{y}\right)^2\right\} \quad \text{.....(4)}$$

3. 11 *Noise Factors and their effect on Quality*

There are aspects of product quality which can be determined at the design stage. Those factors whose quality a designer has some control over quality which should be identified and their variability should be decreased by reducing the noise.

We propose a pictorial representation of the three possible scenarios as shown in Fig. 9 where *a*: factors over which a designer has some control, *a'*: factors whose variability can be reduced and *b*: factors over which a designer does not have control

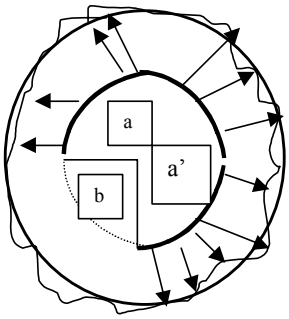


Fig. 9. Noise Factors and their effects on Quality

An example of a factor belonging to the *a* or *a'* is the geometric dimension of a part. Examples of factors belonging to *b* include environmental variables, product deterioration or manufacturing imperfection.

3. *15 Noises inside and outside a product*

In Fig. 9 the inner circle represents the sources of noise (design, process and product), and the outer circle represents the target quality value. The freehand circle represents the sources of noise factors. If the noise is controllable a designer tries to reduce the product's variation by reducing the sensitivity of the product to the source of variation rather than by controlling these sources.

Here, the noises inside and outside a product are described in Fig. 10 where a: material itself has noise, b: noise in manufacture, c; bought-in components noise,

d: noise in use and e: undetected noise during production.

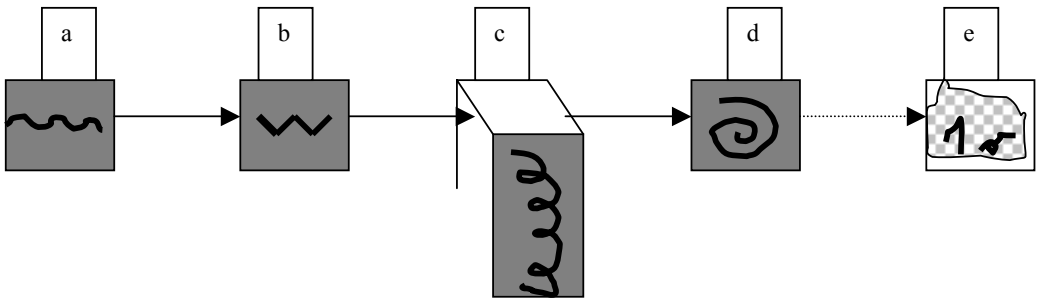


Fig. 10 Inside and Outside Noise

At the design stage these possible noises in the product, the process (including the noises in the process environment) and the various environments where the product is going to be used have to be minimized. The minimized variability in the product is shown as e.

4. Software Development and Taguchi

4.1 *Objectives for quality software products*

In this stage the software developer defines the robust design problem by clearly stating their objectives for the software product or the process improvement, and specifies the product/process response characteristics that reflect these objectives. The list of control parameters and noise variables are to be made. A brainstorming session by the programmers is useful for formulating the problem. The Ishikawa (cause and effect) or fishbone diagram for initiating a project may be used.

We must choose the correct objectives which are measurable. The simplest measurement which is suggested to indicate a product's quality is the number of bugs found during formal inspections which are conducted during the software life cycle under the specified methodology a company adopts.

The count of integration test failures can be summed up under a clearly defined testing procedure of a company's choice and be used as a quality factor. Designers then set the target (the actual incident count of integration testing failure) which can only be possible if the project under the adaptation of the Taguchi Method is one of several sister projects so that the empirical data is available. The final product's execution time also falls into this category.

If the project requires a totally new system development method for a company, measurements such as McCabe's Cyclomatic Complexity, Myers' Essential

Complexity or D-structuredness could be used to evaluate the software quality.

4.2 *Controllable Design Parameters*

Controllable design parameters can be found in the software development process, software products and the software development environment.

4.2.1 Design parameters of software development process

There are choices of the methodology as a design parameter such as V, W and X models, Spiral Model or the Deming or Shewhart Quality Improvement Cycle. When a specific type of the methodology is chosen as one of the design parameters, it will be challenging to adopt a new methodology which requires formal training session to provide the developers characteristic procedures to be taken in software development.

4.2.2 Design Parameters of the software product

We need to conduct the experiment before actually producing code. However, in software development it is not realistic to do this. Therefore for each project the guidance of coding process may be suggested such as " The target number of lines of code per module is less than 150". This might distress the concerned programmers and decrease their performance but the violations caused in the development environment will be considered as a noise factor in the design. The number of settings per design parameter is determined according to the empirical data based on which designers can construct settings.

According to Adrian Burr (Burr *et al*., 1996) the number of lines of code per module, target complexity, McCabe's cyclomatic complexity and the number of parameters

are considered to have significance for software quality. These factors can be used in designing experiments.

4.2.3 Design parameters in the software development environment

1) Machine , Operating System and Languages

The efficiency of a software product such as execution time has high priority as a software quality factor. To maximize the performance of a product, the choice of machines, operating systems and programming language has to be included in the list of parameters.

2) Human Factors

We must mention an additional design parameter which the Taguchi Method does not mention explicitly in the Robust Design. That is, performance variability in a human being such as his/her experience and communication skills needed in a software development team. The developers' performance has an effect on producing quality software products in a similar way to the effect of machines on the manufacturing of products. It is important to maximize and properly maintain programmers' performance. The possible control factors will be conducting educational sessions within and outside a company where software developers are encouraged to learn the new techniques of their interest or polish their skills. Recreational events may help developers to get to know each other better and this will be reflected in better communication and teamwork in an office. At the extreme, the design of the office environment itself is investigated. For example changing the type of chairs in current use to the ones designed to ease backpain caused by sitting all day will be welcomed by the developers. The temperature and humidity in the workplace also can affect the developers' performance. As mentioned before in the Taguchi Method these human

factors are not included in the product quality improvement procedure. Therefore the suggestions made here must be investigated in the software industry.

4.3 Noise Factors

Taguchi considers the effect of factors which are uncontrollable or are too expensive to control. For example, developers' experience can be controlled to a certain extent by the numbers of years in the profession, and by looking at the past projects involved. However no individual is identical. His/her capability may different from that of others' of similar experience. The health of the developers at the time of the investigation may affect on their performance at work.

The specification of machine, the difficulty of the system, inevitable human error and the size of the final system can also be Noise Factors.

It is desirable to develop a system that is insensitive to the noise factors by choosing the set of design parameters which are less affected by the variability of these factors.

4.4 *Design of Experiments*

4.4.1 Selection of Parameters and their settings

It is important to choose the design parameters which do not have reciprocal action among them. Orthogonal arrays are used to design the experiment. The advantage of using an orthogonal array is that the effect of several parameters can be determined along with minimizing the number of experiments. For example, the following list contains some of the widely-used orthogonal arrays which use two and three level design parameters. L18 is most commonly used in the Taguchi Method.

L4: Accommodates three two-level design variables

L8: Accommodates seven two-level design variables

L9: Accommodates four three-level design variables

L12: Accommodates eleven two-level design variables

L18: Accommodates one two-level and seven three-level design variables

4.4.2 An illustrative example

In the design shown in Table 2, it is considered that two levels of the design variables per design parameter is not sufficient to predict the optimum setting of the design parameters which has significance for software product quality improvement. It is assumed that the setting of the design parameter A (number of lines of code per module) in three level is necessary because the domain of this problem design parameter varies from a few number of lines of code per module to several hundreds of lines of code per module.

Table 2. Design Parameters and settings of Design Variables

Design Parameters		**Settings**	**Comment**
A	Lines of code per module	(50, 150, unspecified)	This setting were based on suggestions by Adrian Burr who and information found in Beizer (Beizer, 1990) and Georgiadou (Georgiadou, 1993).
B	Nesting Level according to the design	(3, 5, 7)	Cantata indicates McCabe's complexity 1 to 10 is 'pass' and greater than this results in 'fail'. At the design stage the flow diagram or Pseudo Code can be used to indicate the nesting level before coding.
C	Number of statements per module	(30, 40, 50)	It is difficult to control this parameter but it is not impossible. As is commonly known, the larger the number of statements in a module the harder it is to achieve 100% test coverage because of the cost involved.

4.4.3 Control Orthogonal Array

The choice of an array is entirely dependent on the choice of the design parameters and variables which the designers have determined from the analysis of the empirical data available or the predictions coming from their experience. Table 3 shows the example of an orthogonal array for controllable design parameters. The settings given

in Table 2 are used to demonstrate the table.

The settings in Column C were left empty because orthogonality is preserved despite the empty columns.

Table 3. Control Orthogonal Array - L 9

No. of Experiment (project in this case)	A Lines of code (50,150,free)	B Nesting level (3, 5, 7)	C	D No. of statements (30, 50, 70)
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	2
6	2	3	1	1
7	3	1	1	2
8	3	2	2	3
9	3	3	3	1

4.4.4 Noise Orthogonal Arrays

Two or three value settings are used for the identified noise factors that are important to produce a quality system and an orthogonal array is employed. Some noises originate in human factors in software development. An example is shown in Table 4. It is assumed that three 'noises' are identified and that each has two level settings so that an L4 orthogonal array was employed.

Table 4. Noise Orthogonal Array –L4

No. of Experiment	Team communication level (a, b)	Hardware speed (c, d)	Working space per person (e, f)
1	a	c	e
2	a	d	f
3	b	c	f
4	b	d	d

4.4.5 Matrix Experiment

An orthogonal array of control parameters is crossed by an orthogonal array of noise factors. The response for each combination of control and noise matrix experiments, mean and standard deviation are computed (Resit *et al*, 1995).

The purpose of this Matrix Experiment (Fig.11) in software development is to determine the optimal settings of controllable design parameters, which can be selected from internal attributes of software development, under a variety of work environments.

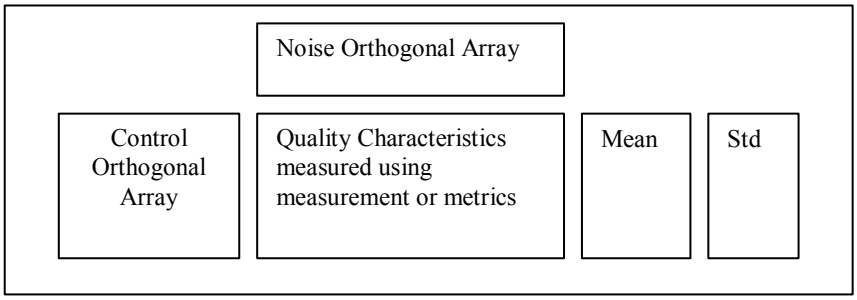


Fig. 11 Matrix Experiment

4.4.6 Quality characteristics and the S/N Ratio

The traditional Experimental Design uses S/N Ratio to observe the variance of data. The Taguchi Method aims to analyse S/N Ratio, and to uses it to reduce variability. An example is shown to demonstrate variability of possible design parameters' in software product followed by suggested quality characteristics in three categories with equations.

Eight GNU applications were downloaded from the internet (GNU, 1984) and analysed. Cantata (IPL, 1994) instrumented these files, calculated a number of metrics and produced CTL files which gave the static analysis of each modules in each application. These data were used to apply the Loss Function in order to see the variability of the listed measurements for the modules (Barbor, 1999).

The variability of listed measurements (S/N ratio) is calculated in two different ways. One is the S/N ratio of each application and the other is overall the variability of the obtained measurement derived by gathering all the modules in those eight applications. The results are plotted in a graph showing the degree of each variability.

Because the modules are different in size, an attempt is made to calculate the S/N ratio only from those which have more than 20 lines of code (Barbor, 1999). The overall characteristics of the degree of variability in each measurement are found to be unaffected. One example with explanatory notes is shown in Fig. 12.

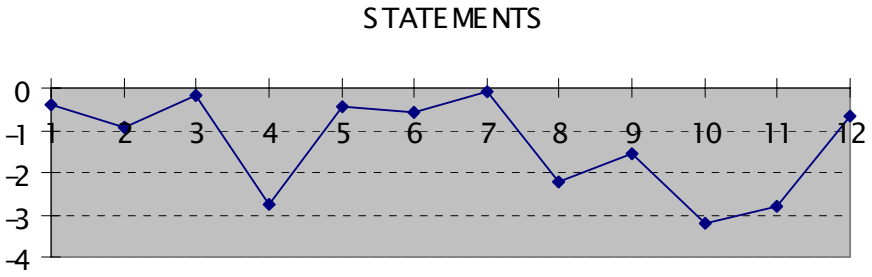


Fig.12. Variability

The numbers labeled for each dot in the graph are:
1: Expression Statements
2: For Loop Statements
3: While Loop Statements
4: Do Loop Statements
5: If Statements
6: Switch Statements
7: Return Statements
8: GOTO Statements
9: Break Statements
10: Continue Statements
11: Null Statements
12: INT Statements

This graph shows that as a whole the number of the four distinct statements (while, if, switch and return) per module has small variability. In other words, it indicates that regardless of the functionality of an application each module can be sufficiently constructed with similar number of these four statements or there is certain tendency of coding techniques or preference to those four statements among the developers of GNU. Furthermore, this result indicates that those four types of statements (3, 5, 6 and 7) are likely to be less sensitive to any variations of other possible design parameters of the eight applications.

The quality characteristics which fall into the following three categories are determined and the S/N Ratio is calculated by using the appropriate equation.

a) Smaller_ the_ better:

Number of testing failure at integration stage

Time spent to produce the application

System's execution time

Number of redesign

$$\eta = 10\log\frac{\mu^2}{\delta^2} \quad \text{.....(2)}$$

b) Nominal_the_better:

Decision coverage at integration stage.

Various academic software metrics values

$$\eta = -10\log\left\{\frac{1}{n}\sum y^2\right\} \quad \text{.....(3)}$$

c) Larger_the_better:

Test coverage

Number of test 'passes'

$$\eta = -\log\left\{\frac{1}{n}\sum\left(\frac{1}{y}\right)^2\right\} \quad \text{.....(4)}$$

4.5 *A framework for applying the Taguchi Method in software Development*

4.5.1 Processes and Products

In commercial environments, a considerable number of empirical data from previous projects have to be analysed with chosen design parameters and noise factors until all the combinations of design parameter settings are found for appropriate orthogonal arrays since it is not practical to conduct experiments with different design parameters settings for every future software projects. Once the data is collected to simulate the experiment, the S/N ratio is calculated and optimal design parameter settings are suggested.

In an academic environment, applications are developed with instructions for chosen internal attributes as controllable design parameters and their settings then quality characteristics are measured and the responses will be compared to determine the optimal design parameter settings.

The data obtained from both commercial and academic environments will be investigated and possible optimal design parameters settings will be proposed for larger scale software development.

As a result of the investigation presented in this paper, we propose a set of guidelines for adapting the Taguchi Method to software development. Fig.13 provides a schematic representation showing the sequence of the processes and the associated products at each phase.

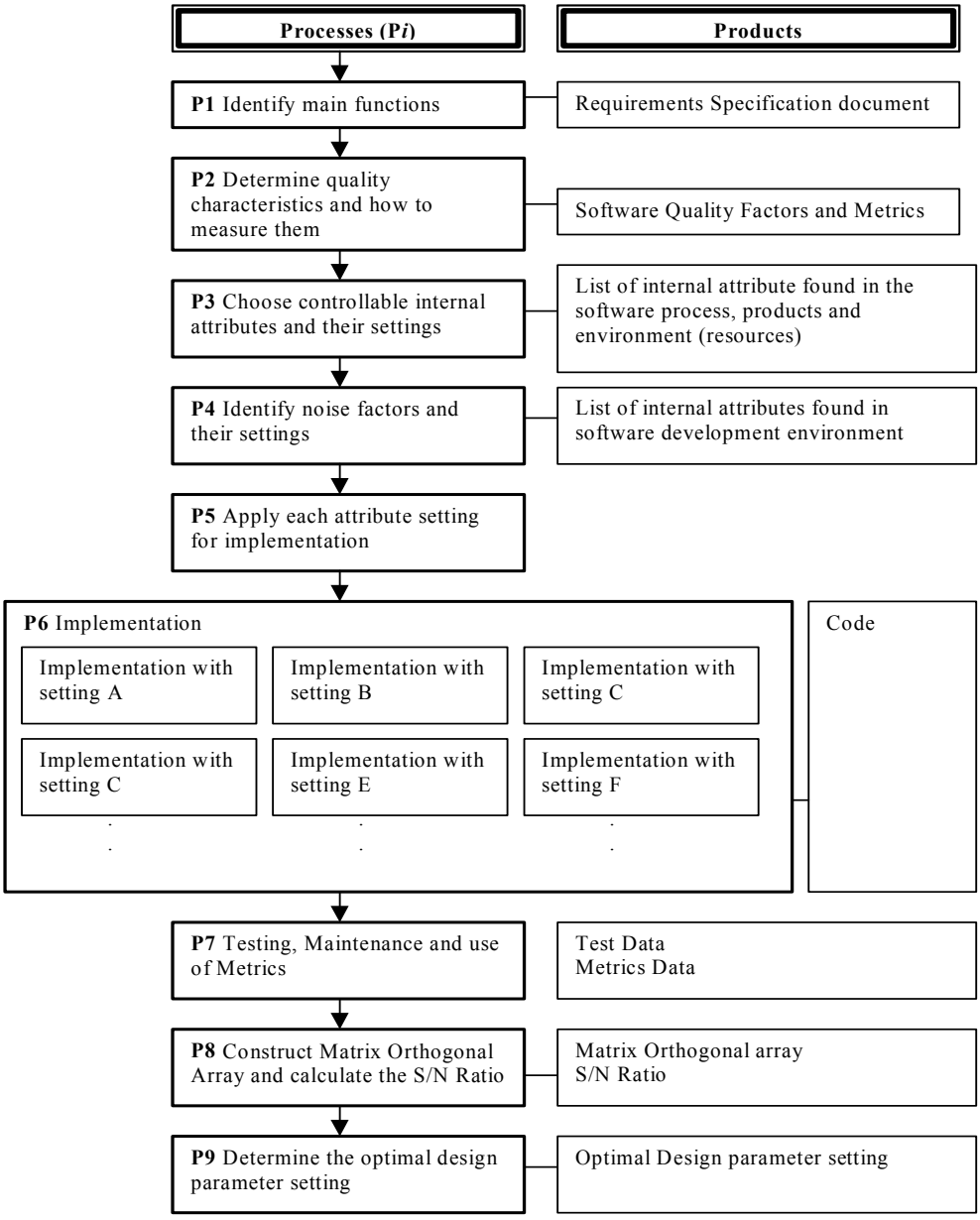


Fig. 13 Application of the Taguchi Method in Software Development

4.5.2 Difficulties

The difference between mass-production and the production of a unique piece of software, the importance of customer evaluation, and unavoidable changes of requirements at any stages of software life cycle make it difficult to apply the Taguchi

Method to software development. However, the authors believe that the concept of the Robust Design for product quality and minimization of production cost can be applied in software development since it enhances the probability of controlling the software production towards high quality software.

Difficulties are also experienced in carrying out the experiment since certain information that is needed when applying the Taguchi Method is often difficult and sometimes impossible to obtain. This includes in-house information such as time and cost of product development, the company's policy or guidelines for producing the code (if these exist), the environment such as the machine used or experience of the programmers.

So far, we have indications that the Taguchi Method can be applied to the pre-implementation stages of the software development Lifecycle (P1, 2, 3 and 4 in Fig 13).

5. Conclusions

The purpose of this paper was to investigate the possibility of adapting the Taguchi Method for Software Quality Improvements. A number of suggestions are made to adapt the Taguchi Method from ensuring quality in manufactured products to the development of quality software. Adjustments are necessary because of the fundamental differences between tangible manufactured products and software artifacts.

Taguchi's philosophy " The better the quality, the less the production cost" is true in the software industry because quality software must have good design as its own property enabling a software company to minimise the cost of repetitive development processes (redesigning, re-coding and re-testing) and the severity of testing. However it is not straightforward to conduct the Robust Design in software developments

mainly because software products are intangible. A problem specific to this study was that there were vast numbers of design factors for which it was not possible to obtain values such as cost and time spent, the methodology used, the experience of the developers, types of machines, the office environments, and whether the programmers have developed a similar type of product before or not. The use of metrics is particularly useful in software development. It will be easier to conduct statistical analysis activities if what needs to be done is clearly identified and when and how these activities should be carried out in the software development procedure is established. A correct and unambiguous set of controllable design parameters which have significance for software quality must be identified. This is only possible after the empirical data is collected and analysed under a systematic measuring operation. Further work will concentrate on using the proposed guidelines for carrying out a number of experiments in order to validate the adapted model.

6. Acknowledgements

The authors would like to thank Adrian Burr for the origin of the idea and his practical help. Sincere thanks are also due to Aasma Saadia for her incisive and useful comments, corrections and suggestions for the completion of this paper. Finally many thanks to IPL for providing the Cantata tool free of charge.

References

Akao, Y., 'Quality Function Development: Integrating Customer Requirements Into Product Design', Productivity Press, 1990.

American Supplier Institute, 'Taguchi Methods and Robust Design', www.amsup.com

Azuma, M., ' Information Technology-Software Product Evaluation-Indicators and Metrics', ISO/JTCI/SC7/WG6Project7, working draft, 1993.

* Baba, I., 'Prospect of the Off Line Quality Control', www.cnde.iastate.edu.jp

Barbor., N, 'Software Quality Improvement – Study of the adaptation of the Taguchi Method in software development-',university of North London, 1998.

Basili, V.R. and Weiss, D. M., 'A Methodology for Collecting Valid software Engineering Data. In Journal of the IEEE Transactions on Software Engineering, Vol.SE-13, No.6, pp.728-738, July 1984.

Beizer, B., 'Software Testing Techniques', 2nd edn, Van Nostrand Reinhold, 1990.

Brooks, F. P., 'The Mythical Man-Month: Essays on software engineering, 2nd edn', Addison-Wesley, 1995.

Burr, A. and Owen, M., 'Statistical Methods for Software Quality', International Thomson Publishing Inc., 1996.

Card, D.N., McGarry, F.M. and Page, G.T., 'Evaluating software Engineering Technologies, In Journal of the IEEE Transactions on Software Engineering, Vol. SE-13 No.7, pp.845-851, July 1987.

*Committee of the Taguchi Method, 'Beginner's Taguchi Method',www. T3.rim.jp

DeMarco, T., 'Structured Analysis and System Specification', Yourdon Press, 1978.

Fenton, N., Pfleeger, S.L. and Glass, R.L., 'Science and substance: A challenge to software engineers', IEEE Software, 1994.

Fenton, N.E, Iizuka, Y. and Whitty, R.W. (eds), 'Software Quality Assurance and Measurement': A Worldwide Perspective, International Thomson Computer Press, 1995.

Fenton, N.E. and Pfleeger, S.L., 'Rigorous & Practical Approach', PWS Publishing Company, 1997.

Galliers, R., 'Information Systems Research: Issues, Methods and Practical Guidelines', Blackwell, 1992.

Georgiadou, E., Karakitsos, G., Sadler, C. and Stasinopoulos, M., 'An experimental examination of the role of re-engineering in the management of software quality', Software Quality Management Vol. 2, Computational Mechanics Publications, 1993.

Georgiadou, E., Karakitsos, G., Sadler, C., Stasinopoulos, M. and Jones, R., 'Program maintainability is a function of structuredness', Software Quality Management Computational Mechanics Publications, 1994.

Georgiadou, E., Karakitsos, G. and Sadler, C, 'Improving the program quality by using the re-engineering factor metric rho', International Conference of the Israel Society for Quality, Nov 1994.

Georgiadou, E., Karakitsos, G. and Sadler, C., 'A method to evaluate the programmer response to equivalent programs', 1st International Conference on Software Engineering in Higher Education, editor King, G., Brebbia, C.A. and Ross, M. and Staples, G., Mar 1994

Georgiadou, E., Milankovic-Atkinson, M. and Suleiman, Z., 'A Formal Experiment to Evaluate the Significance of Object-Oriented Complexity Metrics', Proceedings of INPIRE'99, Crete, Greece, Sep1999, The British Computer Society.

Greenfield, T., 'Taguchi Policy: A ridiculous and dangerous fashion', article in RSS NEWS, 1995.

Hetzel, W. C., 'Making Software Measurement Work: Building an Effective Software Measurement Program', QED, Wellesley, Mass, 1993.

IPL, 'Cantata-Version 3.0', Information Processing Limited, 1994.

Jones, C., 'Applied Software Measurement', McGraw-Hill, 1991.

Keramopoulos E., Pouyioutas P. & Sadler C. 'GOQL, a graphical query language for Object-oriented Database Systems', Third Basque International Workshop on Information Technology, pp.35-45, Biarritz, France, Jul. 1997.

Kitchenham, B, ‘ Software Metrics: Measurement for Software Process Improvement’, NCC Blackwell Ltd., 1996.

Law, D. and Naeem, T., ‘ DESMET: Determining and Evaluation methodology for Software Methods and Tools’, In Proceedings of the BCS Conference on CASE-Current Practice, Future Prospects, Cambridge, England, March 1992.

Logothetis, N. and Wynn, H. P., ‘Quality Through Design: Experimental Design, ‘Off-line Quality Control and Taguchi’s Contributions’, Oxford Science Publications, 1989.

QUANTUM, ‘A measurement-based framework for the assurance of software quality’, DTI, 1992.

Resit, U and Edwin, B.D, ‘Design for Cost and Quality: The Robust Design Approach’, www.dfca.larc.nasa.gov

Schach, S.R., ‘Methodology characteristic Frameworks and Software Specification and Design: A Critique of “ Methodman **2**”, In Proceedings of the Forth International Work shop on Software Specification and Design, IEEE, pp.196-200, 1987.

Schulmeyer, G.G. and McMarius, J.I., ‘ Handbook of Software Quality assurance’, Van Nostorand Reinhold, 1992.

Shepperd, M.J. and Cartwright, M., ‘An Empirical Investigation of an Object-Oriented Software System’, Bournmouth University, October 1997.

Shepperd, M.J., ‘Foundations of Software Measurement’, Prentice Hall International Limited, 1995.

Stephan, H. K., ‘Metrics and Models in Software Quality Engineering’, Addison-Wesley Publishing Company, 1995

*Taguchi, G., ‘Quality Engineering Series 3-SN ratio for assessment of Quality’, Japanese Standards Association, 1993.

*Taguchi G., ‘Quality Engineering Series 5-Examples of the Taguchi Method-Japan’, Japanese Standards Association, 1996.

*Taguchi G., ‘Quality Engineering Series 7-Examples of the Taguchi Method-Measurements’, Japanese Standards Association, 1992.

*Taguchi G., ‘Quality Engineering Series 6-Examples of the Taguchi Method-US and Europe’, Japanese Standards Association, 1993.

Tajima, D. Matsubara, T., ‘The computer software industry in Japan’, IEEE Computer, 1981.

Yano, H., ‘Taguchi Method for beginners’, 1995.

Yourdon, E. and Constantine, L. L., ‘Structured Design’, Prentice Hall, 1979.

* References proceeded by an asterisk were written in Japanese. The given titles are free translation by the authors.

QW2001 Paper 4M1

Mr. David Fern
(Micros Systems Inc.)

How Testers Can And Should Drive Development Cycles

Key Points

- Test/QA should drive the development cycles.
- Build every week as if you are building a release candidate and grade it accordingly.
- Establish a weekly bug triage process to direct the engineering focus.

Presentation Abstract

Having finished our last project, which ended as a fire drill as usual, the managers of the development and test concluded that we weren't ever going to go through that again. All agreed the test team had been riding the development cycle bus long enough.

Our new paradigm has test driving the development cycle bus instead of engineering. Our first task was to organize and sell the idea to product and project management and a skeptical engineering team. Everyone including engineering, management, and especially test are now fully on board and wouldn't want it any other way. The following paragraphs outline our plan currently in use.

* Establish daily events/goals that will allow you to meet weekly build objectives. During the early phase of the software development cycle, engineering builds the software every week. Closer to the Beta release, the software can and is often built semiweekly. Each weekly build is treated as a release candidate in the sense that the state of the software is always known. If the software has to ship, it is known which areas have been tested and which modules are or are not ready to ship. Both engineering and test are assigned daily deliverables for which they are accountable to deliver to each other. Commitment of delivery by each team allows the build to occur successfully, and improves the state of the software weekly.
* The bug triage process is a weekly meeting where the lead engineer and lead tester agree on the bugs that will be fixed in the upcoming week's build cycle. This is the chance for the lead tester and lead engineer to reach a consensus and address the most severe bugs. In most cases, addressing the most severe bugs enables testing to continue. This also increases test's ability to improve the depth of its reporting on the state of the software each week. Triage also provides a time for discussion and risk analysis between the test and engineering teams.
* Grading the software is a two-tiered approach. Grades consist of weekly build

cycle grades and module grades. The grades are calculated by using the bug repository counts and a scale everyone is familiar with: A to F, just like school. Grades are distributed to each team member and posted where all can see.
* The test team grades the weekly cycle build. Most bugs are found during the sanity testing of the weekly build. If an engineer introduces showstoppers in the build which prevent testing, the build receives a letter grade of F. If an engineer breaks the weekly build, it receives the same grade and the engineer who breaks the build has to buy the entire team bagels.
* The software is also segmented into modules. Test also grades each module of the software weekly. Some module's grades will remain static while test recommends engineering to focus on more severe modules. The ultimate goal is to have all modules at a letter grade agreed upon in the test plan. The module grades are calculated using the same counts and scale as the weekly builds.
* This type of grading system goes beyond the use of metrics, in that everyone on the team knows the status of the overall software as well as the individual modules. All are striving for an A and each team member can see the progress and the readiness of the software.
* Finally, once the process is in place, it is easy to accelerate or decelerate the entire process as needed. The hardest part is starting, which includes getting organized, developing the grading criteria, the tools to perform the bug tracking, and of course, selling the idea to management and engineers. In the end, we all win. We are ready for show time and can produce a gold CD with a few finishing touches-just like we've done numerous times before in our weekly dress rehearsals.

About the Author

David Fern has a background in Foodservice Management that spans more than two decades. During the past four years he has been involved in all aspects of Food Service Point Of Sale application and devices, specializing in the development of Large Restaurant and Hotel computer systems. Currently he is a Test Specialist in Research and Development at Micros Systems, Inc. in Columbia, Maryland. MICROS is the premier developer of software for the hospitality industry worldwide.

Prior to his current professional endeavors, David was a Manager for the University of Maryland's Dining and Retail Services responsible for the campus Point of Sale Operations on the campus of over 60,000 students.

David recently had an article entitled "Testing Point of Sale Software" published in the November 2000 Quality Techniques Newsletter. David is a graduate of the University of Maryland College Park, Maryland with a Masters Degree in Industrial Technology and a Bachelors Degree in Business Management.

How Testers Can and Should Drive The Development Cycles

micros FIDELIO

How Test Can Drive The Development Cycles

- Establish Daily Events / Goals to meet weekly Build Objectives
- Make Bug Triage a Weekly Process
- Grade The Software as if it was the Release Candidate
- Let Everyone on the Team Know the Status of the entire Project at all Times
- Manage The Project with Flexibility

micros FIDELIO

When Should Test Take Over?

Code Complete

What is Code Complete?

- All functionality in the Test Plan has been completed and Engineering has completed all unit testing and we are at a state of Bug Fixes only.
- An initial installation program has been developed which includes any database conversion tools.

micros FIDELIO

What Planning is Required?

- Test Plan Outline
- Weekly Schedule of Events and Build Objectives
- Hot List
- Task List
- Grade Sheets
- Turkey Report
- Document Storage Locations

micros FIDELIO

Weekly Schedule of Events and Build Objectives

Purpose: To establish a routine of objectives and deliverables for each individual on the project.

- Daily Briefings
- Weekly Schedule

micros FIDELIO

The Weekly Schedule

Monday - Testing bug fixes is completed
Hot List is updated
Final list of bugs

Tuesday - All code must be checked in by 12:00
Test provides grades for the past weeks build
The Hot List is updated and posted

micros FIDELIO

The Weekly Schedule

(continued)

Wednesday- Weekly Build available by 8:00
Sanity testing completed by 12:00
Testing bug fixes begins

Thursday - Hot List updated
Triage

Friday - No Events - Enjoy the Day!

micros FIDELIO

Make Bug Triage a Weekly Process

Purpose: To establish communication between QA and Engineering in the following areas:

- Previous weeks build
- Determine which defects will be fixed in the next build
- Discuss upcoming weeks testing plan
- Discussion of upcoming possible risks
- General information exchange

micros FIDELIO

The Hot List

Purpose: List of defects that QA views as the most important and would like to have addressed in the next build.

What is a Hot List defect?

- A defect the prevents QA from moving forward on planned testing
- A listing of the most critical defects in the Software

micros FIDELIO

The Task List

Purpose: Important issues affecting the project that are not covered in the Hot List.

Why use it?

- Accountability
- Organization
- Tool to help everyone know the issues affecting the project.

micros FIDELIO

Grade software as if it were the Release Candidate

- Tangible results oriented to an A
- Engineers rewarded for their work
- Grades used to determine readiness of the software
- Talk About the grades
- Negotiate the grades
- Set milestones for a common goal

Developing A Grading Criteria

Grade the software with a two-tiered approach.

- **Build Grades -** Reflect the build as an overall picture of the state of the software.
- **Module Grades -** Reflect the status of each individual module in the project

The Weekly Build Cycle Grades

Purpose: To establish the readiness of the build for release.

How is it determined?

F - Installation is unsuccessful
The application will not start
Modules contain showstopper

micros FIDELIO

The Weekly Build Cycle Grades (continued)

D - Installation has moderate to serious usability issues.
Modules contain excessive severity high bug counts.

C - Infrastructure and core modules have moderate high and excessive severity medium bug counts.
Non-core modules contain moderate severity high and excessive severity medium bug counts.

micros FIDELIO

The Weekly Build Cycle Grades (continued)

B - Infrastructure and core modules contain moderate severity medium bug counts. Non-core modules contain moderate severity medium bug counts.

A - The aggregate bug count for all modules is low and severity low.

Minimum criteria for Beta is a B
Minimum criteria for General Release is an A.

micros FIDELIO

The Weekly Module Grades

Purpose: To establish the readiness of each module in the project.

Who are they determined?

- Bug counts
- Test Lead evaluation

micros FIDELIO

Important Concepts to Remember

Let Everyone Know The Status At All Times

Manage Project with flexibility

- Accelerate or decelerate as needed
- Modify the process for each project
- Don’t start grading or driving to soon

micros FIDELIO

Overcoming Barriers with Engineering and Management

Engineers

- Sometimes distressed over the first grades that are low

Management - Benefits

- People counting on others to deliver on time
- Helps to facilitate teamwork
- Triage initiates communication between QA and Engineering
- Everyone knows the project's status

micros FIDELIO

Why Should Test Drive the Development Cycles?

Benefits to Q/A Test

- Test Planning and scheduling can become easier
- QA will become involved in the entire development process

Challenge:

The entire development team is working toward one common goal of releasing defect free software.

micros FIDELIO

Thank you!

Questions

Please feel free to contact me if you would like further information on any of the topics discussed.

dfern@micros.com

micros FIDELIO

How Testers Can and Should Drive Development Cycles

Having finished our last project, which ended as a fire drill as usual, the managers of development and test concluded that we didn't ever want to go through that again. All agreed the test team had been riding the development cycle bus long enough.

Our new paradigm has test-driving the development cycle bus instead of engineering. The process of change is very difficult in any company; the changes that we have undergone have totally re-engineered our Research and Development department. It took courage for the test team to go from passively waiting for whatever bug the engineers decided to fix (or what piece of code they decided to complete) to the active role as facilitator of the entire process. The engineering team has been helped by having test push for dates and specific bug fixes in weekly meetings. In these weekly meetings each person knows that if they do not do their part as promised other people will not be able to deliver either. The process on the whole has brought the entire development team together working toward one common goal "To Release Quality Software".

The test teams first task was to organize and sell the idea to product and project management and a skeptical engineering team. We all looked back at our past release record and knew that we wanted change for the better. All knew this new process would involve a lot of effort to get started but it was better than doing things the way we always had in the past.

Everyone including engineering, management, and especially test are now fully on board and wouldn't want it any other way. Our innovative formula for success is broken down into the following sections:

- Establish Daily Events/Goals to Meet Weekly Build Objectives
- Make Bug Triage a Weekly Process
- Grade the Software With a Two-Tiered Approach
  - Grade the Weekly Cycle Build
  - Grade Each Module of the Software Weekly
- Let Everyone On the Team Know the Status of the Entire Project
- Manage The Project With Flexibility

In the sections that follow I have laid out the road map for how test can drive the development cycles.

David Fern
dfern@micros.com

When Should Test Take Over?

Test should be involved in the project from its inception. During the initial stages, tasks such as planning, scheduling and creating test cases should be developed. Once engineering has declared that the software is "Code Complete," test should move from the position of observer to being the driving force behind the project completion.

The exact definition of "Code Complete" must be defined in the Test Plan. Our definition of "Code Complete" is when engineering hands over the software and determines that all agreed upon functionality has been coded, unit tested, and an initial installation program including any database conversion tools are available. This means that we have reached a milestone and the project is in the "Bug Fix Only" mode.

If test starts driving the development cycles to early or to late the grading and scheduling becomes less effective. Early in the development process, grades of an F will be meaningless, as the software is not completed. Starting late in the development process by introducing a grading procedure will only disrupt the project flow by appearing to be a new process that seems not to be needed because the project appears to be progressing. The scheduling of weekly and daily events implemented too early in the development cycles will create many cancelled or postponed meetings as the development is sometimes difficult to predict. While starting these schedules too late appears to the team as a reorganization or interruption in their schedule.

What Planning is Required?

As in any testing the most important and useful tool for planning is the Test Plan Outline. In the Test Plan Outline you must include information about each of the tools discussed in this paper in order to make and use them most effectively.

Sections that need addressing are:

- Scheduling of Daily Events/Goals
- Task List
- Triage
- Hotlist
- Grading Methods and Manners
  - Weekly Cycle Grading
  - Weekly Module Grading
- The Turkey Report
- Documentation Storage Sites
- Exit criteria

Much of this information is already included in most thorough test plans though you will see in the sections that follow key points and concepts that will be required in order for the entire process to function as intended.

One final comment on planning, if you can properly plan all parts of the process with the assistance and input of both engineering and test from the start you will be well on your way to success.

Establish Daily Events/Goals to Meet Weekly Build Objectives

During the early phase of the software development cycle, engineering builds the software every week. Closer to the Beta release, the software can and is often built semiweekly or even more frequently. Each weekly build is treated as a release candidate in the sense that the state of the software is always known. If the software has to ship, it is known which areas have been tested and which modules are or are not ready to ship. Both engineering and test are assigned daily deliverables for which each is accountable to deliver to the other. Commitment of delivery by each team as well as each individual allows the build to occur successfully, and improves the state of the software weekly.

All members on the project from test, engineering, project management and documentation attend a daily morning briefing. The test or engineering manager direct the meeting, relaying pertinent information such as the condition of the build and most importantly ask engineering if bug fixes promised will make the build deadline. We then go around the room and each person tells what they will be doing for the day. This allows everyone to know what the other is doing. Telling the group individual objectives is a type of reinforcement because the commitment seems to be more effective for follow through and delivery. The additional by-products of the meetings are that it often uncovers important pieces of information and many times matches people up to solve problems together. The meeting is generally held in a small room with only a few chairs so that the attendees will not prolong the meeting but, get to the point.

Each individual owns weekly build objectives, always conscious that others are relying on them to follow through and deliver what they have promised. By starting these cycles you are practicing to make the Gold CD weekly.

Our Daily Events schedule for one project looks like this:

Monday

- All code must be checked in by 12:00 noon.
- Test provides grades for the past weeks build.
- The Hot List is updated and posted.

Tuesday

- The Build is available by 9:00 a.m.
- Sanity testing is completed by 12:00 noon.
- Retesting bug fixes begins.

Wednesday

- Triage at 10:00 a.m.
- The Hot List is updated.

Thursday

- There are no daily events scheduled.

Friday

- The final list of bugs is updated.
- The retesting of bug fixes is completed.
- The Hot List is updated.

The list above is only the weekly schedule for one project. When you are juggling multiple projects dates become important for the individual team members as well as the configuration management people whose task it becomes to build the actual software for multiple projects throughout the week.

Make Bug Triage a Weekly Process

The bug triage process is a weekly meeting where the lead engineer and lead tester agree on the bugs that will be fixed in the upcoming week's build cycle. This is the chance for the lead tester and lead engineer to reach a consensus and address the most severe bugs. In most cases, addressing the most severe bugs enables testing to continue. This also increases test's ability to improve the depth of its reporting on the state of the software each week. Triage also provides a time for discussion and risk analysis between the test and engineering teams.

The test lead will generally start the meeting with comments or questions about the weekly build, the testing effort or technical questions that the engineer is best suited to answer. Test produces a Hot List, which includes the top bugs as designated by the test team. This list is set up in a spreadsheet and includes the following information:

- The title of the bug in the bug repository
- A bug repository tracking number
- The module where the bug resides
- The status of the bug (bug verified, fixed etc.)
- What build the bug is promised to be fixed in
- Who owns the bug (engineering or test)

Engineering and test then decide which bugs will be worked on during the week. In addition it is verified that the past week's commitments have been met. A very important item to note is that engineering should not be using their precious time fixing bugs not on the list without the approval of test. If test can anticipate what will be in the build they are ahead of the planning game. Additionally if an engineer has extra time they should fix bugs that are in the best interest of the test team.

The expected outcome of the meeting is that test and engineering have addressed the highest priority bugs and decided which ones will be in the next build. This is executed by assigning a specific engineer the responsibility to fix the bug. If an engineer sees their name beside a bug it places necessary pressure to fix it in the allotted time.

It is imperative that the list be maintained since many decisions are based on this list. Once the meeting has adjourned test will update the bug repository and begin to build their test suite. The test lead maintains the list, which is always kept on a shared drive so that everyone has access to the information at all times.

The idea here is to weekly prioritize the bugs into a Hot List and have weekly fix commitments that everyone agrees to.

The Task List

The Task List is a document that has become an integral part of the triage process. This list contains the information about issues in the project that are not represented by an entry in the bug-tracking repository. During the initial project planning all situations and issues cannot be addressed and some are not even predicted. In our organization there is the possibility of features or parts of the project being added or deleted during the development cycles. We have developed the Task List for this type of item.

Items that we have included on our Task List are:

- Test planning - who, when, where, and items such as equipment that we need to find to complete the testing
- Documentation changes, questions and revisions
- Special questions that may arise (e.g., network configurations)
- Various other questions or concerns
- Beta specific questions or tasks

In the Task List we include information such as the task name, a brief description of who is responsible for the action, the expected completion date the actual completion date and, if the expected completion date has passed, what action is being done to complete the task.

It is quite possible to use that bug-repository as a place to store items from the task list, but from our experience taking this type of list to Triage weekly with a commitment from an individual and the expected date move the process along more quickly and you have a list of due dates on tasks at your finger tips.

The Turkey Report

What the heck is a Turkey Report? This is actually just a “Report Card” or general update on the project addressing the state of the software, revised schedules, issues affecting the project or any other possible roadblock.

Once we get into the project the big guys upstairs need an update every so often to ensure that we are making progress. The dates to produce these reports should be included in the test plan.

Why “Turkey Report” you might ask? The Development Manager would ask us, How's your module coming? Is it done like a turkey? Since then the name has stuck and we have a Turkey Report.

Grade the Software With a Two-Tiered Approach

Grading the software is a two-tiered approach. Grades consist of weekly build cycle grades and module grades. The grades are calculated by using the bug repository counts and a scale everyone is familiar with: A to F, just like school. Grades are distributed to each team member and posted where all can see.

This type of grading system goes beyond metrics in that you are not just given a number. The team sees and discusses the posted grades and the team as a whole sets milestones for a common goal.

Grade the Weekly Cycle Build.

The test team grades the weekly cycle build. Most bugs are found during the sanity testing of the weekly build. If an engineer introduces showstoppers in the build, which prevent testing, the build receives a letter grade of F. If an engineer breaks the weekly build, it receives the same grade and the engineer who breaks the build has to buy the entire team bagels.

The weekly cycle build report is a way to look at software as a whole and how it has changed over the past week. This report is set up as a spreadsheet and compiled by the test lead through the bug repository and has the following information:

- Cycle Build Number
- Release Date
- Overall Grade
- Bugs submitted during the past week broken down by severity
- Current open bug counts broken down by severity
- Total bugs on the project opened and closed
- Number of bugs closed during the past week

In the establishment of grading criteria it becomes important to consider key elements and divide the entire project into modules. The key element that we use is the ability of the software to be successfully installed. We break up the project into modules. These modules are then grouped into Infrastructure, Core and Non-Core. The infrastructure is composed of those modules that directly relate to the overall functioning of the software. The core modules are those that are important in the functionality of the software and finally the Non-Core modules are those that are nice to have but are not essential. By categorizing the modules the build grading becomes much easier.

The criteria used in the weekly build grading is based on the following information:

F

- The install was unsuccessful.
- The application would not start.
- The infrastructure modules contain showstopper bugs.
- The core modules contain showstopper bugs.
- The non-core modules contain showstoppers.

D

- The install has moderate to serious usability issues.
- The infrastructure modules contain excessive severity high bug counts.
- The core modules contain excessive severity high bug counts.
- The aggregate of non-core modules contain excessive severity high bug counts.

C

- The infrastructure modules contain moderate severity high and excessive severity medium bug counts.
- The core modules contain moderate severity high and excessive severity medium bug counts.
- The Aggregate of non-core modules contain moderate severity high and excessive severity medium bug counts.

B

- The Infrastructure modules contain moderate severity medium bug counts.
- The core modules contain moderate severity medium bug counts.
- The aggregate of non-core modules contain moderate severity medium bug counts.

A

- The aggregate of bug counts for the entire system is low and the severity is low.

The criteria that we use for a Beta is an overall grade of a **B**. An **A** is awarded after a Beta installation and no further bugs are uncovered. The criteria used for General Release is an overall grade of an **A**.

The test lead is again responsible for calculating and maintaining the build grades which are also kept on a shared drive. This ensures everyone has access to the information anytime.

The point is that everyone knows the overall state of the software at any given time.

Grade Each Module of the Software Weekly

The software is also segmented into modules. Test also grades each module of the software weekly. Some module's grades will remain static while test recommends engineering to focus on more severe modules. The ultimate goal is to have all modules at a letter grade agreed upon in the test plan. The module grades are calculated using the same counts and scale as the weekly builds.

The module grades are a way to look at software by module and to lend additional insight in to the weekly build grade. Everyone focuses on how individual modules have fluctuated over the past week. An important item to note is that it is imperative to have modules set up in the bug repository correctly. Too few or too many modules will skew the module reporting. The size of the module and quantity of bugs are considered when evaluating the modules and assigning grades.

This report is created in a spreadsheet and compiled by the test lead utilizing the bug repository database with the following information:

- The Module Name
- The Number of bugs by module broken down by severity
- The Number of total bugs by module
- A weighted average
- Total Score
- Grade
- Comments

The weighted average—we have four levels of severity for bugs and assign scores in the following way:

- Showstopper = 5
- High = 4
- Medium = 3
- Low = 2

We multiply the number of bugs in each module by their respective weight and then add each to get the Weighted Total. This can be easily set up as a formula in the spread sheet.

The Total Software Score is calculated by an average of each module.

The Grade is calculated using the following values:

F = 0 - 30 = Fail

The bug count contains showstopper bugs; the software is in an unstable condition.

D = 31 - 50 = Below Average

The bug count for module contains excessive severity high bug count; the software needs work.

C = 51 - 70 = Average

The bug count for module contains moderate severity high; the software needs improvement.

B = 71 - 90 = Good

The bug count for module is low and severity is medium; the software may be released.

A = 91 - 100 = Excellent

The bug count for module is low and severity low, “Zero Defect.” The software is ready for General Release.

The test lead is also responsible for maintaining the module grades, which are also kept on a shared drive so that everyone has access to the information at anytime. The key point is everyone knows the state of each module at any given time.

Let Everyone On the Team Know the Status of the Entire Project

This type of grading system goes beyond the use of metrics; everyone on the team knows the state of the overall software as well as the individual modules. All on the project are striving for an A and each team member can see the progress as well as the readiness of the software.

This reinforces the fact that we are all working toward the same goal and allows each person to have the same data on the project. Perceptions may be different but, the grades should align most differences. Slips and broken builds can never be totally eliminated but, if we all agree where we are, it becomes much easier to determine success and reach the next milestone and the ultimate goal of releasing quality software.

Manage the Project With Flexibility

Finally, once the process is in place, it is easy to accelerate or decelerate the cycles as needed. At the beginning of many projects we will not start the module grades until we have built the software for a few weeks. This is due to the fact that unit testing has not been completed and the grades would not reflect the true state of the software. As we get closer to the Beta release we may build three or four times a week depending on the necessary iterations and if any last minute fixes are required.

This formula needs to be adapted for each organization and even each project. In the planning stages each project's weekly cycles, release grades and even module names must be evaluated and agreed upon by everyone on the project. We can't give you exact module or cycle grade sheets because even in our organization some test leads have adjusted the sheets to more efficiently fit their project.

The test manager can not just copy grade sheets and sit back. Each project will be different and require collecting different information. The test manager should continually tweak the process and adjust it as needed. It's hard work but, effort well worthwhile when you see the outcome of using this process.

Overcoming Barriers with Engineering and Management

Even if you successfully have all parts of the plan in place you will need to have everyone on board including the engineers, management and especially test. The most important group to have committed to the process is test because driving the cycles requires more work, effort and responsibility in maintaining and keeping the project on track, though the rewards and total positive effect on the testing process can be great.

Management needs to buy in to give test the ability and authority to impact the project by directing engineering actions and focus. It must be understood that test needs to be involved from the start of the project to be most effective and cannot come in at the end and work around what has been developed. Additionally, by giving test more responsibility you have another set of eyes that are continually evaluating the project and a process that will force the engineers and test to work together towards a common goal each depending on the other to succeed.

Finally, engineering needs to buy in, but these guys may be the hardest to get on board. They may feel as if they have lost some power and will have their work graded weekly. The engineers must understand that they still have as much input as ever and have test as a partner to plan help in the planning for the project. The efforts of engineering will be well rewarded as the grades improve during the development cycles. Everyone needs to understand that in the beginning they may receive Fs for many weeks. The days of the engineer sitting down and fixing many small defects for the sake of saying "I fixed 30 defects today," all of which actually have little effect on moving the project forward, are long gone.

Why Should You Test Drive The Development Cycles?

We are not saying that the test should take over in a coup, which would only exacerbate the rocky relationship between engineering and test. As with any change it takes time and buy in. However, there are tremendous benefits and someone has to drive the bus so why not test? Some of the benefits to the test team are having modules fixed in an order that allows the test schedule to be adhered to, which allows testing to be performed more efficiently. Finally, test is always pressed for time toward the end of a project, but if they start controlling the software's destiny from the beginning the end is not so painful.

The hardest part is starting, this includes getting organized, developing the grading criteria, the tools to perform the bug tracking, and of course, selling the idea to management and engineers. In the end, we all win. We are ready for show time and can produce a gold CD with a few finishing touches-just like we've done numerous times before in our weekly dress rehearsals.

QW2001 Paper 4M2

Dr. Cem Kaner
(Florida Institute of Technology)

Managing The Proportion Of Testers To Developers

Key Points

- Magic ratios, like 1 tester per programmer, don't reflect project context
- The ratio should reflect the actual division of labor between testers and others
- Low ratios of testers to other developers may be better than higher ratios

Presentation Abstract

This talk summarizes ideas surfaced in a working meeting of the Software Test Managers Roundtable. We asked how to decide, for a given project, what is the best ratio of testers to other developers.

That ratio is sometimes 1-to-1, and 1-to-1 can foster close relationships in paired tester-to-programmer teams. But many of us at the meeting reported that our most successful projects had a much lower proportion of testers and that some of our least successful projects had much higher proportions of testers.

We start by asking what it means to say there is a 1-to-1 ratio. It turns out that different people have entirely different ways of calculating this. Many of the comparisons that we've seen are between apples and alligators.

From there, we look at some of the workload factors that favor high numbers or low numbers of testers. For example, a product that has lots of bugs needs lots of testers. A project team that invests heavily in early analysis and design will probably need less tail-end testing. As a different class of example, a project that relies heavily on external configurations and components that are outside of the control of the local project team will need extensive testing. Project-specific factors will drive you toward different ratios, and toward different ratios at different times in the project.

The associated paper lists and explains a wide range of factors and suggests some collaborative methods that you might use to staff appropriately for your particular situation.

About the Author

Cem Kaner is Professor of Computer Sciences at the Florida Institute of Technology.

Prior to joining Florida Tech, Kaner worked in Silicon Valley for 17 years, doing and managing programming, user interface design, testing, and user documentation. He is the senior author (with Jack Falk and Hung Quoc Nguyen) of TESTING COMPUTER SOFTWARE (2nd Edition) and (with David Pels) of BAD SOFTWARE: WHAT TO DO WHEN SOFTWARE FAILS.

Through his consulting firm, KANER.COM, he teaches courses on black box software testing and consults to software publishers on software testing, documentation, and development management.

Kaner is also the co-founder and co-host of the Los Altos Workshop on Software Testing, the Software Test Managers' RoundTable, the Workshop on Heuristic & Exploratory Techniques, and the Florida Workshops on Model-Based Testing.

Kaner is also attorney whose practice is focused on the law of software quality. He is active (as an advocate for customers, authors, and small development shops) in several legislative drafting efforts involving software licensing, software quality regulation, and electronic commerce.

Kaner holds a B.A. in Arts & Sciences (Math, Philosophy), a Ph.D. in Experimental Psychology (Human Perception & Performance: Psychophysics), and a J.D. (law degree). He is Certified in Quality Engineering by the American Society for Quality.

Managing the Proportion of Testers to Other Developers

Cem Kaner, J.D., Ph.D.
Florida Institute of Technology

Elisabeth Hendrickson
Quality Tree Software, Inc.

Jennifer Brock
Ajilon Software Quality Partners

Acknowledgements

This presentation is partially based on a meeting of the Software Test Managers Roundtable (STMR 3) in Fall 2001. The meeting participants were:

Sue Bartlett
Fran McKain
Bret Pettichord
George Hamblen
Brian Lawrence
Jennifer Smith-Brock
Hung Quoc Nguyen
Laura Anneker
Elisabeth Hendrickson
Chris DeNardis
Jim Williams
Cem Kaner
Kathy Iberle
and Neal Reizer.

A Puzzle

At STMR, we asked what were the managers' largest and smallest ratios of testers to other developers, and how these ratios felt:

- **There were very small ratios (1-to-7 and less) and very large ratios (5-to-1).**
- **Some of each worked and some of each failed.**
- **Many of us remembered successful projects with ratios lower than 1-to-1 more favorably than successful projects with larger ratios.**

Why is there such a range of successful ratios, and why would test managers be happy with relatively low ratios?

These Ratios are Incommensurable

Staff	Counted as
4 programmers	programmers
1 development manager	programmer
1 test lead	Tester
1 black box tester	Tester
2 test automation engineers	Testers
1 buildmeister	Tester

1-to-1 ratio? But if there is a new build, how many black box testers are available to test it?

What IS this Ratio?

Staff	Counted as
1 programmer	programmer
1 toolsmith	programmer
1 buildmeister	programmer
1 development lead	programmer
1 development manager	programmer
1 test lead	Tester
4 black box testers	Testers

1-to-1 ratio? How many programmers are available to fix five testers' bug reports?

What IS this Ratio?

Staff	Counted as
5 programmers	programmers
5 on-site consultants (doing programming)	???
1 project team (10 people) under contract to deliver components.	???
1 full-time, on-staff test engineer	tester
3 technicians	???
3 temporary technicians (work for a contracting agency)	???
3 testers who work offsite in an independent test lab	???

How to count technicians and consultants? Is the ratio of testers to programmers 10-to-5, 10-to-20, 1-to-20, or something else?

Incommensurable Ratios

- **Who to count?**
  - **Experienced people count the same as juniors?**
  - **Consultants, techs, contractors count the same as full-time employees?**
  - **Managers?**
- **What do the counted people do?**
  - **When programmers do testing, inspections or reviews, are they testing? (Do we count them as testers?) Even if they are testing their own code?**
  - **If testers help with debugging, are they programmers?**
- **When to start counting?**
- **Compare headcount or budgets?**

Small Ratios can be Better

- **Code coming into testing is clean, designed for testability, and has good debug support.**
- **Prevention is emphasized, and a person who makes a bug is expected to fix it.**
- **Bug churn rate is low.**
- **Staff turnover in testing and programming groups is low.**
- **Company hires skilled, experienced testers not "bodies."**
- **Shared agreement on the role of the test group.**
- **Trust and respect between programmers and testers.**
- **The groups help each other become more productive (e.g, helping them build tools).**
- **Extensive unit test library that programmers run when they update the product. (Read about Extreme Programming.)**

Small Ratios can be Worse

- Time to market seen as more important than finding / fixing defects.
- Dominant market position allows seller to ship defects and charge extra for maintenance and support.
- Testers perceived as not contributing because they don't write code.
- Testers perceived as too expensive, testing is easy anyway.
- Testers perceived as incompetent, counterproductive twits.
- Test manager perceived as a whiner who uses his staff ineffectively.
- Test group's work perceived as poor, overemphasizing unimportant issues, or as politically motivated overemphasizing process.
- Toxic relation between testers and programmers, resulting in bug churn, excessive turnover.
- Product is so complex that it is too expensive to train new testers.

Large Ratios can be Better

- Product might be knitted together from many externally written components or it might be an upgrade of an existing product.
- Extensive configuration testing needed.
- Extensive documentation and repetitive labor needed because of high litigation risk (e.g. safety-critical).
- Extensive documentation needed for software sold in its entirety to a customer who assumes responsibility for future maintenance, support and enhancement.
- Market is picky about fit and finish.
- Load testing is needed.
- Testers serve multiple roles, such as domain expert, build support, archivist, network administrator, debugging, spec writer, code reviewer, benchmark competing products, etc.

Large Ratios can be Worse

- **Testers don't understand domain or combinatorial testing, try too many redundant tests.**
- **Large numbers of low-skill testers.**
- **Manual execution of large sets of fully scripted test cases.**
- **High test group turnover, constantly in training mode.**
- **Testers have inadequate tools, space, and equipment.**
- **Inefficient testing because the project doesn't use basic control procedures such as smoke tests or configuration management.**
- **Software was not designed for testability.**
- **Excessive time spent collecting / gossiping about non-productive "metrics."**
- **Programmers send excessively buggy code into testing.**
- **Programmers don't test their own code, relying on testers instead. The more testers, the less these programmers do.**

Ratios Should Emerge from Project

- **Test groups play different roles in different companies.**
- **Testing projects vary widely in the amount of new code needed compared to the amount of testing needed.**
- **Programs vary widely in their complexity and bugginess.**
- **Markets vary in their error-tolerance.**
- **Projects differ widely in their documentation requirements.**

To justify your staff size, work from your staff's tasks.

Beware of overstaffing, it can do more harm than good.

MANAGING THE PROPORTION OF TESTERS TO (OTHER)[1] DEVELOPERS

Cem Kaner, J.D., Ph.D.
Florida Institute of Technology

Elisabeth Hendrickson
Quality Tree Software, Inc.

Jennifer Brock
Ajilon Software Quality Partners

ABSTRACT

One of the common test management questions is what is the right ratio of testers to other developers. A credible benchmark number offers convenience and bargaining power to the test manager working with an executive who has uninformed ideas about testing or whose objective is to spend the minimum necessary to conform to an industry standard.

We focused on staffing ratios and related issues for two days at the Fall 2000 meeting of the Software Test Managers Roundtable (STMR 3).[2] This paper is a report of our results. We assert the following:

- One of the common answers is 1-to-1 (1 tester per programmer) or that 1-to-1 is the common ratio in leading edge companies[3] and is therefore desirable. Our experience has been that (to the extent that we can speak meaningfully about ratios at all) 1-to-1 has sometimes been a good ratio and sometimes a poor one.

---

[1] In most companies, testers work in the product development organization and they are part of the technological team that develops software products. Testers *are* developers. The ratios that we are interested are the ratio of testers to the *other* developers on the project.

[2] Software Test Managers Roundtable (STMR) meets twice yearly to discuss test management problems. A typical meeting has 15 experienced test managers, a facilitator and a recorder. There is no charge to attend the meetings, but attendance must be kept small to make the meetings manageable. If you are an experienced test manager and want to join in these discussions, please contact Cem Kaner, kaner@kaner.com. The meeting that is the basis for the present paper was STMR3, in San Jose, CA. Participants included Sue Bartlett, Laura Anneker, Fran McKain, Elisabeth Hendrickson, Bret Pettichord, Chris DeNardis, George Hamblen, Jim Williams, Brian Lawrence, Cem Kaner, Jennifer Smith-Brock, Kathy Iberle, Hung Quoc Nguyen, and Neal Reizer.

[3] We thank Ross Collard (1999) for providing us with a summary of his interviews of senior testing staff at 18 companies that he classed as "leading-edge," such as BMC Software, Cisco, Global Village, Lucent, Microsoft. Six companies reported ratios of 1-to-1 or more, and the median ratio was 1-to-2.

- Ratios are calculated so differently from project to project that they probably incomparable.
- Project-specific factors will drive you toward different ratios, and toward different ratios at different times in the project. Such factors include (for example) the incoming reliability of the product, the extent to which the project involves new code that was written in-house, the extent to which the code was subjected to early analysis and review, the breadth of configurations that must be tested, the testability of the software, the availability of tools, the experience of the testers and other developers, corporate quality standards, and the allocation of work to testers and other developers.
- More is not necessarily better. A high ratio of testers to programmers may reflect a serious misallocation of resources and may do more harm than good.
- Across companies, testers do a wide variety of tasks. The more tasks that testers do, the more tester-time is needed to get the job done. We list and categorize many of the tasks that testers perform.
- The set of tasks undertaken by a test group should be determined by the group's mission. We examine a few different possible missions to illustrate this point.

PROBLEMS WITH RATIOS

What do we mean when we refer to a 1-to-1 ratio of testers to other developers? Across groups, these ratios can have wildly different meanings.

Consider the following stories:

Jane manages a project with the following personnel:

Staff	Counted as
4 programmers	programmers
1 development manager	programmer
1 test lead	Tester
1 black box tester	Tester
2 test automation engineers	Testers
1 buildmeister	Tester

According to the numbers, there's a 1-to-1 ratio between programmers and testers. However, when a new build comes into the test group, only one person is available to test it full time—the black box tester. Because of the apparent 1-to-1 ratio, management is puzzled by how long it takes the test group to do even simple tasks, like accept or reject a build. Jane is hard-pressed to explain the bottleneck to management—they keep coming back to the 1-to-1 ratio and insisting that means there are enough testers. The testers must be goofing off.

Now consider Carl's dilemma. His staff looks like this:

Staff	Counted as
1 programmer	programmer
1 toolsmith	programmer
1 buildmeister	programmer
1 development lead	programmer
1 development manager	programmer
1 test lead	Tester
4 black box testers	Testers

According to these numbers, there are 5 programmers and 5 testers, a comfortable 1-to-1 ratio. The testers report dozens of bugs per week. However, because they have no access to the source code (they test at the black box level), they cannot isolate the bugs they report. It takes the programmers significant time to understand and fix each reported issue. Carl is hard-pressed to explain why the testers can find bugs faster than his staff can fix them. Are his programmers lazy?

Sandy's department provides even more counting challenges:

Staff	Counted as
5 programmers	programmer
5 on-site consultants (doing programming)	???
1 project team (10 people of various specializations) who are under contract with Sandy's company to write and deliver a series of components to be used in Sandra's product.	???
1 full-time, on-staff test engineer	tester
3 technicians (they work for Sandry's company, are supervised by the engineer, but have limited discretion and experience)	???
3 temporary technicians (they work for a contracting agency, not Sandy's company, they report to the test engineer, but are not counted in the company's headcount)	???
3 testers who work offsite in an independent test lab	???

Should we count consultants as programmers? What about programmers who work for other companies and are simply selling code to Sandy's company? Should we count technicians as testers? What about technicians or other testers who work for other companies and provide testing services under contract? We don't know the "right" answer to these questions. We do know that different companies answer them differently and so they would calculate different ratios (ranging from 1-to-10 through 10-to-1) for the same situation.

Here are even more of the classification ambiguities in determining the ratio of testers to programmers:

- Are test managers testers? Are project managers developers? What about test leads and project leads? If a test lead sometimes runs test cases, should we count her as a tester for the hours that she is hunting for bugs? What about the hours she spends reviewing the test plans of the other testers?

- When programmers do code reviews, they find defects. Should we count them as testers? Imagine a six-month project that has one officially designated tester and ten officially designated programmers. In the first four months, the programmers spend 60% of their time critically analyzing requirements, specifications, and code, doing various types of walkthroughs and inspections. They find lots of problems. (In the other 40% of their time, they write code.) The tester also spends 60% of her time reading and participating in the meetings. Her other 40% is spent on the test plan. For these four months, should we count the ratio of testers to programmers as 1-to-10 or as 7-to-4? (After all, didn't the programmers spend 6 person-months doing bug hunting and only 4 person-months writing code?)
- If the testers write diagnostic code or tools that will make the programmers' lives easier as well as their own, are they working as testers or programmers?
- Imagine a six-month project that starts with four months of coding by ten programmers. During this part of the project, there are no testers. In the last two months, there are ten testers. Should we count this as 10-to-10 ratio or 60-to-20? (After all, there *were* 60 programmer-months on the project and only 20 tester-months.)
- Suppose that your company spends $1,000,000 licensing software components. These components required 36 programmer-months (and an unknown number of tester-months) to develop. Your company uses one programmer for 6 months to write an application that is primarily based on these components. It assigns one tester for 6 months. Is the ratio of testers to programmers 1-to-1 or 1-to-7 or something in between?
- How should we count technical writers, tech support staff, human factors analysts, systems analysts, system architects, executives, secretaries, testing interns, programming interns, marketeers, consultants to the programmers, consultants to the testers, and beta testers?
- If the programmers dump one of their incompetents into the testing group and one of the testers has to work half-time to babysit him, did the ratio of testers to programmers just go up or down? In general, if one group is consistently more (or less) productive than industry norm should we count them as if there were more (fewer) of them?

The answers to these questions might seem to be obvious to you, but whatever *your* answers are, someone respectable in a respectable company would answer them quite differently. At STMR 3, we marveled at the variety of ways that we counted tester-units for comparison with programmer-units. Because of the undefined counting rules, when two companies (or different groups in the same company) report their tester-to-programmer ratios, we can't tell from the ratios whether a reported ratio of 1 (tester) to 3 (programmers) involves more or less actual quality control than a ratio of 3 (testers) to 1 (programmer).

To put this more pointedly, when you hear someone claim in a conference talk that their ratio of testers to programmers is 1-to-1, you will probably have no idea what that means. Oh, you might have an idea, but it will be based on *your* assumptions and *not their*

situation. Whatever your impression of the staffing and work-sharing arrangements at that company is, it will probably be wrong.

WHAT FACTORS SUPPORT DIFFERENT RATIOS?

Most of the participants at STMR 3 (including us) had worked on projects with high ratios of testers to programmers, as many as 5 testers per programmer. Most of us had also worked on projects involving very low ratios, as few as 0-to-7 and 1-to-8. Some of the projects with high ratios had been successful, some not. Some of the projects with low ratios had been successful, some not. This corresponds with what we've been told by other managers, outside of STMR.

Why are some projects successful with very few testers while others need so many more?

Low Ratios of Testers to Programmers

Most testers have seen or worked in a test group that was flooded with work and pushed up against tight deadlines. These groups typically staff projects with relatively few testers per programmer. The work is high stress and the overall product quality will probably be low.

However, some projects are correctly staffed with low ratios of testers to programmers. In our experience, these projects generally involved programmers (and managers) who had high quality standards and who didn't rely on the test group to get the product right.

Projects with low ratios of testers to programmers might occur:

- routinely, in a company with a healthy culture whose projects normally succeed
- routinely, under challenging circumstances
- on a project-by-project basis based on special circumstances of that project

Healthy Culture

Healthy cultures that have successful projects with relatively few testers often have characteristics like these:

- The test group has low noise-to-work ratios. "Noise" includes wasted time arising out of organizational chaos or an oppressive work environment.
- Staff turnover in the testing and programming groups is probably low. It takes time for testers to become efficient with a product--time, for example, to gain expertise and to build trust with the programmers.
- The company focuses on hiring skilled, experienced testers rather than "bodies."
- There is a shared agreement on the role of the test group, and little need for ongoing reevaluation or justification of the role.
- There is trust and respect between programmers and testers, and members of either group will help the other become more productive (for example, by helping them build tools).

- Quality is seen as everyone's business. The company emphasizes individual accountability. The person who makes a bug is expected to fix it and to learn something from the experience. There is a low churn rate for bugs--they don't ping-pong between programmers and testers ("Can't reproduce this bug", "I can", "It's not a bug anyway", "Marketing says it is", etc.)
- The code coming into testing is clean, designed for testability, and has good debug support.
- There may be an extensive unit test library that the programmers rerun whenever they update the product with their changes. The result is that the code they give to testers has fewer regression errors and needs less regression testing (Beck, 2000)
- In general, there is an emphasis on prevention of defects and/or on early discovery of them in technical reviews (such as inspections).
- In general, there is an emphasis on reuse of reusable test materials and on intelligent use of test tools
- The expectation is that reproducible coding errors will be fixed. Testers spend relatively little time justifying their test cases or doing extensive troubleshooting and market research just to convince the programmers that an error is worth fixing.
- The culture is more solution-oriented than blame-oriented.

Challenging Circumstances

Some companies need much more testing than they conduct, but they might not do it because:

- The product might be so complex that it is extremely expensive to train new testers. New testers won't understand how to test the product, and they'll waste too much programmer time on unimportant bugs and misunderstandings.
- They may have decided that time to market is more important than finding and fixing defects.
- They are in a dominant market position in their niche and their customers will pay extra for maintenance and support. There is thus (until competitors appear) relatively little incentive to the company to find and fix defects before release.
- They believe that it's right and natural for people in high tech to work 80+ hour weeks for 6+ months. In their view, adding staff will reduce the free overtime without increasing total productivity.
- The testing group may be perceived as not contributing because they aren't writing code.
- Other members of the project team might believe that testing is easy. ("What's so tough about testing? Just run the program! I can find bugs just by installing it! In fact, we should just bring in a bunch of Kelly temps to do this.")

- Testers might be perceived as too expensive.
- Testers might be perceived as incompetent, counterproductive twits.
- The test manager might be perceived as a whiner who should use his staff more effectively.
- The test group's work might be perceived as poor, with an overemphasis on unimportant issues ("corner cases") and the superficial aspects of the product.
- The testing group may have little credibility. They are seen as politically motivated and being preoccupied with irrelevant tests (e.g. some extreme corner-case tests). Therefore they are not sufficiently funded.
- The relation between testers and programmers may be toxic, resulting in excessive turnover in the testing group.

Some companies will never develop respect for their testing staff, and will never staff the test groups appropriately, no matter how good the testers or test managers. But in many other companies, testing groups build their own reputations over time. Some testing groups work too hard to increase their power and control in a company and not hard enough to improve their credibility and their technical contribution. Down that road, we think, tight staffing, high turnover, and layoffs are inevitable.

Project Factors

To some degree independently of the corporate culture, some ***projects*** are likely to succeed with few testers because of factors specific to those projects. For example:

- The product might involve low risk. No one expects it to work well and failures won't harm anyone.
- There might be little time-to-market pressure.
- The product might come to the test team with few defects (perhaps because this particular project team paid a lot of attention to the design, did paired programming or did a lot of inspections, etc.)
- The code might be particularly easy to test or relevant test tools that the testers are familiar with might be readily available.
- There might be no need to certify this product, no need for extensive documentation of the tests or (except for bug reports) the test results, and no requirement for detailed evaluations of the final quality of this product.
- The testers might simply not have much work to do on this project because it is easy, reliable, intuitive, testable, etc.

High Ratios of Testers to Programmers

We've met testers who respond enthusiastically when they hear of a group that has a very high ratio of testers to programmers. The impression that they have expressed to us is that

such a high ratio must indicate a corporate commitment to quality, and a healthier lifestyle (less stress, less grinding overtime) for the testers.

In many cases, though, a high number of testers results from (and contributes to) dysfunction in the product development effort.

One of us worked on a project that had roughly three times as many testers as programmers by the end of the project. The programmers were under intense time pressure—and they couldn't help but notice the large pool of people next door just waiting to catch their mistakes. The result? The bug introduction rate skyrocketed. One programmer commented about a particularly buggy area of the program under test, "Oh, yeah. I knew there would be bugs there—I just didn't have time to look for them myself."

Programmers find the vast majority of defects in their own code before they turn it over for testing. When a programmer finds a bug in her own code, she can usually isolate it quickly. She doesn't have to spend much time documenting the bug, replicating the bug, tracking it, or arguing that it should be fixed. When programmers skimp on testing, testers must spend much more time per bug to find, isolate, report, track, and advocate the fix. And then the programmer wastes time translating a black box test result back to code.

We suggest that there is a significant waste of project resources whenever an error is found by a black box tester that could have been easily found by the programmer using traditional glass box unit testing techniques. Some of these errors will inevitably creep through to testing, but we think staffing and lifecycle models that encourage over-reliance on black box testers are pathological.

Having an army of testers can encourage a spiraling drop in productivity and quality. (We talk more about this in Hendrickson, 2001, and Kaner, Falk & Nguyen, 1993, Chapter 15).

The best solution for severely buggy code is not to add testers. The best solution might be to freeze (or even reduce) the size of the testing group while adding programmers. The programmers should fix and test code, not add even more buggy features.

Healthy Cultures

Some companies need more testers because of the market they are in or the technology they use. The examples below might describe the culture of the company or the circumstances of a particular project. Examples:

- Much more formal planning, documentation, and archiving of all artifacts of the testing effort is needed when developing safety-critical software. Heavy documentation might be required for other software because of regulatory agency interest or high litigation risk.
- Extensive documentation may also be needed for software that will be sold in its entirety to a customer, with the expectation that the customer will assume responsibility for future maintenance, support and enhancement.

- Some markets are particularly picky about fit and finish errors or are more likely to expect / demand technical support for problems that customers in other markets might seem small or easy to solve. If you are selling into that market, you'll probably do much more user interface testing and much more scenario testing.
- Extensive configuration testing is needed for software that must work on many platforms or support many different technologies or types of software or peripherals.
- Load testing is needed for software that is subject to bursts of peak usage.
- Some companies hire domain experts into testing or train several testers into domain expertise. These testers become knowledgeable advocates for customer satisfaction improvements and are particularly important in projects whose designs emerge over time.
- Some testing groups have a broad charter. Along with testing, they provide several other development services such as debugging, specification writing, benchmarking competing products, participation in code reviews, and so on. The broader the charter, the more people are needed to do the work.
- If the company relies on outsourced testing (this is sometimes a requirement of the customer's), there is substantial communication cost. The external testers need time to understand the product, the market, and the risks. They also need significant support (people to answer questions and documentation) from in-house testing staff.
- Software that involves a large number of components can be very complex and requires more testing than a simpler architecture.
- The development project might involve relatively little fresh code, but a large end product. The product might be knitted together from many externally written components or it might be an upgrade of an existing product. The testers will still have to do system testing (the less you trust the external code or the modification process, the more testing is needed). When the external components come from many sources, the test group may have to research, design and execute many different usage scenario tests in order to see how well the components work together to meet actual customer needs.

Challenging Circumstances

Some companies or projects back themselves into excessive testing staff sizes. For example:

- Some test groups don't understand domain testing or combinatorial testing, so they try to test too many values of too many variables in too many combinations.
- Some test groups rely on large numbers of low-skill testers. Manual execution of large sets of fully scripted test cases can be extremely labor-intensive, mind-numbing for testers and test case maintainers, and not very effective as a method of finding defects.

- Test groups that suffer high turnover are constantly in training mode. The staff may never get fully proficient with base technologies, available tools, or the software under test. Tasks that would be easy for a locally experienced tester might take a newcomer tremendously longer to understand and do.
- Testers may be given inadequate tools. Most testers need at least two computers, access to a configuration or replication lab, a decent bug tracking system, and various test automation tools. To the extent that the software under test runs on platforms for which there are few test tools, the testers have less opportunity to become efficient.
- Testing can be inefficient because the team doesn't use basic control procedures such as smoke tests and configuration management software.
- Software that was not designed for testability will be more difficult and thus more time consuming to test.
- Some corporate metrics projects waste time on the data collection, the data fudging (see Kaner, 2001; Hoffman, 2000), and the gossiping about the dummies in head office who rely on these stupid metrics. We are not suggesting that metrics efforts are necessarily worthless. We are saying that we have seen several such worthless efforts, and they create a lot of distraction.
- Programmers might focus entirely on implementing features. In some companies, testers write installers, do builds, write all the documentation, etc. This is not necessarily a bad thing. Instead it reflects a division of labor that might be wise under the circumstances but that must be factored into the budgets and staffing of both groups.
- Programming teams might send excessively buggy code into testing, perhaps because they are untrained in base technologies, or new to the project, or managed to implement features as quickly as possible, leaving the testing to testers. The worst case of this reflects a conscious decision that they don't have to test the code because they can count on the testers to find everything. Add more testers and the programmers do even less checking of their work. This can become a vicious spiral of increasing testing costs paired with declining quality (Hendrickson, 2001).

One-to-One Ratios of Testers to Programmers

Sometimes groups that describe their work in terms of one-to-one ratios really mean that they use paired teams of programmers and testers. For a given type of feature, a specific tester and a specific programmer work together, perhaps for several years. There are many advantages to this approach, but of course it can be problematic if the pair doesn't get along or if the pair develops too idiosyncratic a model of what things are acceptable to customers or reasonable to report and fix.

Final Notes on Ratio Factors

In analyzing the factors that support your company's reliance of a given balance of staffing between testers and other developers, you might find it useful to think in terms of categories of factors and to analyze the different category issues one at a time.

Rothman (2000) organized her paper in terms of 3 factors:

- Product—some are harder products to test than others.
- Project and its process—some projects employ better processes than others
- People and their skills—some developers and testers are more capable than others

We've found it useful to think in terms of these factors:

- Product under test
- Market expectations
- Project details (e.g. what resources are available when)
- Process (principles and procedures intended to govern the running of the project)
- Methodology (principles and procedures intended to govern the detailed implementation of the product or the development of product artifacts)
- Test infrastructure
- People
- Partnerships between testers and other stakeholders
- Allocation of labor (responsibility for different tasks) between testers and programmers

We don't think this is the ultimate list. You might do well to generate your own. Our point is that if you are trying to understand your staffing situation, it can help to start by listing several different dimensions to consider. Considering them each in turn, alone or preferably in a brainstorming session with a small group, can lead to a broad and useful set of issues to consider.

ALLOCATION OF LABOR

The most important driver of the ratio of testers to programmers should be the allocation of labor between the groups. If testers take on tasks that go beyond the minimum essentials of black box testing, it will take more time or more testers to finish testing the software.

Before attempting to estimate how many testers you need to perform the job, you need a clear idea of what those testers are going to do. At a bare minimum, the testers will probably:

- Design tests
- Execute tests

- Report bugs

They will probably also spend time interpreting results, isolating bugs, regressing fixes, and performing other similar tasks.

In some organizations, the testers have a much broader range of responsibilities. For example, testers may also:

- Write requirements
- Participate in inspections and walkthroughs
- Compile the software
- Write installers
- Evaluate the reliability of components that the company is thinking of using in its software
- Provide technical support
- Provide risk assessments
- Collect and report statistical data (software metrics) about the project
- Build and maintain internal test-related tools such as the bug tracking system
- Configure and maintain programming-related tools, such as the source control system
- Archive the software
- Benchmark competing products
- Evaluate the significance of various hardware/software configurations in the marketplace (to inform their choices of configuration tests)
- Conduct usability tests
- Lead or audit efforts to comply with regulatory or industry standards (such as those published by SEI, ISO, IEEE, FDA, etc.)

We do not espouse a preferred division of labor in this paper. Different groups have different charters. Any of the tasks above might be appropriately assigned to a test group. There is nothing wrong with that, as long as the group is appropriately staffed for its tasks.

To determine how many testers you need for your tasks, start by listing the tasks that they will do and estimate, task by task, how much work is involved. (If you're not sure how to do this, Kaner, 1996, describes a task-by-task estimation approach.) The total number of staffed tester-hours should be based on this estimate. The ratio of this staff to the programming staff size will emerge as a result, not as a driver of proper staffing.

CLOSING COMMENTS

Ratios out of context are meaningless. Attempting to use industry figures for ratios is at best meaningless and more likely dangerous.

Testers often ask about industry standard ratios in order to use these numbers to justify a staff increase. To justify an increase in staff, we suggest that you argue from your tasks and your backlog of work, not for a given ratio.

Even if you have a backlog, adding testers won't necessarily help clear it. Many problems that drive down a test group's productivity cannot be solved by adding testers. For example, poor source control, blocking bugs, missing features, and designs that are inconsistent and undocumented are not going to be solved by doing more testing.

References

Beck, Kent (2000), *Extreme Programming*, Addison Wesley.

Collard, Ross (1999), "Testing & QA Staffing Levels: Internal IS Organizations", Collard & Company.

Hendrickson, Elisabeth (2001), "Better Testing--Worse Quality?", Proceedings of the International Conference on Software Management, San Diego, CA. <http://www.qualitytree.com/feature/btwq.pdf>

Hoffman, Doug (2000) "The Darker Side of Metrics", Proceedings of the 18th Pacific Northwest Software Quality Conference, Portland, OR.

Kaner, Cem (1996) "Negotiating Testing Resources: A Collaborative Approach", Proceedings of the Software Quality Week conference, San Francisco, CA.

Kaner, Cem (2001) "Measurement Issues & Software Testing", QUEST Conference Proceedings, Orlando, FL.

Kaner, Cem, Jack Falk, & Hung Quoc Nguyen (1993; republished 1999) *Testing Computer Software*, John Wiley & Sons.

Rothman, Johanna (2000), "It Depends: Deciding on the Correct Ratio of Developers to Testers," <http://www.testing.com/test-patterns/index.html>.

QW2001 Paper 6M1

Mr. Geert Pinxten
(I2B)

The Extended Product Quality Model: Dynamic Focussing On Those Quality

Key Points

- Radical Innovation
- Incremental Innovation
- The Product Life Cycle
- The Product Quality Model
- The Extended Product Quality model

Presentation Abstract

Software development today is characterised by a lot of innovation: business innovates, technology innovates. As innovation, by its definition always concerns novelty, inexperience and immaturity, assuring quality has never been so difficult. This presentation will present a framework based on the Product Quality Model of J. Rothmann [1] that could be applied by those that need to insure the quality of all these new technology software products and e-commerce solutions. This framework will bring quality assurance in ICT to a new level of professionalism and will also solve the problem of speed, time-to-market or product dynamics. The solution will put a dynamic focus on those quality attributes that really matter.

About the Author

Geert Pinxten is Managing Partner of I2B. He is in consulting business since 1995. His experiences are mainly located in 2 areas: ERP (SAP) and structured testing. As a certified SAP technical consultant Geert Pinxten was involved in the implementation of SAP systems in several large organisations.

As a Test Consultant for a Belgian consultancy company, Geert Pinxten has been involved in several types of activities. He has been performing Test Assessments in large organisations active in the world of telecommunication, micro-electronics and business software development. During these audits he refined the auditing method used. Geert Pinxten was involved in setting up test organisations and was as a project leader responsible for numerous test projects.

As a Development Co-ordinator Geert Pinxten was responsible for the continuous improvement of the test expert knowledge at his previous employer. These

activities ranged from developing new test training packages till the improvement of the test method and its implementation. Geert Pinxten holds a degree of Industrial Engineer, with a specialisation in Electronics and Information Technology.

Idea to Business

I2B

Consultancy for those who change the world

http://www.i2b.be

Idea to Business I2B

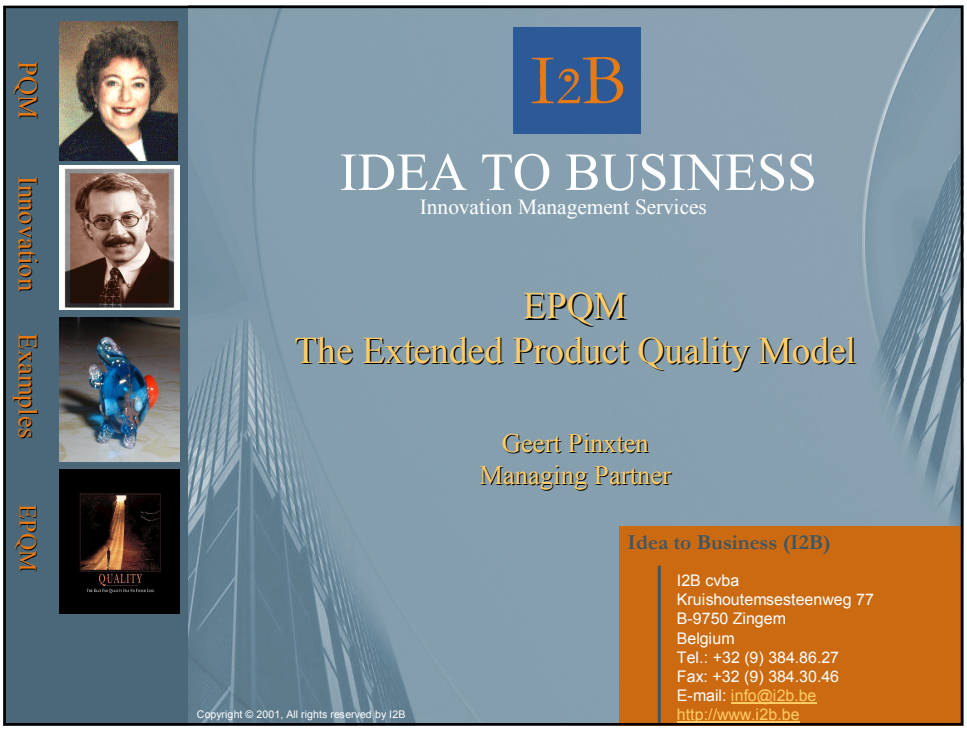
Update of this presentation exclusively available for the Quality Week audience on:
http://www.i2b.be/main/ipresentations

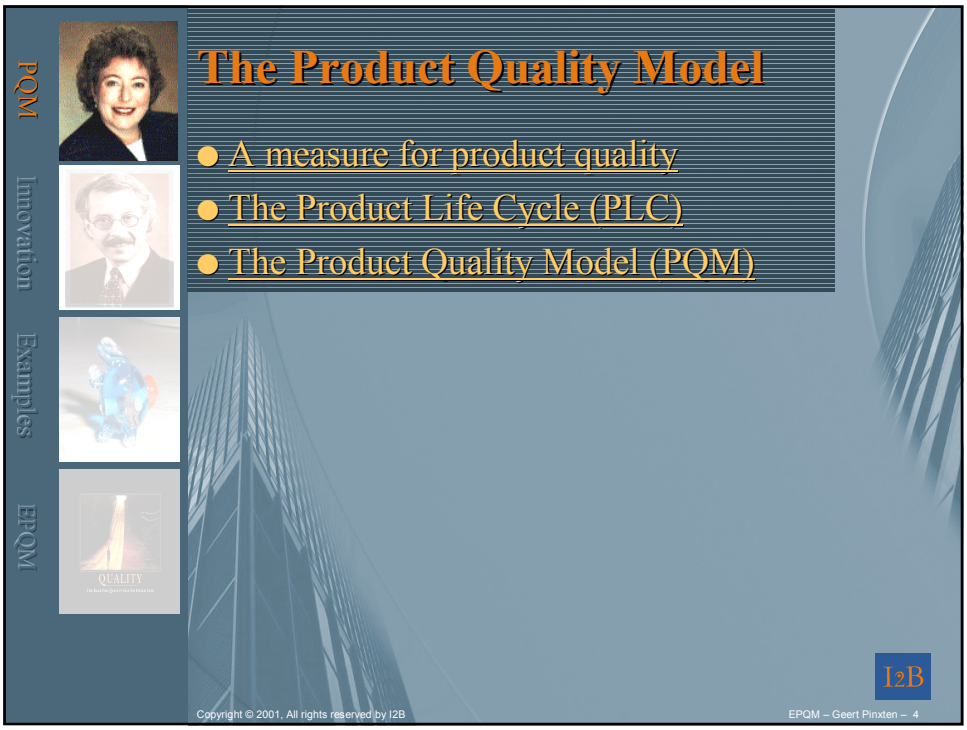
Login :
Password :

Consultancy for those who change the world

http://www.i2b.be

No Phones Thanks





A measure for product quality

PQM Innovation Examples EPQM I2B

Grady

3 common goals for software product development
- Minimize time to market
- Maximize customer satisfaction (Functionality)
- Minimize defects

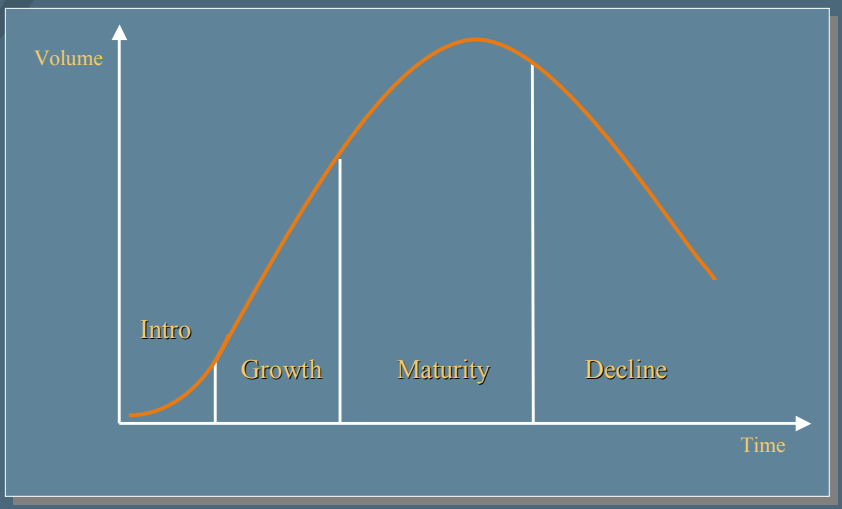
Weinberg

Product quality is value to the customer

Goto Start
Back
Goto End

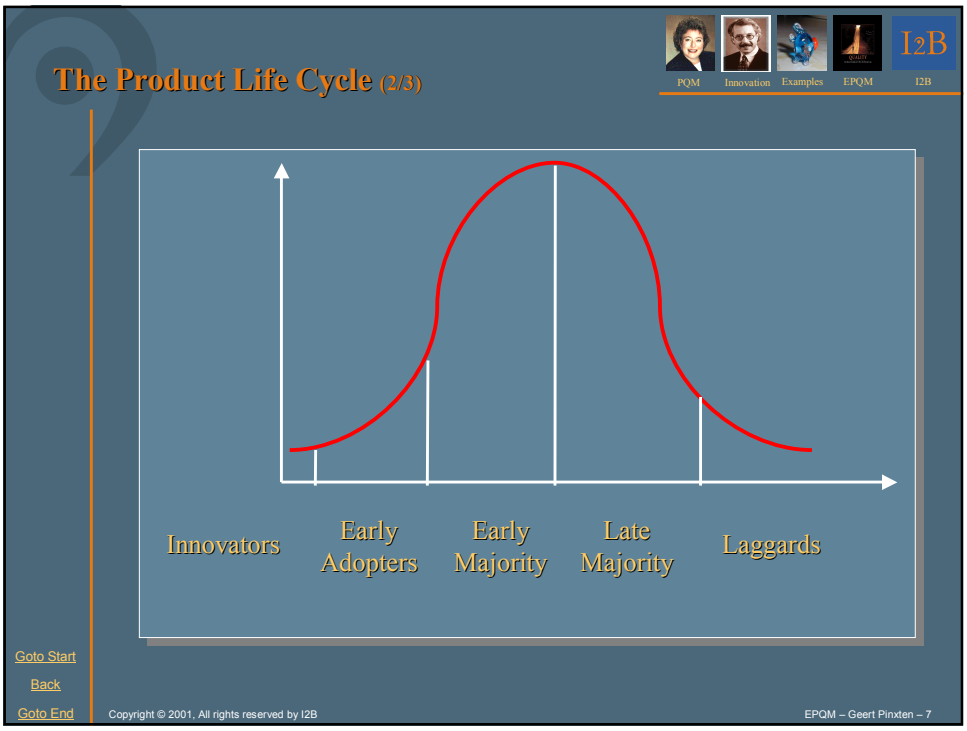
EPQM – Geert Pinxten – 5

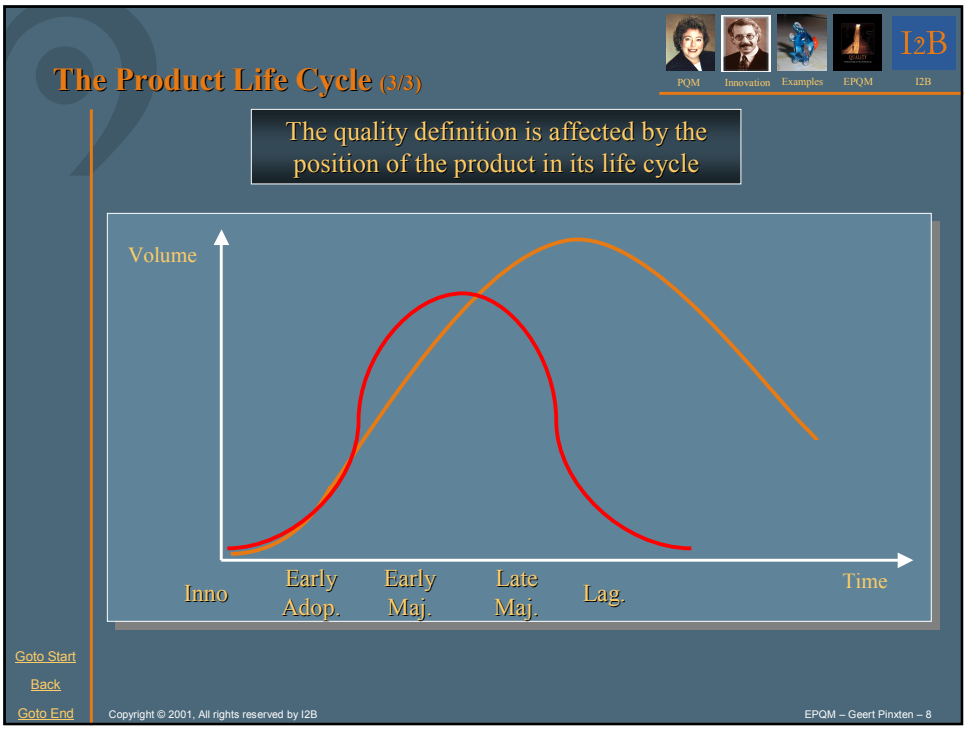
The Product Life Cycle (1/3)



Goto Start
Back
Goto End

EPQM – Geert Pinxten – 6





The Product Quality Model

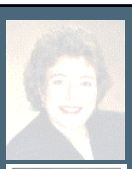
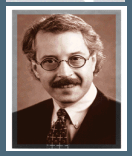
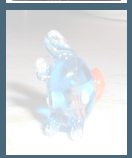
Product Live/ Market Pressure	Innovators	Early Adopters	Early Majority	Late Majority	Laggards
Time to market	High	High	Medium	Low	Low
Functio-nality	Low	Medium	Low	Medium	Medium
Minimize Defects	Medium	Low	High	High	High

J. Rothmann

Goto Start
Back
Goto End

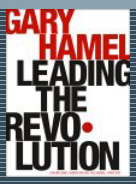
EPQM – Geert Pinxten – 9

PQM Innovation Examples EPQM

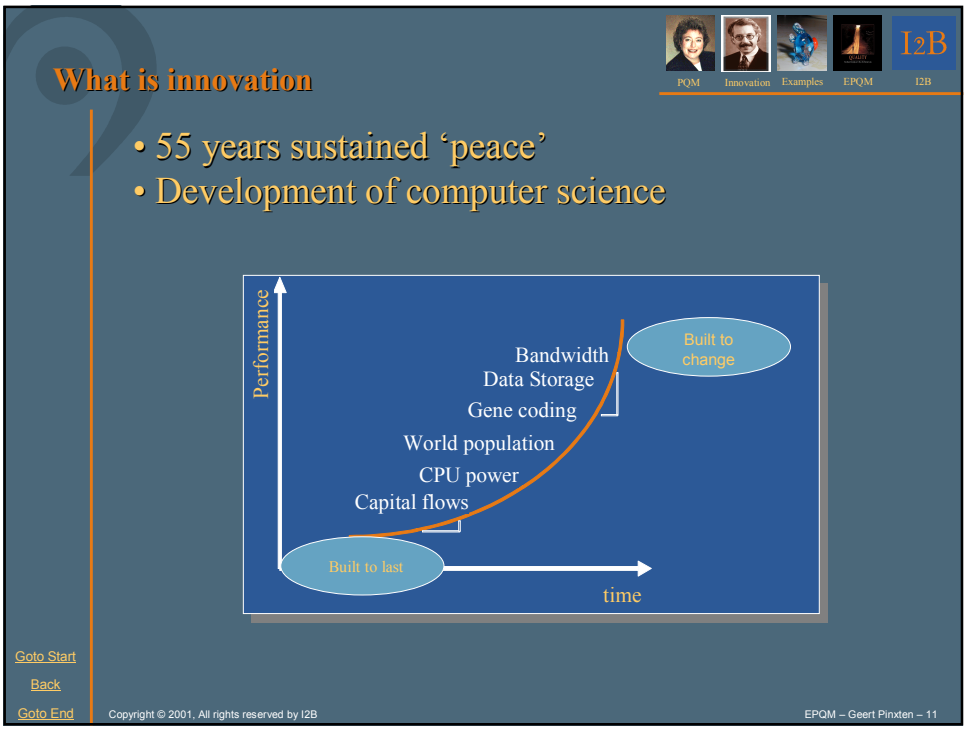

Innovation

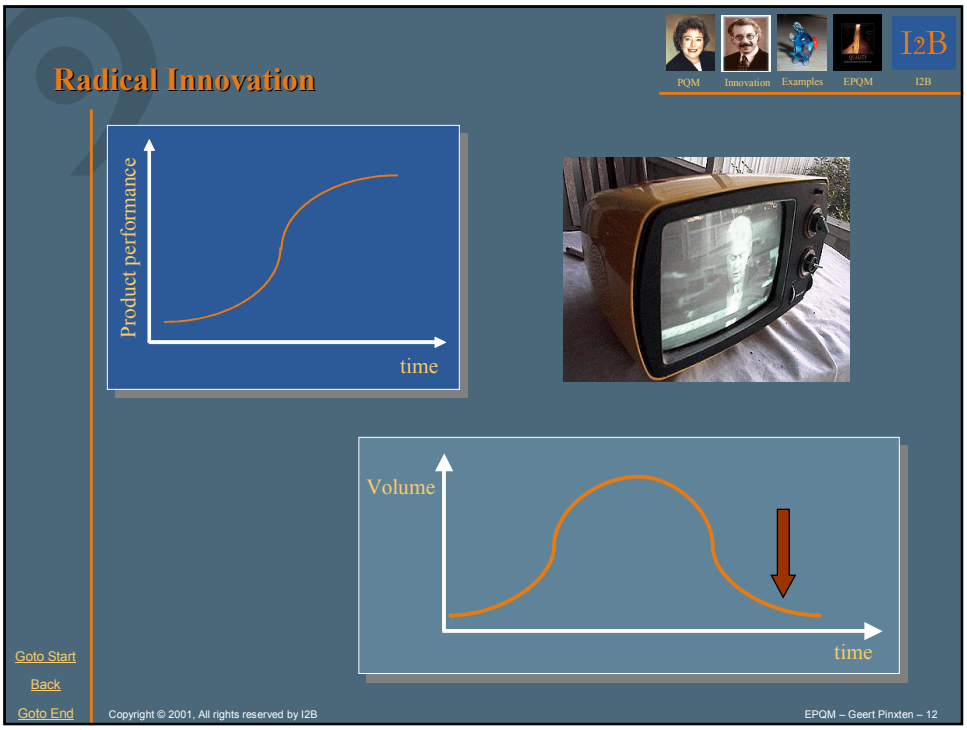
- What is Innovation
- Radical Innovation
- Incremental Innovation

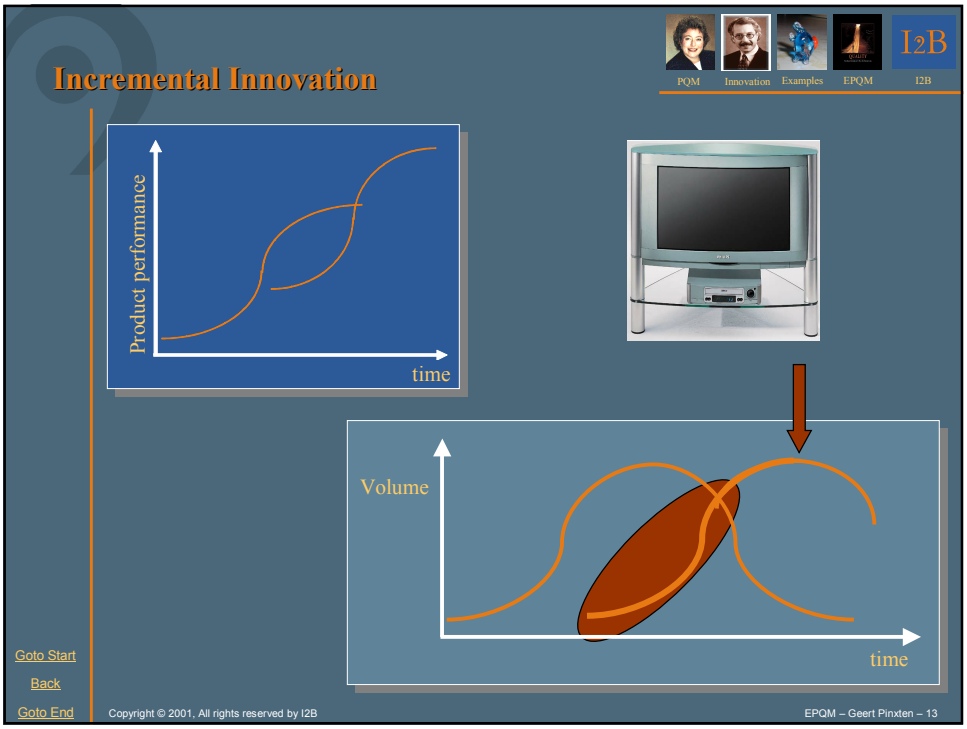


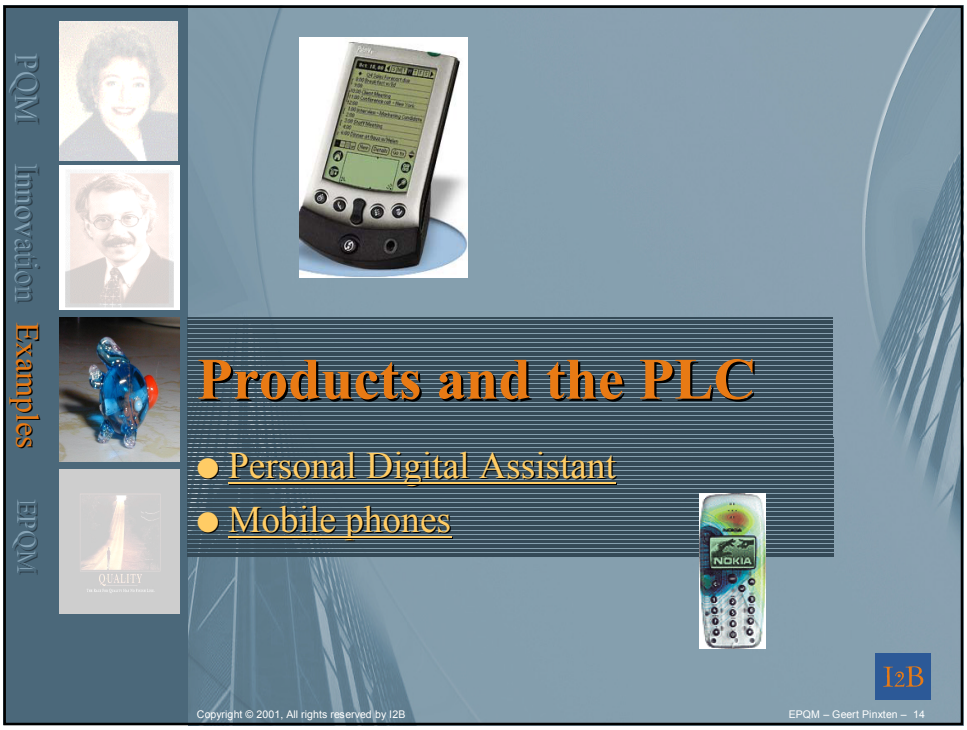
I2B

EPQM – Geert Pinxten – 10

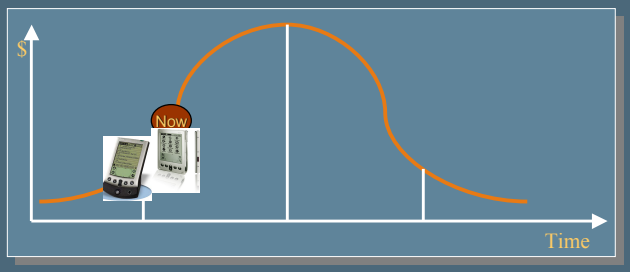








Personal Digital Assistant (PDA)



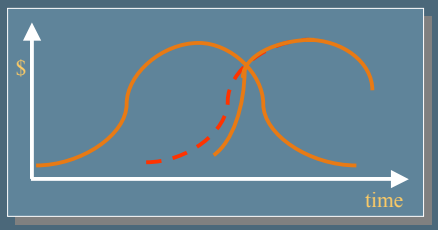
- Early Adopters
  - Time to Market: High
  - Features: Medium
  - Low defects: Low
- Characteristics
  - Copy
  - No innovation

Goto Start
Back
Goto End

Mobile phones

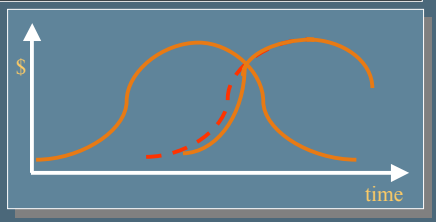
1. Main product characteristics slightly improving

Weight
Display
Size

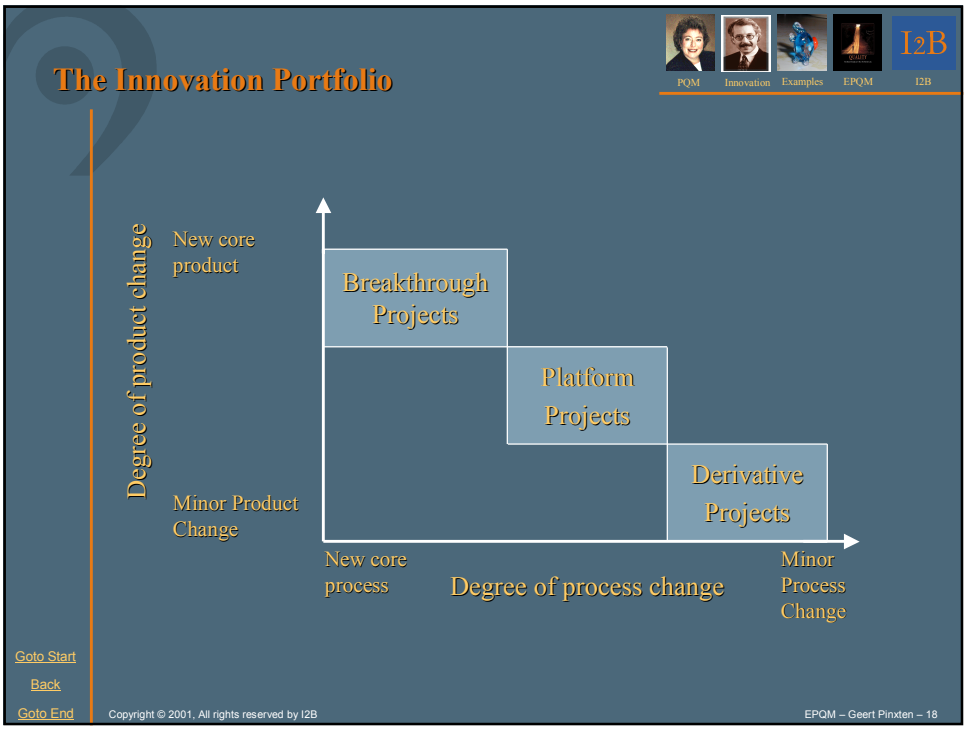


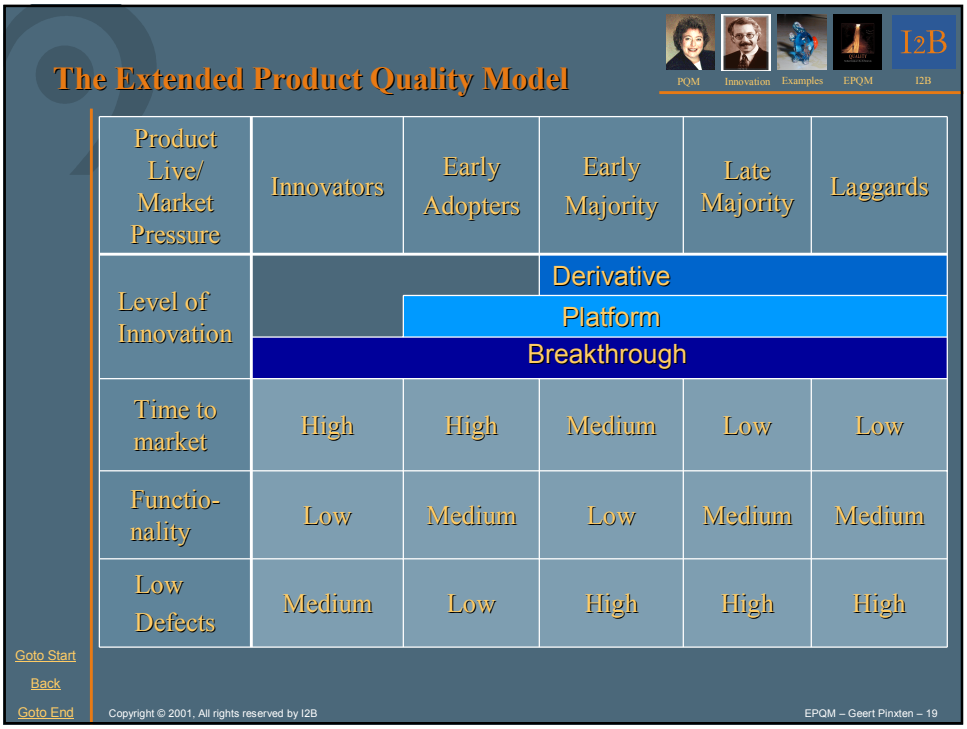
2. More important change in product characteristics

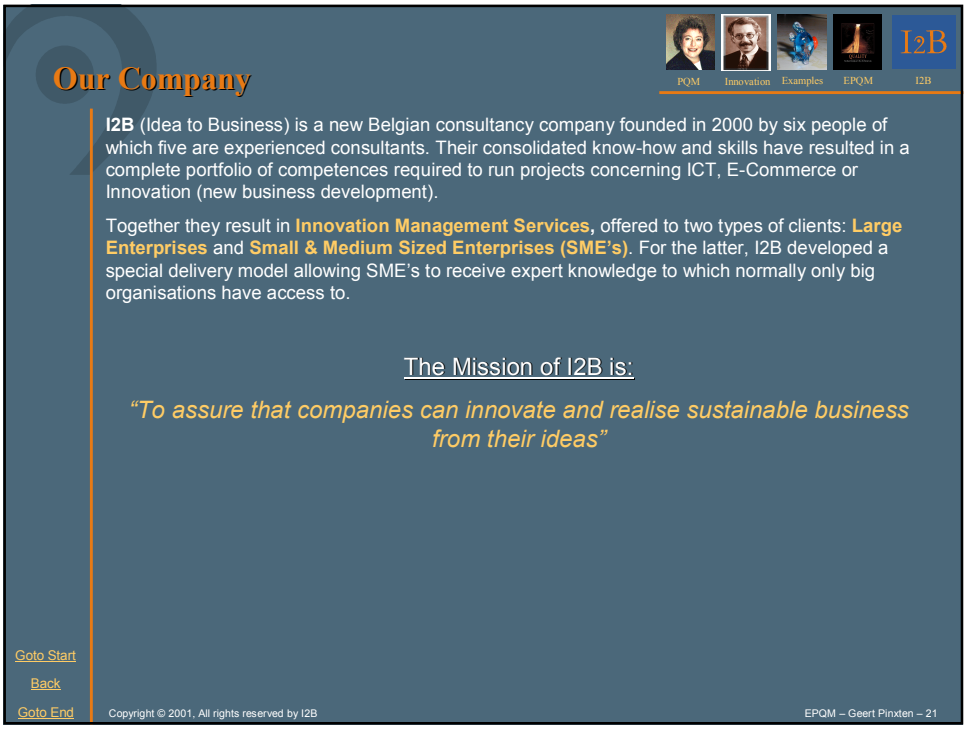
WAP
GPRS
I-MODE

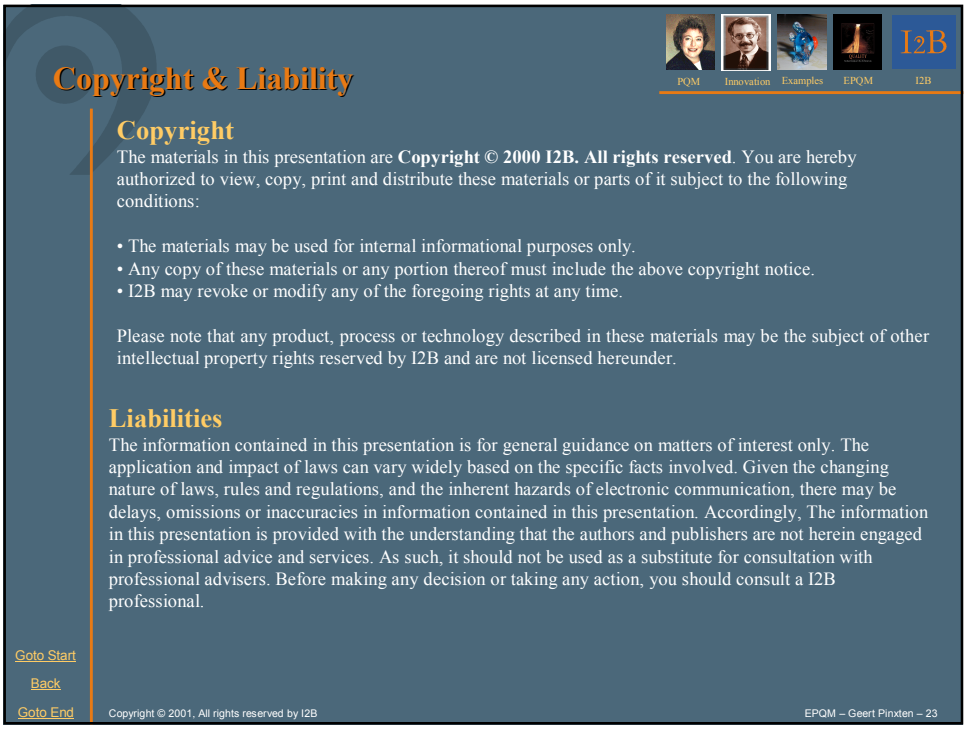


Goto Start
Back
Goto End








PQM
Innovation
Examples
EPQM
I2B
Copyright & Liability
Copyright
The materials in this presentation are Copyright © 2000 I2B. All rights reserved. You are hereby authorized to view, copy, print and distribute these materials or parts of it subject to the following conditions:
• The materials may be used for internal informational purposes only.
• Any copy of these materials or any portion thereof must include the above copyright notice.
• I2B may revoke or modify any of the foregoing rights at any time.
Please note that any product, process or technology described in these materials may be the subject of other intellectual property rights reserved by I2B and are not licensed hereunder.
Liabilities
The information contained in this presentation is for general guidance on matters of interest only. The application and impact of laws can vary widely based on the specific facts involved. Given the changing nature of laws, rules and regulations, and the inherent hazards of electronic communication, there may be delays, omissions or inaccuracies in information contained in this presentation. Accordingly, The information in this presentation is provided with the understanding that the authors and publishers are not herein engaged in professional advice and services. As such, it should not be used as a substitute for consultation with professional advisers. Before making any decision or taking any action, you should consult a I2B professional.
Goto Start
Back
Goto End
Copyright © 2001, All rights reserved by I2B
EPQM – Geert Pinxten – 23


Idea to Business
I2B
Consultancy for those who change the world

QW2001 Paper 6M2

Ms. Johanna Rothman
(The Rothman Consulting Group)

Using Requirements To Create Release Criteria

Key Points

- What release criteria are
- How to use release criteria during the project, not just at the end
- How to define release criteria
- Working with the project manager, to make sure the release criteria are used

Presentation Abstract

“We can’t stop to define what we’re going to do-we’re working on Internet Time”. You’ve heard that before. The push to move to the Internet or to continue your Internet business can feel overwhelming, especially when senior management is already demanding to make money with the product. Your project manager wants to know if the product is ready and when it will be released. You’d like to know what the heck you’re supposed to do, to know that you’ve done the necessary testing. If your organization is struggling and does not want to review all of the requirements in detail to get a project moving, you can create product release criteria to capture the critical requirements in a way that makes sense.

About the Author

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people.

Johanna publishes "Reflections", an acclaimed quarterly newsletter about managing product development. Johanna's handbook, "Hiring Technical People: A Guide to Hiring the Right People for the Job," has proved a boon to perplexed managers, as have her articles in Software Development, Cutter IT, IEEE Computer, Software Testing and Quality Engineering, and IEEE Software.

Johanna is the founder and principal of Rothman Consulting Group, Inc., and is a member of the clinical faculty of The Gordon Institute at Tufts University, a practical management degree program for engineers.

Using Requirements to Create Release Criteria

Johanna Rothman
Rothman Consulting Group, Inc.
www.jrothman.com
jr@jrothman.com

How Do You Know When the Software Is Ready to Release?

- "Is the software ready yet?"
- What does "done" mean?

Problems When You Don't Use Release Criteria

- You're responsible for deciding if the software is ready to release
- You know when you're supposed to release, you don't know how good the software has to be
- You can't easily explain why you're not done testing, you just know you're not done yet
- You are told to stop testing, the product is being released
- Release decisions are made by gut feel
- A variety of people can veto the release decision

 www.jrothman.com jr@jrothman.com 3

Develop Release Criteria

- What's critically important to this project
  - What's special about this release, for the company, for the customers
  - What does success mean?
- Quantify how to recognize success
- Get agreement from project team and senior management that you'll use release criteria to decide if the product is ready to release

 www.jrothman.com jr@jrothman.com 4

What Does Success Mean for This Project?

- What problem (or problems) is this project trying to solve?
- What are the project's requirements?
  - What are you being paid to deliver?
  - How good does it have to be?
  - When do the customers and the company want it?
  - What are the other constraints?
- What are the product's requirements?
- Then, plan and execute the testing portion of the project

About Success

- Success is what the customers will be able to do with the project when you're done with it
- Success has nothing to do with defects *per se*

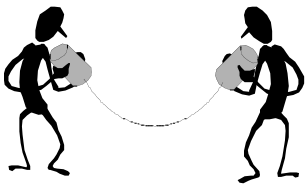
Use Context Free Questions to Define Success

- What does success look like?
- Why are these results desirable?
- What is the solution worth to you?
- What problems does this system solve?
- What problems could this system create?

 www.jrothman.com jr@jrothman.com 7

When You Ask Context Free Questions

- Ask why without asking *WHY*
  - Why might put people on the defensive
- Use How with care to avoid design decisions
- Have a conversation, not an interrogation



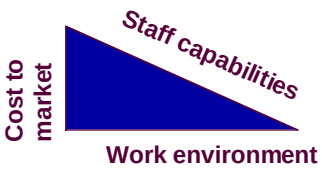
 www.jrothman.com jr@jrothman.com 8

What's Important for This Project?

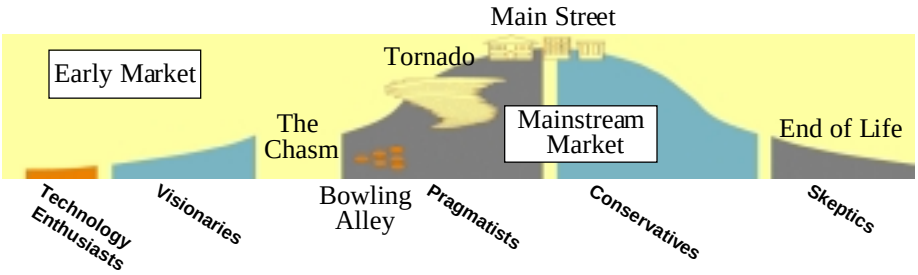
- Define quality
  - "Quality is value to someone" -- Weinberg
  - Each someone wants something different

Software Project Quality Perspectives

- Every project has requirements and constraints
- What do your customers care about the most?
  - Time to market
  - Feature set
  - Defect levels
- Internal Perspectives or Constraints: Your customers don't care about these. You do.
  - Cost to market
  - People and their capabilities
  - Work environment



Different Projects Have Different Customer Pressures for Quality



Introduction	Early Adopters	Mainstream	Late Majority	Skeptics
1. Time to Market	1. Time to Market	1. Low Defects	1. Low Defects	1. Low Defects
2. Feature Set	2. Feature Set	2. Time to Market	2. Feature Set	2. Feature Set
3. Low Defects	3. Low Defects	3. Feature Set	3. Time to Market	3. Time to Market

Rita's Story

- Originally just the date was the release criterion
- During one project, the product transitioned to the mainstream
  - Release was not well-received by customers

Updated Criteria for the Next Release

- Rita drafted new criteria, a balanced perspective on what was good enough to release
  - All code must compile and build for all platforms.
  - Zero high priority bugs.
  - For all open bugs, documentation in release notes with workarounds.
  - All planned QA tests run, at least 98 percent pass.
  - Number of open defects decreasing for last six weeks.
  - Feature *x* unit tested by developers, system tested by QA, verified with customers A, B before release.
  - All open defects evaluated by cross-functional team.
  - Ready to release by June 1.

Gain Consensus on Criteria

- PM explained about other pressures and two favorite customers
- Rita and the PM presented these criteria to the project team:
  - All code must compile and build for all platforms.
  - Zero high priority bugs.
  - For all open bugs, documentation in release notes with workarounds.
  - All planned QA tests run, at least 90 percent pass.
  - Number of open defects decreasing for last three weeks.
  - Feature *x* unit tested by developers, system tested by QA, verified with customers A, B before release.
  - Ready to release by June 1.

About Release Criteria

- Objective and measurable (SMART)
  - Specific, Measurable, Attainable, Relevant, Trackable
- Agreed to by entire project team and understood by senior management
- Reasonable
  - Release criteria are not the place for stretch goals
- If you have resistance to release criteria, discover why
  - Assumptions about how projects work
  - Fear of being measured
  - ….
- Help you resolve those assumptions and fears before you release

 www.jrothman.com jr@jrothman.com 15

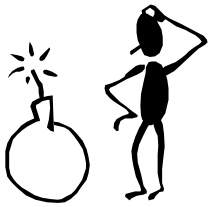
Other Ways to Gain Consensus on Release Criteria

- Drafting something in advance helps with the discussion
- Develop release criteria at a project team meeting
- Develop release criteria with the PM and then discuss with the project team
- Don't leave senior management out of the picture altogether

 www.jrothman.com jr@jrothman.com 16

Working with Senior Management

- Verify that senior management agrees with the release criteria
- Verify that senior management will use the criteria to make the release decision
  - Explain that vetos or early release decisions are inappropriate



Release Criteria can Illuminate Testing and Product Goals

- Must we meet this requirement by the requested release date?
- What is the effect on our customers if we do not meet this requirement by the release date?

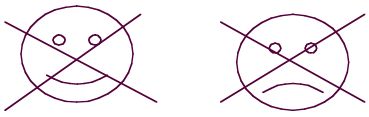
Using Release Criteria

- Evaluate the state of the project's "done-ness" throughout the entire project
- Early warning sign that you're not going to make it

 www.jrothman.com jr@jrothman.com 19

Release Criteria Are Not Partially Met

- Each criterion is either met or is not met
- I don't do happy faces or happy colors or happy anything
  - Don't confuse release criteria with a testing or project dashboard
- The project team evaluates each criterion, asking, have we met this criterion yet?



 www.jrothman.com jr@jrothman.com 20

When You Don't Meet the Release Criteria

- Be honest
- Make a conscious decision to release or not
- Make a conscious decision to change the criteria or not
- Decide what to do for the next time

Summary

- Say "Release it" with pride
  - You've met your commitment to your company and to your customers
- Plan your testing well, to take advantage of every minute available
- Use consensus so release criteria are not abandoned under pressure
- Let the PM take it from there

QW2001 Paper 7M1

Mr. Michael Ensminger
(PAR3 Communications)

Walk & Stagger Through Review Process

Key Points

- Peer Reviews
- Design Reviews
- Process Improvement

Presentation Abstract

The use of peer reviews, walkthroughs and inspections as a cost-effective method for achieving higher quality software is well documented. These techniques are not regularly used in many organizations, especially those that say they are operating on “Internet time”. At two Internet startups, I have used a modified, 2-phase walkthrough process to successfully deliver high quality software. The process proved successful, especially with teams without review experience. Contrary to what others in the organization were proposing, the teams using the process were able to deliver more quickly even with the increased “up front” work than other teams in the same organization.

The modified walkthrough process begins with the “Walk Through” (two words to distinguish it from the traditional use of the word walkthrough). During the walk through, the item being reviewed is examined in the order of machine execution or user experience in the “normal” course of operation. This first phase ensures that the item meets its stated goals. Once the “normal” course of action is completed, it is time to throw obstacles in the course of execution so that the team must stagger through what they just verified was correct. The stagger through phase examines error cases and “what if” scenarios. The division of labor - first focusing and algorithmic correctness and then looking at error handling and other situations - allows the team to stay focused, review each item with more or less the same thoroughness and does not waste team time fully inspecting an item which does not meet the basic goals. The paper details the team composition, process and scheduling details that a team lead must consider when implementing this approach.

A case study is presented for a project from Partes Corporation involving an Internet based data store and desktop machine analysis. The team, on the whole, was inexperienced in producing production quality code. The walk / stagger through process extended the design phase to probably twice the length that other

teams would declare design complete. However, the coding and testing phases progressed much faster than expected more than offsetting the time spent in design.

About the Author

Michael Ensminger is Director of Quality Assurance at PAR3 Communications based in Seattle, WA. Prior experience (both management and practitioner of test and development teams) includes Internet, shrink-wrap and niche retail banking software. He holds a M.S. in Computer Science from University of Texas at Dallas.

Walk and Stagger Through Review Process

Michael Ensminger
mensming@ieee.org
PAR3 Communications

Traditional Review Methodologies

- Desk Check
- Walkthrough
- Informal Review
- Formal Review
- Fagan Style Inspections

Challenges to Introducing Reviews to an Inexperienced Team

- Background knowledge of the team
- What are the responsibilities of the individual team members?
- When is the review done?
- What was the quality level of the review?

Walk Through Defined

The systematic review of a use case, design, code, etc. as the final system will execute it with a focus on the "normal" -- that is, non-error -- processing.

Stagger Through Defined

The systematic review of a use case, design, code, etc. as the final system will execute it with a focus on the "non-normal" -- that is error and uncommon execution paths -- processing.

Walk Through Process

1. Determine Focus and Scope
2. Select Review Team
3. Pre-Read
4. Review Meeting
5. Record Issues
6. Make Changes
7. Follow-up Before Stagger Through

Stagger Through Process

1. Prepare for Stagger Through Meeting
2. Review Meeting
3. Record Issues
4. Determine Next Steps
5. Make Changes

Case Study: Partes Corporation

- The Product
- The Team
- The Initial Schedule
- The Review Process
- Schedule Impact

Questions?

- Contact Information:

  Email: mensming@ieee.org

Walk and Stagger Through Review Process

Michael Ensminger
PAR3 Communications
(Email: mensming@ieee.org)

Abstract

Presents a two-phase review methodology. The first phase, the walk through, concentrates on the logical correctness of the functionality covered in the review item. The second phase, the stagger through, concentrates on error handling and unexpected flows through the review item. The method is particularly suited to teams without a lot of development experience. A case study from an internet company is described at the end of the paper.

Introduction

The use of peer reviews, walkthroughs and inspections as a cost-effective method of achieving higher quality software is well documented. [MYER79, BOEH87] These techniques are not widely used in many organizations, especially those who say they are operating in "internet time". I have used a modified, two-phase walkthrough process at two different internet startups. This process has allowed the team to achieve high-quality deliverables in "internet time" while other teams in the same environment were struggling with quality and schedule pressure.

The traditional definitions of walkthroughs and inspections are presented here for reference:

- Walkthrough – Loosely defined term where two or more team members review an item for the purpose of finding issues and improve the quality. (As opposed to trying to solve a known issue.) Usually, the author of the item leads the group through it in a lecture format. [MCCO96, FREE90]
- Inspections – Formal method of reviewing initially developed at IBM by Michael Fagan. In an inspection, each team member has a formal role such as moderator, scribe, etc. Usually, the focus of the inspection is narrowly focused to only one or two aspects. [FREE90]

The advantages and disadvantages of these methods have been studied thoroughly.

A small team may find it difficult to follow the rigor that an inspection requires. The leniency provided by the walkthrough may not provide the expected results. The following method works well with small teams, especially those with less professional experience. Non-development personnel may also fully participate in the process.

Walk Through and Stagger Through

In this paper, I present a two-phase review process. Phase 1 is labeled the walk through -- but differs from the traditional use of the term. Phase 2 is the stagger through -- to indicate that obstacles will be placed in the way and the team must stagger around them.

Phase 1: The Walk Through

The first phase, the walk through, is the systematic review of a use case, design, code, etc. as the final system will execute it with a focus on the "normal" -- that is, non-error -- processing. The walk through can be applied to any software artifact. The key is focusing on tracing through the artifact in the same order as will occur in the final system. Also, "normal" processing may include (and probably should) some common error cases. For example, a file not found error during a file open command is fairly common. How the system handles this condition should be examined in the walk through phase.

Overall the walk through process is as follows:

1. Select the focus of the walk and stagger through and determine the scope.
2. Determine the review team and any special assignments.
3. Pre-read.
4. During the actual walk through, trace through the logical flow of the artifact concentrating on normal processing.
5. Record any issues in the item(s) being reviewed. Note items that may be issues in items not covered in the current scope.
6. Give the team time to make corrections based on the discovered issues.
7. Follow-up before the stagger through.

Determine Focus and Scope

The focus and the scope of the walk and stagger through depend on many factors including time available for review, review item availability, item risk, etc. For a pilot program, a small items which can be covered in an hour or less should be selected. In no case, should an item be selected that cannot be covered in less than two hours.

For a small project, it is possible to follow the system from startup until termination. For most projects, this is unreasonable. Picking a unit whose invocation is well defined is a good starting point. This could be a feature, subfeature, method, routine, etc. When dividing up items for review, keep an eye out for items that do not lend themselves to this kind of examination. This may be the first sign of an issue for the item.

Multi-threading presents its own share of problems. Initially, examine thread execution in isolation. Then, expand the scope to include thread cooperation, concurrency issues, possible race conditions, other timing problems, etc.

Review Team and Special Assignments

The author of the item under review is one of the main participants. They will respond to many questions like "what will happen when..." or "where did this data come from?" Note that the role of the author differs from the traditional role of the author in a walkthrough. Instead of presenting the item in lecture style, the author is responding to questions from the team. When possible, the authors of items that will interact with the item under review will also be present. They should be especially aware of interface / integration issues as well as how errors will propagate through the system. A moderator should direct the session. The moderator will play the role of the users, the operating system, the data, etc. If the item under review contains a technology that is new or something that has been problematic in the past, one of the attendees should focus on that area. Quality assurance and test personnel can "virtually" execute the suite of test cases in this

environment. Operations personnel can determine the environmental needs of the system. Marketing can get a feel for various aspects of the final product and use that to craft their message. Junior personnel will gain knowledge of the system and the issues that must be addressed in a professional level deliverable. Management may also benefit by understanding the complexity that underlies seemingly simple functionality.

Pre-Read

Everyone's time will be better spent when all have done their homework. The pre-read is essential to the success of the walk through and stagger through. All participants should "actively" read the review material. By "actively", I mean reading with pen in hand and noting issues, questions and "gotcha" situations. The moderator should decide on the "flow" of the walk through, determining the starting point, order to take branches, which items to treat as black boxes, which items to descend into detail, etc. Other members of the team must achieve a general understanding of the item and drilling deeper based on their role.

It may be beneficial to submit issues and questions in advance. Especially if there is a large number, it will be time better spent for all to resolve these issues beforehand. Usually, the author and the moderator can determine whether revision should occur before continuing with the process.

Review Meeting

Once at the review meeting, the moderator instructs the author where in the material to being - usually at some point that results from an operating system event or user action. The author then leads the group through the logical flow from that point. All participants should be insuring the logical correctness and making notes of what could go bad when not taking the non-error path. Whenever a piece of data is accessed, it should be clear where the data comes from. When the author has a choice to branch through the code, the moderator will decide which direction the review should follow.

When another function / method / unit is called or referenced, the moderator must decide whether to treat it as a black box or to drilldown into the details. This decision should not be treated lightly. Drilldown will require time and effort. The team will need to know whether to prepare for drilling down into material that may not be apparent as the subject of the review. Treating the item as a black box may result in issues not being discovered. My rules of thumb are:

Coverage Criteria

There are varying levels of coverage. Which is used depends on the risk level and the time available.

- Statement coverage - each item is reviewed at least once
- Branch coverage - For each possible branch, traverse each path
- Predicate coverage - Consider all possible combinations of true / false values in a logical function.
- Handling loops - Try to skip the loop, execute once, twice, a typical number and the maximum number of times.

Stricter levels of coverage are available. See the references below for more information. [BEIZ90, BEIZ95, KANE99]

Coverage and Various Design Artifacts

Applying the above criteria is easily applied during design and code reviews. It is more difficult for other software development artifacts. For those documents written in a natural language, the criteria can be used as guidelines when the following are encountered:

- The words 'and' and 'or' are encountered in the document
- Statement containing the word 'if' or 'while'
- Statement contain the words 'for each' or 'every'
- Any item that contains an alternate way to accomplish an action

- If the item is in the scope of the review and has not been reviewed before, drilldown.
- If the item is out of the scope of the review and has been reviewed in a prior review, treat it as a black box. If it appears that the current input types were not covered in the previous review, a mini-review to drilldown in this area will need to be scheduled.
- If the item is out of the scope of the review and has not been reviewed, treat it as a black box. Pay close attention to the interfaces. Note the inputs and outputs that the current review item expects. During the review of this item, pull out the notes from the current review to insure that the item behaves as expected.

Many times it will be necessary to review an item several times. This is due to the fact that we are taking one logical path through the item. We will need to retrace our steps to take other logical branches through the item. Use general coverage criteria (see sidebar above) to decide which branch to take, focusing on the "normal" path.

Other activities may also be taking place during the review meeting. A traceability analysis is readily undertaken when one team member is marking which upstream item is covered during the current review item. Technical writing staff may be reviewing their documentation covering the material in the review, making notes as to items not covered or misrepresented. The creation of test cases, online help, error documentation, etc. is aided by identifying key areas brought to the surface early in the development cycle during the review. Just as important as what is reviewed is what isn't. If a portion of the review item is never "executed" this may indicate unnecessary functionality, "dead" code, or an error (the item really should have been covered…).

All participants should question the assumptions of the item currently being "executed" in the review. They should actively participate by asking questions that were developed during the pre-read or have just occurred to them. When appropriate, any issues should be raised. However, this is not the time to raise the "gotcha" issues -- save these for the stagger through.

Record Issues / Make Changes

A successful walk through will identify numerous issues. There should be a recorder in the review meeting. In some peer review methodologies, it is suggested that there be a scribe to record the issues who is not actively reviewing the material. While this allows all actively reviewing team members to concentrate on the material being reviewed (and not busy recording the issues), I believe that something is lost. Writing down the issue helps to solidify it in the mind of the writer. If the issue is unclear, the recorder can raise it at that time. If the recorder is an impartial scribe, they may miss the nuances of the issue. Therefore, I believe the author of the item should be responsible for recording the issues with the review item. The author's notes should be readily visible to the entire team. All other team members should take copious notes to keep everyone honest and to resolve ambiguities later on. Each team member will also use these notes during the stagger through meeting.

After the meeting, everyone who had issues identified in their work should start working on corrections before the stagger through meeting. This "updated" work will be used as the basis of the stagger through. This allows the changes to be reviewed by the team and see what implications arise from the changes made. Even those who did not have items identified in their deliverables may need to make updates. The review should be personally successful to them if issues identified in the review item shed light on potential issues in their own work.

Sufficient time should be given to make the necessary changes. However, too much time between the walk through and the stagger through meetings reduce the effectiveness of the process.

Therefore, the schedule should be aggressive to get the changes in. If the changes are so extensive that they cannot be made in a day or two, it may indicate that a new walk through is needed before the stagger through.

Follow-up before the Stagger Through

Preparation for the stagger through is similar to the preparation for the walk through review meeting. However, since the scope and the team are already selected, the preparation concentrates on the pre-read. Before the revised documents are available, the team members should review their notes and the original documents to identify items they want to clarify in the stagger through. Once the revised documents are available, the updated items should be pre-read again, focusing on the changes.

Since the focus of the stagger through are error and non-normal paths through the review item, the team members should allow their more sinister side to show. At this point, team members should identify those circumstances where the review item may break. [SHUL00] These "gotchas" will flesh out the stagger through.

Phase 2: The Stagger Through

The second phase, the stagger through, is the systematic review of a use case, design, code, etc. as the final system will execute it with a focus on the "non-normal" -- that is error and uncommon execution paths -- processing. The stagger through process is very similar to the process used in the walk through -- only the focus has shifted. In general, the following occurs.

1. Prepare for the Stagger Through (pre-read, review changes, create "gotchas")
2. Stagger through review meeting
3. Record issues
4. Determine next steps
5. Make changes, etc.

Preparation for the Stagger Through

After the walk through, the team begins to prepare for the stagger through as detailed in the section "Follow-up before the Stagger Through". There may be some additional preparation beyond the pre-read. During the walk through it may have become apparent that the review team did not have all of the knowledge necessary to perform an adequate review. In this case, the moderator may wish to expand the team. The new member(s) will need to come up to speed on the review items and what occurred in the walk through.

During the walk through, questions may be raised about items peripheral to the review that cannot be answered within the team. All effort should be made to find these answers before the review meeting. This will allow the team to have all of the knowledge needed to make the review effective. It also lets each team member know their input is important and will be followed up on.

Finally, the moderator should follow up with each team member and see if anyone has any suggestions on how to improve the next phase. Many times, it will be a suggestion not to spend as much time in one section. It is a judgment call whether to heed this suggestion. As always, the moderator must determine whether the change will allow the team to effectively find additional

issues. The moderator should follow up on all suggestions and publicize any changes before the stagger through review meeting.

Stagger Through Review Meeting

The stagger through meeting follows the logical control flow through the review item as was done during the walk through. Normal processing which has not changed since the walk through can be covered quickly. It should not be skipped entirely since team members may have discovered new issues with understanding they gained during and after the walk through review meeting. Normal processing that has changed since the walk through session should be reviewed more thoroughly.

The effectiveness of the stagger through lies in the change of focus. Hopefully, the entire team is satisfied that the item under review "works". (If this is not the case, perhaps the walk through or the changes requested were not detailed enough. The moderator must take care of this situation, perhaps be convening a new walk through before the stagger through may continue.) The team now shifts to trying to "break" this review item that they are convinced "works" by concentrating on the non-normal control flow through the review item. Each team member has prepared a list of "gotchas" to spring on the review item at the appropriate time.

The first few errors will usually take longer to work through than subsequent ones. During the first few errors, the team will be examining the error handling philosophy and how well it deals with fatal, severe, routine and trivial errors. Care should be taken to examine error propagation and following the error to the response to the user. Once the team is satisfied with the general error handling philosophy, the team can determine whether the error will be handled (by the generic error mechanism or by some special case logic) and whether this behavior is appropriate. So what kind of errors should be examined? Here are a few:

Data

- Nonsensical user inputs
- Corrupt input files
- Database or file system errors
- Boundary cases
- Duplicated data (in unique situations)

Runtime Issues

- Failed system function calls
- Out of memory
- Out of disk space
- Multi-threading issues - lock and race conditions
- Recursion issues
- Services not available

Security

- User or system has inadequate permissions
- System under attack

Date Related
(Many business systems perform special processing based on the time of the year)
- First day of the year
- Last day of the year
- Last day of a leap year
- Leap day
- First day of the month
- Last day of the month
- First day of the quarter
- Last day of the quarter
- Transition to daylight savings time
- Transition to standard time

Interface Related
- Interface not available
- Interface returns an error
- Interface fails to return or timeouts

Infrastructure Related
- Network down
- Site down or unreachable
- Firewall / Proxy server issues
- Device offline or unreachable
- Normal maintenance activities

These are just a few of the types of errors that can be examined during a stagger through that are often missed during other types of reviews.

Record Issues

The record issues step is identical to the same step in the walk through process. If the walk through / stagger through is to be considered a formal review, a report to management will need to be created.

Determine Next Steps

The moderator and the team must decide what the next steps should be for the review items. If the stagger through identifies substantial issues that must be addressed, a new walk through / stagger through process will need to be scheduled. If the issues identified are more local in scope, another stagger through review meeting may be scheduled. If all of the issues are determined to be minor, another review session is probably not necessary. It is recommended that at least one team member read through the revised review item to make sure that all of the issues are adequately addressed.

Make Changes

The output of the stagger through will be a list of issues that need to be resolved. The author of the review item will need to address these issues in his deliverable. Even more so than the walk through, team members should have a list of items that need to be addressed in their own, yet-to-be-reviewed deliverables. Especially early in the development phase when many items have not been reviewed, stagger throughs will identify deficiencies in error handling and propagation. Consistency issues in how these items are dealt with will need to be addressed.

Case Study: Partes Corporate

The most successful implementation I have seen of this process was several years ago at Partes Corporation. Partes Corporation (since acquired by EDGAR Online) created software to enable the retrieval and analysis of SEC filing data in many different formats. I was recruited into the company as manager of quality. Besides the CTO and myself, most of the development team members had less than a year of professional development experience. The product was an internet enabled add-in to Excel to allow users to retrieve parsed SEC data from the Partes web accessible datastore and perform various analyses on the downloaded data.

A spiral development lifecycle was used with at least three iterations planned. The first iteration was to enable the selection and downloading of files into an Excel workbook. The subsequent iterations would add basic time series analysis functionality followed by more complex analysis. For the first cycle, the development plan called for about a third of the time spent in design, third in implementation and a third in testing. All of the staff had the theoretical knowledge to perform these tasks but perhaps not the practical experience to pull it off. We decided to reduce the risk by using the walk through / stagger through process on all design artifacts. The plan was to walk through the design in the same order as execution. For the first iteration, we decided on following logical flow through the application: Launch the Excel add-in, select a company, download some filings, save the Excel workbook and the load a saved analysis.

The first session involved explaining the process and starting with the initialization of the add-in. This particular session did not last long as the team had left out the mundane details of initialization. The existing design jumped right into the details of the functionality. The team's expectation of what was needed was reset and they were given a couple of days to make corrections. Once we got past the initialization review (which we would return to time and time again each time a data element was first referenced) the fun began.

In the session where we were tracing the flow of selecting a company we decided to drilldown to every element. This required many developers to be ready to have their items reviewed. The process of loading the company search dialog and constructing the request to the datastore took two sessions. Many interface issues were discovered along with some missing functionality. All in all, the walk through portion of selecting a company took nearly eight times longer than we expected. The team felt a large sense of accomplishment after completing the walk through and had a much better idea of what was required for professional application. After making the necessary corrections identified in the walk through, we were ready to start the stagger through.

Soon after introducing the first "non-normal" processing condition, it was apparent to all that a large portion of the design was missing. What error handling that was designed did not take into account the need to propagate the error up the calling chain (and eventually to the user

interface). There was also no clear agreement among the team which errors were fatal and which were recoverable. The team requested a week to rework the error handling strategy for the product and integrate it into the design. When the reviews reconvened, the increase in the quality of the design was immediately evident. What had not been taken into account was the shear number of things that could go wrong in the application. All developers were seen immediately expanding their design to take into account new classes of errors that were exposed in the review meeting.

This process continued through the logical flow. We would often revisit reviewed items as questions arose. Sometimes these would result in revisions to the previously reviewed items. This was especially true of the initialization routine. The design process ended up taking nearly twice as long as we expected. The schedule had been adjusted using the same proportions mentioned above. We extended coding and testing by about twice as long. To our surprise, coding went incredibly fast. This was due primarily to the fact that most on the incongruities that would become apparent in coding has already been discussed and resolved during the review meeting. Testing went smoothly. There were few integration issues. Most of the changes introduced during testing were related to usability or special circumstances arising from the SEC data. In subsequent projects with this team, we did not drilldown to the same level of detail. This was due to the increase in experience. We did get burned a few times and wished we had gone into more detail.

Conclusion

The walk through / stagger through review methodology can be effectively used to verify the correctness and completeness of a development artifact. It is especially useful for teams with little development experience, experience in a technology or experience working together. The segregation between algorithmic correctness and error handling allows the team to focus on one at the exclusion of the other. This results in a more thorough review then trying to concentrate on both at the same time. Traditional review techniques can be supplemented with this approach based on team experience and risk analysis.

References

BEIZ90 Beizer, Boris. Software Testing Techniques, 2nd Edition. New York: Van Nostrand Reihnold, 1990.

BEIZ95 Beizer, Boris. Black-Box Testing, New York: John Wiley & Sons, Inc., 1995.

BOEH87 Boehm, Barry. "Industrial Software Metrics Top 10 List." *IEEE Software*. (v4, n9, September 1987), pp. 84-85.

FREE90 Freedman, Daniel P., and Weinberg, Gerald M. Handbook of Walkthroughs, Inspections, and Technical Reviews. New York: Dorset House Publishing Co. Inc., 1990.

KANE99 Kaner, Cem, Falk, Jack, and Nguyen, Hung Quoc. Testing Computer Software, 2nd Edition. New York: Jogn Wiley & Sons, Inc. 1999.

MCCO96 McConnell, Steve. Rapid Development. Redmond, WA: Microsoft Press, 1996.

MYER79 Myers, Glenford J. The Art of Software Testing. New York: John Wiley & Sons, 1979.

SHUL00 Shull, Forrest, Rus, Ioana, and Basili, Victor. "How Perspective Based Reading Can Improve Requirements Inspections." *IEEE Computer*. (v33, n7, July 2000), pp. 73-79.

About the Author

Michael Ensminger is Director of Quality Assurance at PAR3 Communications based in Seattle, WA. Prior experience (both management and practitioner of test and development teams) includes Internet, shrink-wrap and niche retail banking software. He holds a M.S. in Computer Science from University of Texas at Dallas.

QW2001 Paper 7M2

Prof. Warren Harrison, Dr. David Raffo & Dr. John Settle
(Portland State University)

Process Improvement As A Capital Investment

Key Points

- Economics of Process Improvement
- Making the Business Case
- Return on Investment

Presentation Abstract

Firms invest in process improvements in order to benefit from increased productivity sometime in the future. However, there are a large number of alternate investment opportunities, while at the same time, the available budget is often constrained. To make things even more complicated, each alternative may result in different cost savings or income over different periods of time with different levels of risk. Thus, we're faced with the question: "in which opportunity should we invest?" We present a well-accepted method of budgeting for capital expenditures from the financial community, and apply it to software process improvements.

About the Author

Warren Harrison is Professor of Computer Science at Portland State University. His research interests include both software engineering and internet technologies. Professor Harrison's software engineering research includes return on investment for process improvements, software quality assurance, software measurement, and empirical studies of software engineering. He is an active member of the software engineering research community, serving as Editor-in-Chief of the Software Quality Journal and co-EIC with Vic Basili and Lionel Briand of Empirical Software Engineering, as well as being involved with the organizing committees of numerous international conferences and workshops each year. His PhD is from Oregon State University.

Dr. Raffo completed his Ph.D. at Carnegie Mellon University in 1995. His research involves developing a theoretical framework and associated quantitative

techniques to predict the impact of potential process changes on cost, project schedule, and software quality. These concepts and theories have been field tested at leading software development organizations.

John W. Settle has a B.A. from Pomona College and a B.S., M.B.A. and Ph.D. from University of Washington. He is also a Chartered Financial Analyst (CFA). Dr. Settle teaches corporate finance, investments and portfolio management. He has done research in the areas of mergers and valuation issues. His current research interests are in investor psychology, market returns, and applied corporate financial concepts.

Process Improvement as a Capital Investment

Warren Harrison
David Raffo
John Settle
Portland State University

Quality Week 2001
May 29-June 1, 2001

Process Improvement

- Productivity Gains
- Reduced Rework
- Constrained Resources
  - Limited pool of resources for process improvement
  - Can't afford to do everything
  - Compare options and make decisions to get the "biggest bang for the buck"
  - "Invest Wisely

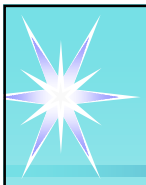
The Value of an Investment

- **Returns** - how much do you get back?
  - Increased productivity, reduced Rework
- **Timing** - when do you get it back?
  - During development? At release? During production?
- **Risk** - how likely is it that you really will get it back?
  - Uncertain benefits. Lack of data.
- Use discounted *cash flow techniques* to normalize timing and risk

Discounted Cash Flow Techniques

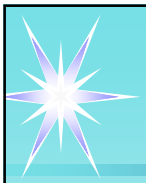
- A design inspection procedure will save 500 hours in rework effort 12 months in the future
- How many of "today's hours" are those 500 hours one year in the future worth if our discount factor is 1% per month?

$$PV = \frac{return}{(1+k)^n} = \frac{500}{(1+0.01)^{12}} = 444$$



What Do We Really Know?

- Can't know for certain that a formal inspection will find 70% of the defects or rework cost will be 25 hours per defect
- The present value analysis would change greatly if post-release defects really only cost 2X instead of 100X to fix …
- The returns from process improvements are *risky*



Financial Risk and Process Improvement

- Financial Risk - *volatility of the return - how much is the return likely to vary from your prediction?*
- If the outcome of a process improvement is uncertain, it is less desirable than a process improvement where the outcome is known
- Herbsleb observed productivity gains from 9% to 67% with a median of 35%.

Including Risk in the Discount Rate

- Effective Discount Rate:

$$k = r_f + \phi$$

  - r_f - risk free rate
  - ϕ - risk premium

- The risk premium adjusts the required return for the volatility of expected returns

Reuse and Financial Risk

Components Reused	Probability	Return in Two Years
0	5%	0
10	10%	$25,000
20	70%	$50,000
30	10%	$75,000
40	5%	$100,000
Expected Return	$19,462 ($\sigma$) 0.39 (*cv*)	$50,000

Financial Risk of an Alternative - CASE Tools

Utilization	Probability	Return in One Year
None at All	10%	0
Modest	15%	$25,000
Moderate	50%	$50,000
Heavy	15%	$75,000
Exclusive	10%	$100,000
Expected Return	$26,352 ($\sigma$) 0.52 (*cv*)	$50,000

Relative Risk Between Options

- The *relative* risk between two alternative projects can be approximated by the ratio of the coefficients of variation:
  - $Risk_{reuse} = 0.39$
  - $Risk_{case} = 0.52$
  - $\lambda_{reuse} = Risk_{reuse} / Risk_{case} = 75\%$

Why is One Option Riskier than Another?

- Financial Risk is the property of the outcome differing from what you expected
- A given process improvement may be more risky just because it is
- … but it may just be because we don't have very much information to go on … additional information may lead to less financial risk

Reference Risk Premiums

- Organizations may differ in how much return they want for a given amount of risk
- Establish a *baseline project* that reflects the price of risk for your organization - provides a "reference risk premium" for a given amount of risk - adjust for other projects
- Relative Risks are expressed as a percentage of the reference risk premium

Deriving a Discount Rate

- The *relative* risk between a proposed project and the baseline project is determined and a specific risk premium established:
  - $\phi_{option} = \lambda_{option} * \phi_{baseline}$

Computing Project Value Given a Baseline Project

- Let the CASE Tool initiative represent the reference risk premium, and set it's premium at 20%
- Compute the value of the reuse initiative

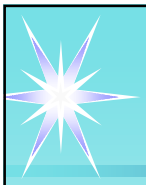
$$PV_{reuse} = 50{,}000/(1+r_f+\phi_{reuse}*\lambda_{reuse})$$
$$= 50{,}000/(1+0.05+0.20*0.75)^2$$
$$= \$34{,}722$$

Important Capabilities

- In Order to do Risk-Adjusted Discounting you need to be able to:
  - predict the expected returns
  - predict the timing of the expected returns
  - assess the variability involved in the expected returns
  - establish a reference risk premium

Using NPV in Valuing Information

- Some process improvements are inherently risky
- Some process improvements appear risky because we don't have any data
- We can use the change in Net Present Value between a project with and without data to assign value to the data



Future Work

- the Value of Mitigating Financial Risk with Better Information
- The "cost of capital" for software process improvements
- The utility of "resources" - how to trade-off hours against releasing on time or safe operation

Process Improvement as a Capital Investment: *Risks and Deferred Paybacks*

Warren Harrison[*]
David Raffo[†]
John Settle[‡]
Portland State University

Abstract

Firms invest in process improvements in order to benefit from decreased costs and/or increased productivity sometime in the future. However, there are a large number of alternate improvements available, each of which may yield different levels of cost savings. To make things even more complicated, different alternatives may result in different savings over different periods of time with different levels of risk. Thus, we're faced with the question: "*in which opportunity should we invest?"* We present a well-accepted method of budgeting for capital expenditures from the financial community, and apply it to software process improvements.

Introduction

The motivation for process improvements is typically reduced cost and/or increased productivity. For instance, early prevention of defects reduces the cost of rework later in the lifecycle and the practice of reuse improves productivity [Basili,1994; Dion,1993; Lipke,1992; McGarry,1993; Wohlwend,1993]. Of course, both the costs and returns of any particular process improvement can vary greatly. The costs of software process improvement have been reported to range between $490 and $8,862 per engineer, per year, with productivity gains ranging from 9% to 67% [Herbsleb,1994; Jones,1996]**.** The assumption is that the returns will outweigh the costs of implementing the process improvement.

Many different process improvements have been proposed. However, many organizations can only afford (or only choose) to implement one or two options at a time. Therefore, it is important to be able to evaluate and compare different alternatives. The analysis and comparison of projects is known as *Capital Budgeting.* Most contemporary Capital Budgeting techniques utilize the concept of the “Time Value of Money”. This has been addressed in various aspects of software engineering in the past, such as quality assurance [Slaughter 98] and software maintenance [Vienneau 95]. Perhaps the most common technique is referred to as *Present Value* (PV). Briefly, the idea of Present Value analysis involves “discounting” a future cash flow at some discount rate, resulting in the expected value of in today’s dollars.

[*] Department of Computer Science, Portland State University, Portland, OR 97207-0751
[†] School of Business, Portland State University, Portland, OR 97207-0751
[‡] School of Business, Portland State University, Portland, OR 97207-0751

Assessing Value Using Discounted Cash Flow

To illustrate, assume a *hypothetical* reuse initiative from which we expect to receive $50,000 in benefits due to increased productivity two years later when the reusable components are actually utilized. Applying a 10% discount rate to the benefits expected to accrue two years in the future, we obtain the following present value (PV):

$$\begin{aligned} PV &= PV_{2,10\%}(50{,}000) \\ &= \$41{,}322 \end{aligned}$$

This represents the current value of introducing reuse given $50,000 in increased productivity two years in the future (this is somewhat unrealistic since the yield from reuse would ostensibly accrue over many years).

Adjusting the value of the effort by the amount of time necessary to start seeing a return from our investment is useful. If we wish to compare the reuse initiative with another opportunity with a different pattern of returns, this gives us a means by which we can compare the two. For instance, instead of a reuse initiative, we might choose to acquire a CASE tool that yields $50,000 in increased productivity after only a year:

$$\begin{aligned} PV &= PV_{1,10\%}(50{,}000) \\ &= \$45{,}455 \end{aligned}$$

Thus, the present value of the CASE tool adoption is $45,455 as compared to a present value of $41,322 for the reuse initiative. Given these hypothetical figures, the CASE tool would be the preferred investment of the two.

Financial Risk and Discounting

Addressing the issue of future returns is only a part of discounting returns. The cost of capital reflected in the discount rate also typically incorporates the impact of “risk” on the value of a specific investment. For instance, in the earlier example, if the CASE tool investment only had a 50% chance of yielding $100,000 in increased productivity, but the reuse option was a sure thing we would expect the analysis to be different. This is because the discount rate should actually reflect the return expected from an investment of a particular risk. This way, the discount rate that would be used in the analysis of a very risky investment will typically be much higher than a “sure thing”.

Discount rates are comprised of a base, “risk-free” rate which reflects the value of money to be received in the future with certainty, and a “risk premium” reflecting the uncertainty of the future pay-off. The more variable the return, the more risky the investment, the higher the risk premium and thus, the greater the discount rate.

In the case of software process improvement, the risk derives from the fact that the benefits of process improvement are not a “sure thing”. The improvements may not be

well-accepted by the developers, or if accepted, they may not be implemented correctly. Even if well-accepted and implemented appropriately, the project may not respond to a given process improvement due to a lack of opportunity. For instance, there may be few latent requirement errors to find, so a requirements review process itself may yield little improvement over not reviewing the requirements at all.

Risk means not only the risk of not having a positive return, but also the uncertainty in terms of *how much* of a return we will get. Herbsleb [Herbsleb,1994] observed productivity gains from software process improvement ranging from 9% to 67% with a median gain of 35%. Clearly, this is riskier than some other investment that yields a future return with certainty. Risk is addressed within Present Value Analysis, by using a discount rate *k*, which is related to the uncertainty of the project:

$$k = r_f + \phi$$

that is, the risk-free rate of return r_f (say the rate of return on Treasury Bills, currently about 5%), plus a risk premium ϕ, which is a function of the *variability of the return*.

Naturally, an important issue is how one measures the "variability of the return". In order to do this, we have to recognize that it is possible for different scenarios to occur in the future which impact the return of the process improvement. Also, we assume that we can, with some degree of accuracy, predict the likelihood of these scenarios occurring.

Given the hypothetical reuse initiative we have (somehow) determined that each instance of reuse is worth $2,500 (annualized) in increased productivity. We believe that there is:

- a 5% chance that none of the components will be reused,
- a 10% chance that 10 of the components will be reused,
- a 70% chance that 20 of the components will be reused,
- a 10% chance that 30 of the components will be reused and
- a 5% chance that 40 of the components will be reused.

based on data from other organizations. This is summarized as follows:

Components Reused	Probability	Return
0	5%	0
10	10%	25,000
20	70%	50,000
30	10%	75,000
40	5%	100,000

Table 1

The "expected return" is computed as the sum of each potential return multiplied by the probability of its occurrence. Thus, the expected return for this scenario is $50,000, with a standard deviation of $19,462. The coefficient of variation (standard deviation divided

by the expected value), 0.39, can be viewed as a measure of the volatility of the return. This information can be considered a relative measure of the risk, $Risk_{reuse}$ that would yield a risk-based discount rate k_{reuse}.

On the other hand, consider adopting a CASE tool. Assuming that the return from the tool is a function of its adoption by the developers, we might hypothesize the following cases, annualized returns, and probabilities of their occurrence:

Utilization	Probability	Return
None at All	10%	0
Modest	15%	25,000
Moderate	50%	50,000
Heavy	15%	75,000
Exclusive	10%	100,000

Table 2

The "expected return" is still $50,000, however the uncertainty of the outcome has been increased, with the new standard deviation being $26,352, for a coefficient of variation of 0.52, which implies this is a much riskier project.

Thus:

$$Risk_{reuse} = 0.39$$
$$Risk_{case} = 0.52$$

and

$$\lambda_{reuse}. = Risk_{reuse} / Risk_{case} = 75\%$$

The parameter λ_{reuse} implies that the reuse initiative is approximately 75% as risky as the CASE tool purchase. Consequently, the risk premium for the reuse initiative's discount rate should be 75% of the risk premium for the CASE tool purchase's discount rate.

Given a risk-free rate of 5%, a 20% risk premium for the CASE Tool purchase and a one year payoff for both, the PV of the two proposed process improvements can give us some direction in selecting between the two. Because the reuse initiative is 75% as risky as the CASE Tool purchase a 15% risk premium will be assigned. With these assumptions, we compute the following PV:

$$PV_{reuse} = 50{,}000 / [1+ 0.05 + 0.15]^{1}$$
$$= \$41{,}666$$

and

$$PV_{case} = 50{,}000 / [1+ 0.05 + 0.20]^{1}$$
$$= \$40{,}000$$

Using these assumptions, the reuse initiative would be the preferred option.

A serious issue is the base “risk premium” that we glibly “assumed” was 20% for the reference project. If we are simply interested in ranking options with similar return patterns, the specific base risk premium is of less importance. However, if the timing of the returns vary, or if we’re trying to establish if the process improvement actually sustains the cost of capital to the organization, this must be established with more care.

Necessary Capabilities

In order to use a contemporary Capital Budgeting approach to compare process improvement opportunities, four capabilities must be present:

1. the ability to predict the expected return of the process improvement,
2. the ability to predict the timing of the expected returns,
3. the ability to quantitatively assess the “risk” involved in the process improvement returns,
4. the ability to establish a reference risk premium (or, equivalently, a reference cost of capital) for the firm's "typical" or reference project.

While more exhaustive treatments of each of these are beyond the scope of this paper, we will briefly address these capabilities here.

Predicting Returns and Timing

Predicting the returns and their timing can be more difficult than predicting the costs. Nevertheless, there is no dearth of attempts at assessing the benefits of various software process improvements. For instance, in [McGibbon, 1996], the benefits of inspections (as well as the benefits of a variety of other process improvements) are modeled as a function of rework costs RC:

$$RC = R \bullet \sum d_i t_i$$

where d_I is the number of defects detected in phase i, t_I is the amount of time in hours to detect and fix an error in phase i. And R is the average hourly rate to find and fix an error.

The key is either good historical data within your organization, or access to “industry standards” (which won’t quite fit). For instance, O’Neill [1995] observes that inspections will detect 10-20 errors per thousand lines of code. Thus, with our 100,000 line project, we can expect 1,000-2,000 errors to be found. What would the cost of these errors be if they were either (a) found later or (b) not found until after the product was released. This is particularly frustrating because little industry data exists as to the actual costs of correcting defects later in the lifecycle, and organizations seldom tend to keep track of such data.

It is also not always clear what the timing of the returns would be – for instance, should the benefit of finding an error early through inspections accrue when the error is found? When the product is undergoing testing (which is when the error might otherwise be found and corrected)? When the product is released? When the rework would actually occur? As our earlier examples have shown, deferring the returns can have a big impact on the perceived value of the process improvement when making investment decisions.

Assessing Risk

Anyone who has ever spent time looking at past process data understands that real outcomes seldom fit the nice, mathematical prediction models developed by researchers. However, these models often provide a good basis for an expected outcome, with potential outcomes being clustered around that point. A good source of data on past experiences can go a long way towards understanding what the clustering looks like. Is it a peaked, low-risk distribution like the reuse initiative data showed earlier, or a flat, high-risk distribution like the CASE Tool purchase?

Lacking quantitative, organization-specific historical data, other sources of information can be tapped to assess risk. For instance, consultants, small, ad hoc experiments, expert judgement, etc.

The Reference Risk Premium

It is presumed in this treatment that the firm is already using capital budgeting techniques, and has a basic understanding of what its reference cost of capital is. Determination of a firm cost of capital, although a critical step in deciding on investments and deserving of attention in software development contexts, is beyond the scope of this paper.

The Role of Metrics Repositories in Establishing Costs, Returns and Risk

From this brief discussion, it is clear that the capabilities needed for capital budgeting of software process improvements are highly dependent on data. It is interesting to note that the decreased risk due to good historical data from a metrics repository can actually be used to quantify the value of a metrics initiative within an organization.

Summary

In this paper, we have briefly described the application of traditional, time and risk based capital budgeting techniques to software process improvements. As we saw, these techniques can be valuable in choosing among alternatives. The major constraint to applying these techniques is access to the capabilities we discussed. These are all highly dependent on historical data describing past experiences within the organization.

References

- [Curtis,1995] Curtis, W., "Building a Cost-Benefit Case for Software Process Improvement," Notes from Tutorial given at the Seventh Software Engineering Process Group Conference, Boston, MA, May 1995.
- [Dion,1993] Dion, R., "Process Improvement and the Corporate Balance Sheet," *IEEE Softwar*e, July 1993, pp. 28-35.
- [McGibbon, 1996] Thomas McGibbon, "A Business Case for Software Process Improvement", Data Analysis Center for Software State-of-the-Art Report, prepared for Rome Laboratory, September 30, 1996
- (WWW URL: http://www.dacs.dtic.mil/techs/roi.soar/soar.html).
- [Hayes,1995] Hayes, W., Zubrow, D., "Moving On Up: Data and Experience Doing CMM-Based Software Process Improvement," Presentation at the *Seventh Software Engineering Process Group Conference*, Boston, MA, May 23, 1995.
- [Herbsleb,1994] Herbsleb, J., Zubrow, D., Siegel, J., Rozum, J., Carleton, A., "Software Process Improvement:State of the Payoff," *American Programmer*, Vol. 7 no. 9, September 1994, pp. 2-12.
- [Jones,1996] Jones, C., "The Pragmatics of Software Process Improvements," *Software Process Newsletter* of the *Software Engineering Technical Council Newsletter*, No. 5, Winter 1996, pp. 1-4.
- [Lipke,1992] Lipke, W., Butler, K., "Software Process Improvement: A Success Story," *CrossTal*k, Number 38, November 1992, pp. 29-31, 39.
- [O'Neill,1989] O'Neill, Don. Software Inspections Course and Lab. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1989.
- [O'Neill,1996] O'Neill, Don. "National Software Quality Experiment: Results 1992-1996." Proceedings of the Eighth Annual Software Technology Conference. Salt Lake City, UT, April 21-26, 1996. Hill Air Force Base, UT: Software Technology Support Center, 1996.
- [Slaughter,1998]. S. Slaughter, D. Harter and M. Krishnan, "Evaluating the Cost of Software Quality", *Communications of the ACM*, August 1998, pp 67-73.
- [Vienneau,1995]. R. Vienneau, "The Present Value of Software Maintenance", *Journal of Parametrics*, April 1995, pp. 18-36

QW2001 Paper 8M1

Ms. Sandy Sweeney
(Compuware Corporation)

Risky Business -- Adding Risk Assessment To The Test Planning Process

Key Points

- Move away from the 'we did the best we could' dysfunctional model of testing
- When picking and choosing the tests you can pull off in the allotted time - how can you know you are picking the right ones?
- Getting everyone - not just the test group - to understand and be involved in making these tradeoffs

Presentation Abstract

Testing is clearly a tool for risk management. The ability to use the results of test execution as a measure of the (relative) quality of an application under test is obvious. However, the typical metrics and measures are implemented during the Test Execution phase and look at the application as a whole. In the real world, however, it is clear that not all parts of an application are equally important, or equally buggy and that Test Managers are often frustrated by time and resource constraints that put them at a disadvantage in completing their assessment of product quality.

About the Author

Ms. Sweeney has been a Software Engineering Professional since 1976. She has been involved all aspects of systems engineering and software development, including management. She has functioned as Programmer, Analyst, Designer, Tester, Quality Assurance Engineer, and Project Manager for a wide range of companies in the Manufacturing, Financial, Health/Medical, Transportation, Retail/Wholesale and Telecommunications industries. She has extensive experience in Software Engineering methods and practices and has been involved in software development on a wide range of hardware and software platforms.

Risky Business

Adding Risk Assessment to the Test Planning Process

A presentation for Quality Week 2001
San Francisco, CA
May 31, 2001

Sandy Sweeney, Compuware
QAArchitect, Testing Senior Specialist
sandy_sweeney@compuware.com

What is Risk?

- **Something bad that might happen**
  - ***Probability***
- **Something that would have a bad effect if it did happen**
  - ***Consequences***

Risk Exposure = Probability * Consequences

Testing as Risk Management

- **Typically Manage to Schedule Risks**
  - **Determine Scenarios**
  - **Assess and Rank Risks**
  - **Risk of not implementing**
- **Generally don't Manage to Quality Risk**
  - **Risk of implementing**

Quality Risk = Risk Exposure of Implementing

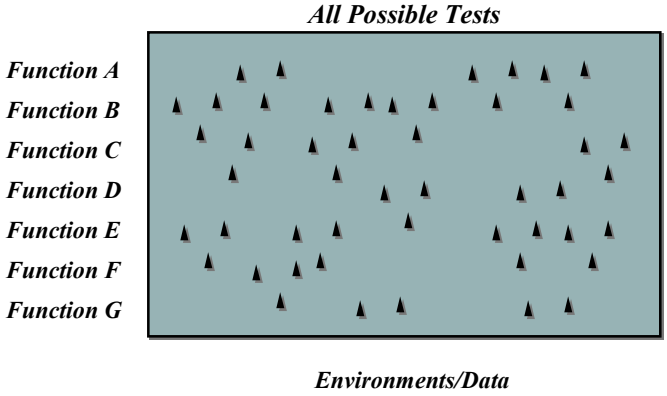
Measurements in Testing

- **Coverage (Code/Test)**
  - **Tries to assess probability**
- **Defects**
  - **Tries to assess consequences**
- **The hole --**
  - **How to measure the exposure of undiscovered defects**

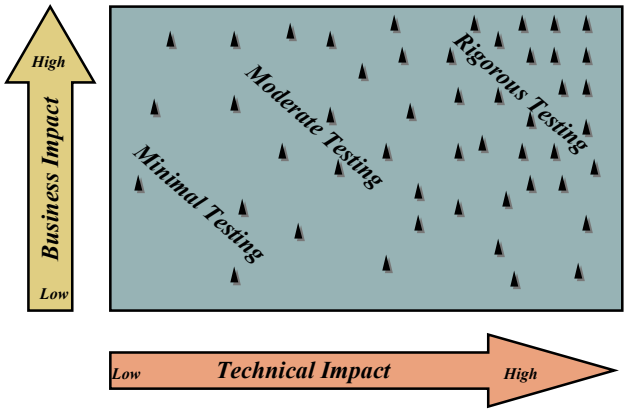
Filling the hole

- **Risk Assessment in the Test Planning Process**
  - **Determine areas of greatest risk before any testing starts**
  - **Build a Risk Profile**
  - **Use the Risk Profile as a Topographical Map**

Non-Risk Assessed Testing



Risk Based Testing



 7

Risk Assessment in Test Planning

- **Need a process that is quick and easy**
- **Matrix/Table based**
- **Estimation and assignment of number is 'relative' not 'absolute'**
- **Working for order of magnitude – not pinpoint precision**
- **Profile will emerge through collaboration and communication**

 8

Risk Profile

- **Risk Factors**
  - **Predictors of Risk**
- **Risk Weights**
  - **Balances the Risk Predictors based on experience**

 9

Determining Risk Factors

- **Mix Probability and Risk**
- **Consider all groups**
  - **Users**
  - **Developers**
  - **Testing**
- **Ask other groups for suggested factors**

 10

Examples of Risk Factors

- 'Age' of the element
- Complexity of the element
- Cross-element integrations
- Ties to Business Goals
- Number of Users
- Prior History of Defects
- History of Element in Production
- Maturity of the Process
- Politics of the Organization
- Experience with the Technology

Risk Weights

- Not all factors predict risk equally
- When first assigned just take a stab
  - don't expect perfection the first pass

Rank each Factor for each Element

- **This is collaboration, investigation and negotiation**
- **'Blank Sheet' can be a problem**
- **Focus attention on one or two areas**
- **Use what you did as an example**
- **It does get easier over time – on the first pass just give it your best shot**

 13

Do the Math

- **Score = Weighted Sum**
  - **Sum all (Factor Rank * Factor Weight)**
- **Index = Weighted Average**
  - **Score / Average of Weights**
- ***An 'automated' solution can make the math much easier***

 14

Check your work

- **Check the assumptions of your model**
  - **Look at 2-3 elements for 'gut feel'**
- **Have someone else look at 2-3 other ones**
  - **Build buy-in to the answers**

Typical Problems with Model

- **Risk Weights can skew the results**
  - **Double check the assumptions on it's predictive ability**
- **Not everything is critical**
  - **Too many of the same values in a column make the ranking meaningless**

Use the Model for Test Planning

- Items with high risk index should be tested as early as possible
- Items with high risk index should have the most test cases
- Negotiate to get high risk index items developed and delivered to test as early as possible
- Develop more stringent entry and exit criteria for high risk items
- Don't totally ignore medium and low risk items – test according to risk

 17

Profile helps re-plan for change

- **Low risk items may be able to be sacrificed when there is schedule slip**
- **Have a clear road-map agreed to across the board**
- **Re-sizing is risk-based**

 18

Profile is a Key Test Asset

- **Provides a History of the planning**
- **Re-useable on the next project**

Questions?

Sandy Sweeney, Compuware
QAArchitect, Testing Senior Specialist
sandy_sweeney@compuware.com

Risky Business

Adding Risk Assessment to the Test Planning Process

A presentation for Quality Week 2001
San Francisco, CA
May 31, 2001

Sandy Sweeney, Compuware
QAArchitect, Testing Senior Specialist
sandy_sweeney@compuware.com

Testing is clearly a tool for risk management. The ability to use the results of test execution as a measure of the (relative) quality of an application under test is obvious. However, the typical metrics and measures implemented during the Test Execution phase cannot provide a complete risk profile – due to some problems with the metrics as well as the tendency to look at the application as a whole. In the real world, however, it is clear that not all parts of an application are equally important or equally buggy and that Test Managers are often frustrated by time and resource constraints that put them at a disadvantage in completing their assessment of product quality.

This paper outlines a practical technique that has been used on numerous projects to insert a secondary risk assessment technique into the Test Planning process.

What is Risk?

A Risk is typically thought of as 'something bad that might happen' or 'something that would have a bad effect if it did happen'. These common sense thoughts of risk cover the two aspects of Risk Exposure – the *probability* that something will happen and the *impact* (consequences/cost/size) of the resulting problem.

Risk Management in software projects revolves around managing the many "What will we do if …?" situations that might occur throughout the project in an attempt to keep the project on schedule. What will we do if the hardware does not come in on time? What will we do if we can't get enough experienced programmers?

The first step of managing these risk scenarios is to assess them and understand which of the many possible scenarios warrant further attention. Assessment involves the assignment of a Risk Exposure value to the scenario. Risk Exposure is the product of the probability that the scenario will happen and the cost (consequences/size) of it happening. By multiplying probability and cost we get the 'probable-cost' of the problem. With this 'probable cost' we can build the appropriate plan of attack for each scenario based on how much it might be worth to prevent the problem:

(1) The scenario can be ignored because it has a real low probable-cost

These are the 'So what?' scenarios. These are either very low probability events or situations that can be dealt with cheaply and easily when they occur. For these scenarios, the cost of developing a special plan to prevent or deal with them generally is more than the risk exposure of the situation itself.

(2) A risk mitigation plan can be developed to reduce the exposure and/or implement an alternative 'Plan B' if it occurs

Because Risk Exposure is made up of equal parts probability and consequences, a risk mitigation plan may put steps in place to reduce either the probability or the consequences of the scenario.

(3) An alternate approach can be developed to avoid a significant challenge

Some scenarios are so probable or have such serious consequences that the best approach is to avoid the situation if at all possible. These are the probable disaster scenarios. By understanding these critical risk exposures early in a project, alternate methods can be explored that are less risky.

Unfortunately, Risk Management processes can be time-consuming and difficult. The calculation of probabilities and consequences is not trivial. Also, if the Project Manager is doing risk identification, prioritization and planning as parts of the project initiation the typical risks addressed are schedule risks – not quality risks.

Testing as a Risk Management Strategy

With the Project Manager's focus on managing risks to the schedule (the risk exposure of not implementing on time), there is often little focus on Quality Risk. Quality Risk is the risk exposure of an organization when deploying an application – the probable cost of extra work, re-work, poor reputation, etc. Very often, Test Execution is the key (or sole) element of Quality Risk Management.

The link between Test Execution and Quality Risk is clear. Defects in production are a large component of the quality risk exposure. Every test that identifies and allows us to correct a defect before deploying the application reduces the risk exposure of that defect to zero because its probability is now zero. It also reduces the total risk exposure of the application by the cost value of that defect.

The problems with Defect and Coverage Metrics

The information used to understand quality risk in an application is typically related to Test/Code Coverage (Test Coverage, Test Activity and Success Status) and Known Defects (Error Discovery, Open Defects). These Test Execution measures are trying to cover both of the components – probability and cost – of risk. Code and test coverage metrics are the probability component. A higher coverage percentage should lower the probability that there is an undiscovered defect. Defect Management metrics are cost based in that they help understand the cost component of going into production with identified defects.

These metrics are often tracked and reported as trends. By looking at these trends the Test Manager is expected to assist in making the decision of 'are we ready to ship?'. The testing textbooks tell us that as Coverage measures trend up (we are defining more tests, executing more tests and more tests are passing) and Defect measures trend down (we are discovering fewer errors and the discovered defects are being closed) there is higher quality in the software.

Used together these metrics try to give management some idea of the relative quality risk of the product. But these measures do not provide a complete picture of deployment risk because there is limited ability to predict the risk exposure of the defects that have not been found – either in probability or cost. One major undiscovered defect in an uncovered area could be more costly than all the defects uncovered by all of your testing.

Increasing predictability with Risk Assessment in Test Planning

As mentioned earlier, Risk Management starts with Risk Assessment – identifying risk scenarios and assigning probability and consequences to each scenario. The critical scenario related to Quality Risk is Defects. Quality Risk Assessment takes the form of trying to calculate the probability and the cost of defects in the application. While an overall risk of defects in an application may be valuable, what is truly indispensable in Test Planning is to understand the 'risk topography' of the application.

Not all parts of an application are equally important and not all parts of an application are equally buggy. In order to do effective Quality Risk Management we need a mechanism to quickly point us to the most important parts of the application – where defects are more probable and where defects would be most damaging.

This mechanism is a risk profile – a topographical map – of the application. This risk profile will allow us to view the application in terms of the relative risks. Each part of the application will have a risk area that is similar to the risk classes discussed above (ignore, prepare, special avoidance plan). With this information, the team's testing efforts can be focused on building and executing the set of tests with the highest probability of finding the most costly defects in this application – those focused on the part of the application with the greatest risk exposure.

Even using the same number of tests cases as we would have had in a random testing scheme, we can use the risk profile to decrease the risk exposure of the application. High Risk areas can be bombarded with very rigorous testing, medium level risk areas can be supported with moderate testing and the lowest level risk areas can be simply spot-checked with minimal testing.

Also, as a by-product of dividing the application into smaller risk assessed segments, this risk profile gives the Test Manager the ability to structure quality gates that monitor the quality of the most risky areas long before they are delivered into the System Test environment.

A technique for Assessing Relative Risks

The remainder of this paper describes a process and a tool to develop risk profiles of an application.

Even in a simple field, risk assessment can be difficult. Assignment of values to probability and consequences can be time-consuming and error-prone. The development of mitigation schemes can be a nightmare of mathematics and meetings. Understanding this problem, and knowing that this process must be repeated many times over the course of a project, the Test Risk Assessment process was designed to be as quick and flexible as possible.

There are three important mind-set assumptions in using this process. First, everything in this process is based on 'relative' values. There is no attempt to get 'absolute' values. Take a stab, put something down and do not spend time agonizing over decisions. Second, this is a collaborative process that is as much about communication and discovery as it is about building a profile. Third, the profile will evolve and emerge through the process and it is expected to change as we get more information – or the situation changes.

The risk assessment process is embedded in a spreadsheet or matrix mechanism. A paper version of this matrix is attached as the final page of this paper as a visual tool to assist you in understanding the process. This process can be done with this paper version but Compuware has had a lot of success with implementing this concept in Excel to handle the mechanics and recalculation as things change.

The process is built on the concepts of Risk Factors and Risk Weights.

Risk Factors are a set of criteria that are considered predictors of the probability that errors exist in the function or application. These factors span the entire software development lifecycle: from definition of requirements by the user group, through development practices and testing practices and even taking into account prior testing and production history. Each of the Risk Factors must have a rating scale. The rating scale indicates how the criterion should be scored. Some of the criteria attempt to rate the probability of failure and others attempt to assess the impact of a failure were it to occur.

Risk Weights are a mechanism that lets us balance these Risk Factors between the probability and the consequences of failure and across multiple organizations that contribute to the production of the software. These weighting factors are used in risk ranking formulas to help understand the overall probability that an error exists. This weighting technique allows us to take into account that some Risk Factors have a higher impact on the overall probability ranking than others will.

The Risk Factors are combined by a Rating Formula that adds the Factors using the Weights as a multiplier for the Factor.

Determine the Elements

The first step in building a risk profile mechanism is to agree what elements you are building a profile table for.

This process and tool can be implemented at any level in the software project – the elements being profiled could be applications – to help decide which of many competing projects should get the most test resources. It could be the sub-systems or requirements embedded in the application – to decide which parts of the application to test heaviest. The elements could even be individual programs/classes/panels – to help set the level of development testing required of the element.

The ‘elements’ that make up the rows should be determined in an inventory process. To help with this inventory process, there can be levels of decomposition. For example, if the elements are the integrations of the application, a major category might be Integration with Application X with sub-categories that describe the transactions that occur through that integration.

This decomposition can add organization and clarity but the Risk Profile is built for the lowest level in this decomposition.

Determine the appropriate Risk Factors

The next step is to agree what factors contribute to software risk for these elements in your organization. This will vary from organization to organization but there are two guidelines in determining the factors:

1. The factors that contribute to a risk profile should be a mixture of factors related to the probability that a defect would exist in the element and factors related to the consequences of a defect if it were to exist in the element

2. The factors should represent each group involved with the software. The factors should build on what each group knows contributes to risk – this should consider how the user community would define risk, how the development group understands it, what testing can add, etc.

Some typical factors to consider:

- The ‘age’ of the element (considers probability, is based in development)
- The complexity of the element (probability, development)
- The number of cross-element integrations (probability, users and development)
- The importance of the element to business goals (consequences, users)
- The importance of the element to key users (consequences, users)
- The prior history of defects in the elements – the more buggy the more testing needed until the bug history goes down (probability, test)
- The history of problems with the element in production (probability, test and users)
- The number of people that use this element (consequences, users)
- The experience of the development staff with the technology in this element (probability, development)
- The maturity of the process being used to develop the software (probability, development)
- The ‘politics’ of delivering to a specific group (consequences, management)

Each factor chosen should be defined in a way that will allow ranking this factor from 1 to 5 – with 5 indicating a high risk and 1 indicating a low risk. This understanding of clear criteria for the assignment of values should be documented. For example:

MARKETING RELATED

Message / Story.

- This column should contain a rating that indicates the amount of participation this product has in the Marketing Message or Story.
- Valid Range is 1-5 (5 designating major involvement in the overall message or story or involvement in a major message or story)

Competitive Environment.

- This column should contain a rating that indicates the degree or strength of competition for this product.
- Valid Range is 1-5 (5 designating a highly competitive product field)

Determine the relative participation (risk weight) of each Factor

Not all factors that contribute to risk are equally good at predicting defects. Some factors (experience with the technology) might be stronger predictors than others (prior testing history) in some projects and weaker in others. Consider, for example, how these two factors would be considered in a new development versus a maintenance project. Additionally, there are factor contribution differences that vary from organization to organization.

With this in mind, the next step is to assign a weighting factor to each risk factor. The weighting factor should be in the range of 0-2 with 0 meaning it is not expected to be a predictor and 2 meaning it is a very strong predictor.

At this point in the process, this weighting should be a first guess attempt. We will revisit these weighting factors later as we validate the calculated results.

Weight each Factor for each Element

At this point in the process we have a template for calculating the risk profile. Up until now, this has been a private document being built by the test team. At this point we begin filling it in through conversations and meetings with all of the interested groups.

This is the collaboration, investigation and negotiation portion of the process. A big, empty matrix/table of factors and elements can be intimidating. If this were sent around via e-mail with the subject ‘fill in your risks’ you would probably get nothing back (except some flaming responses).

This process is helped greatly by a one-on-one or two-on-two discussion. It is helpful to focus the discussion of people who are unfamiliar with this process to the risk areas they understand:

- Point users to items related to requirements, the user community and the market for the application
- Point development at the technology related aspects.

It also helps the discussion along greatly if the test organization has completed their rankings and can discuss how they determined the values as an illustration of the expectations.

This is probably the most time-consuming and difficult part of the process. It is truly an education process. Each time you go through this process it gets easier and people are more comfortable distinguishing between risk rankings. It also gets easier as people become comfortable that this is not an empty exercise but that the information supports a valuable planning tool.

Do the math

This is where the implementation of this process in a spreadsheet tool will pay off. Scores and indices in this table are based on the Risk Factors and Risk Weights.

The 'Scores' columns are the results of this weighted addition of individual factors. That is, each factor is multiplied by the weight of the column and the resulting products are summed to get a weighted product. This is the 'raw score' that is this element's risk profile.

These raw scores are then assigned a relative ranking of 1-5 as a quick 'Index'. This Index or category assignment is done by calculating the weighted average of the raw score. That is, the combined total is divided by the combined weight values assigned in the weighting area for the score. For example, if there were three rating values with relative weights of 1, 2 and 1.5, the total score would be divided by 4.5 (1+2+1.5).

This index is similar in concept to the ignore, prepare, special avoidance plan assessment mentioned at the start of this paper. The highest risk profile items are our 'disaster' scenarios – these are the ones that need the most attention and which could really benefit from some special planning. The lowest risk profile items are our ignore scenarios.

Validate the calculated results

At this point it is prudent to check the assumptions of your risk profile model and see that your results don't deviate significantly from your expectations. This is especially true if this the first or second time you are using this model.

The best way to check the model is to pick 2 or 3 elements that you are really familiar with and decide if you agree with the score and index values. It is also useful to have a couple of people from different groups do this sanity check exercise. Not only does this outside involvement double check the work but it increases the communication and ownership of the results.

Typical problems when the score and index do not match your expectations:

- One or more of the risk weights has skewed the results – some factors are being made too important or not important enough.
- There is not enough distribution of factor rankings. Not every item should be a ranking 1 or a ranking 5 – there is generally a normal distribution to these things and if you see too many of the same ranking down a column this indicates that there needs to be more work on considering if these are really the right rankings.

Use it in the overall planning process

We have now reached the point where all of the grunt work has been done and we can get real value out of this tool. The Index values provide the Risk Profile of the application that provides a topographical map for the testing effort:

- The items with the highest risk index value should be tested as early as possible.
- The items with the highest risk index value should have the most test cases developed and executed.
- Negotiate with development to get high risk index items developed and delivered to testing first.
- Develop more stringent entry and exit criteria for system testing of the highest index elements.

Additionally, this completed risk profile can help you size the overall testing effort. There is a relationship between the index value and the number of test cases. The higher the index value, the more test cases you should have for the element. The more test cases that you need to develop and execute, the more time you will need.

At the very least this risk profile will help communicate why some projects take longer to test than others:

- If time and resource constraints cannot be adjusted to cover all of the testing needed – at least it can be focused on the highest risk areas.
- By involving people all across the organization and educating them in the risks, there is more across-the-board support of the answer of ‘how much testing do we need?’
- Occasionally, this discussion can help development re-focus their efforts to reduce the risk profile of an area or drop requirements.

Use it as things change

As the landscape of the project changes – perhaps because development has slipped or new functionality has been added – this tool continues to provide value in the re-planning process. The majority of the work to build the tool was completed during Test Planning. As changes occur it typically requires only minor adjustments of rankings or weights to understand how a development change has affected the risk profile of the application.

The impact of a schedule slip in development while holding to the initial release schedule can be quantified. If notified that you have 2 less weeks to test, you can quickly understand and communicate where testing will suffer and what the possible additional risk exposure is.

If the schedule slip means that testing of low risk areas will be sacrificed this may be acceptable. However, when schedule slips start cutting into medium and high risk areas this can be clearly communicated to management. Now the risk of not implementing can be balanced against some real understanding of the risk of implementing.

As with the initial schedule and sizing discussions the process will be risk based and no longer solely owned by the testing organization. The User community and the developers were involved in the process that set the relative risks. With this assessment done and documented there is less of a tendency to 'hope for the best' or downplay potential risks.

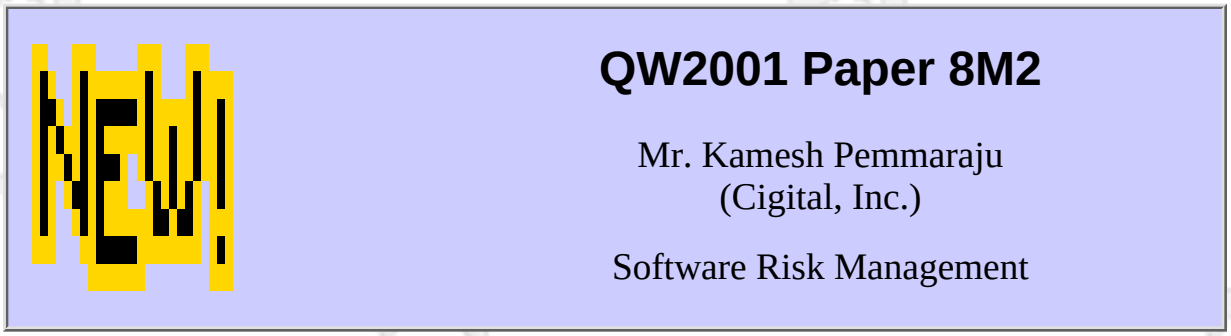
Keep the matrix as a history

The completed Risk Profile is a history of your planning. It provides justification of the decisions that guided your testing.

Additionally, it will be useful the next time you have to test this application – most of the work will be done and Risk Assessment for the new project can be one of adjustment and refinement.

	Risk Factors								Score Weighted Product of all Factors	Index Weighted Average of Score
	User Factors		Development Factors			Test Factors				
Risk Weight										

Elements	Risk Factor Ranking for Element								Risk Profile	



QW2001 Paper 8M2

Mr. Kamesh Pemmaraju
(Cigital, Inc.)

Software Risk Management

Key Points

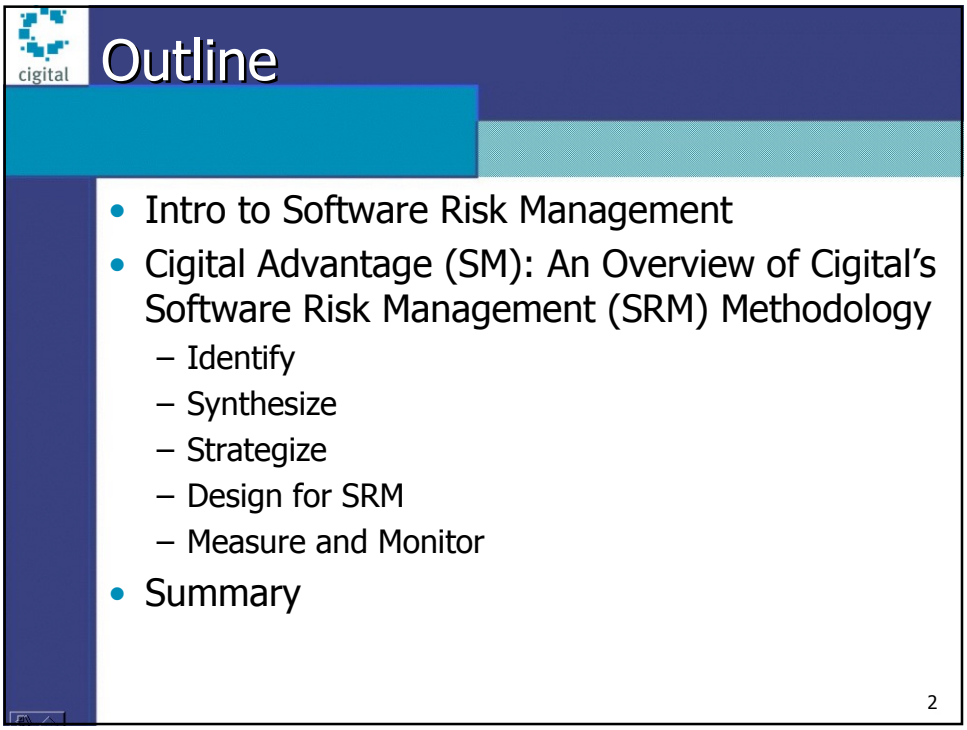
- Companies can prevent disatrous software risk by planning ahead.
- How does software risks impact business goals and what is the cost?
- How should software risks be prioritized and managed?

Presentation Abstract

It is hard to imagine a company today who doesn't use a piece of software in day-to-day operations. But what happens if that piece of software that is found in a car's braking system fails due to a faulty line of code? What happens if a line of code is faulty in an oil refinery and production is delayed a day? Software risks can harm a company's reputation and revenue. Today's business leaders need to realize the full effect software risks can have and how they can be prevented.

About the Author

Mr. Pemmaraju has over twelve years of hands-on experience in all aspects of software development: design, development, and testing of mission/business critical software. He currently works with Cigital (formerly known as Reliable Software Technologies), the leading authority and industry visionary on Software Risk Management (SRM).



Software Risk Management

- What is Risk?
  - The possibility of Loss or Injury

  [Source: Merriam Webster's Collegiate Dictionary, 10th ed.]
- Two types of risk
  - *Pure Risks*: Risks that can only result in a loss
    - Example: Airplane or a Car crash
  - *Speculative risks*: Risks that can result in *Profit* or *loss*
    - Example: buying stocks or gambling in a casino
    - Example: Software Projects!!!

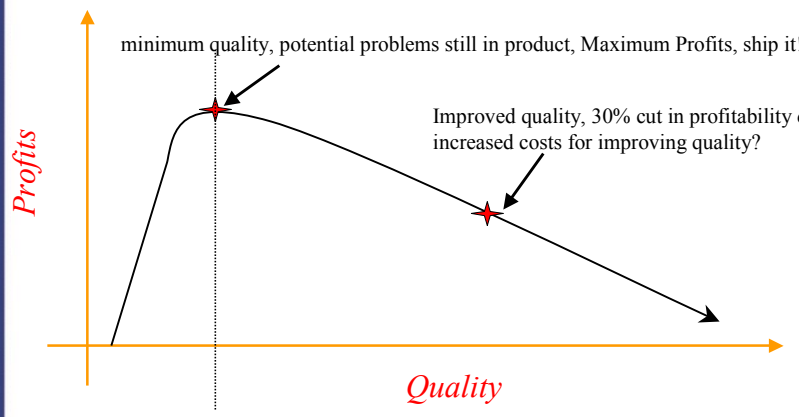
3

Software risk management

- Software risk management has traditionally focused on project risks:
  - Future happenings that can affect the project
  - Major sources of problems in the project
  - Technical or managerial factors that threaten the success of the project
- Of course this depends on the definition of "success"
  - Software publishers (they take speculative risks) typically define success as finishing the project at the lowest possible cost and the fastest possible time, while maximizing profitability.

4

What about Quality? Software publisher's view



Microsoft reportedly shipped Windows 2000 with 65,000 remaining bugs!

5

But software users suffer!

- Software failures are pure risks to the users. These failures sometimes affect the community, the economy, and the environment:
  - Business/economic losses
    - Loss of Revenue
    - Liability
    - Brand Damage
    - Windows 95 crashes
      - so what, this happens all the time? But consider this: it is estimated to cost the US industry $11.25B/year!!
  - Community/environment losses
    - Aircraft crashes
    - Medical device pumps 100 Amps through your brain
    - Nuclear power plant blows up

6

Publishers suffer indirectly!

- When users suffer, software publishers suffer, too!
  - Liability costs
  - Brand damage
  - Recall costs
  - Rework and maintenance costs
- These costs are enormous, sometimes greater than the project cost itself!

7

Software-induced business losses last year

- $500B was lost by businesses last year due to software-induced business risks:
  - **Loss of Revenue:** Hershey lost $150M last Halloween season due to a software glitch in their SAP system
  - **Brand Damage:** Online Music retailer CDUniverse's reputation was damaged due to a security flaw—a hacker stole 300,000 credit card numbers from their web-site
  - **Liability:** Pharmaceutical distributor sued SAP for $500M for allegedly bringing their business to a virtual standstill

8

Meanwhile publisher challenges continue..

- Highly compressed development schedules
- Fiercely competitive markets
- Tight budgets
- Lack of experienced professionals
- Employee turnover
- Constantly changing requirements

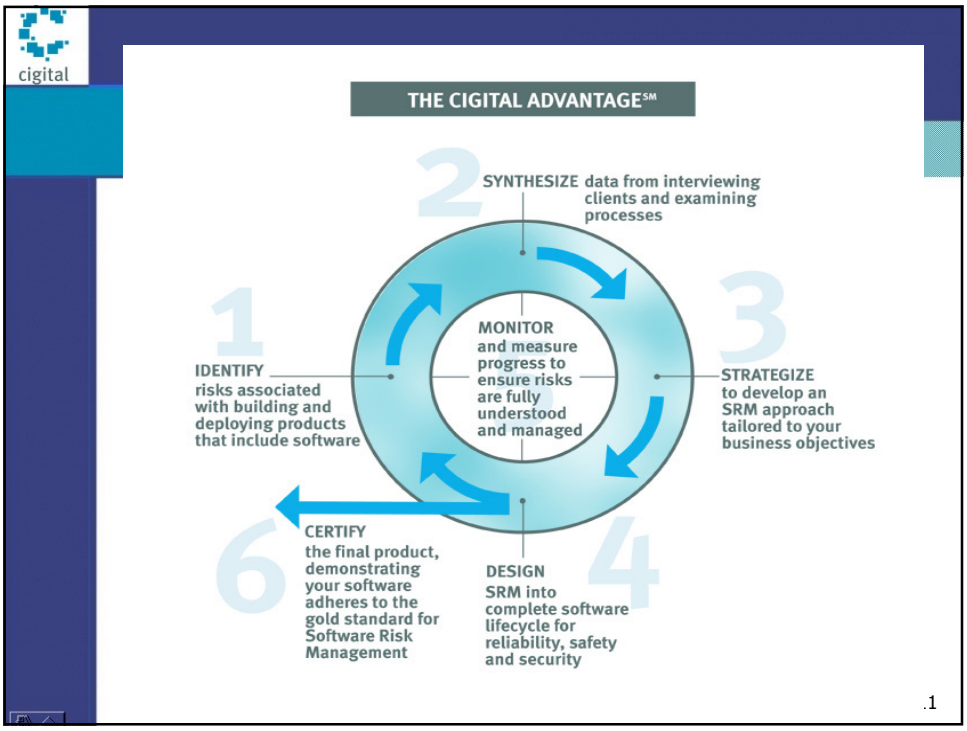
ALL OF THESE PROJECT REALITIES INCREASE RISKS OF POOR QUALITY SOFTWARE.

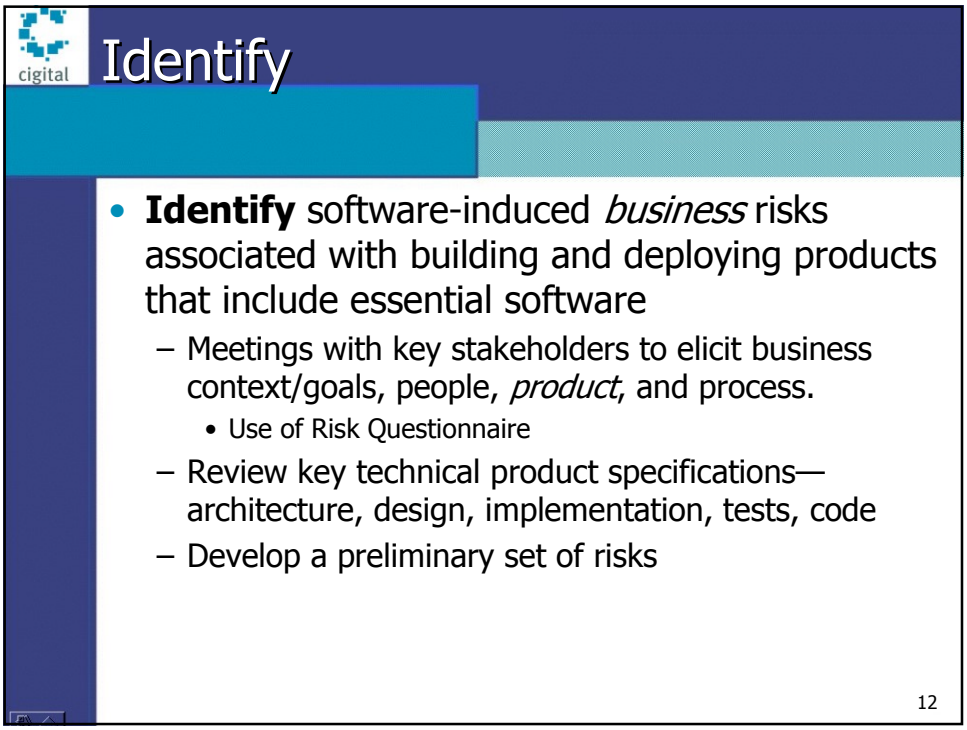
9

Business risk exposure due to software

- So what business risks am I exposed to due to poor software behavior? This leads to more questions:
  - What are the most important software risks and how do these software risks impact critical business drivers?
  - How big are these software risks?
  - What are their technical and business consequences?
  - How much will these software risks cost if they materialize?
  - How should these software risks be prioritized and managed?
  - What should be done to mitigate these software risks and how much will it cost?
- A structured, iterative, full life-cycle Software Risk Management Methodology focused on business goals is required to answer these questions.

10


cigital
THE CIGITAL ADVANTAGE℠
2
SYNTHESIZE data from interviewing clients and examining processes
1
IDENTIFY risks associated with building and deploying products that include software
MONITOR and measure progress to ensure risks are fully understood and managed
3
STRATEGIZE to develop an SRM approach tailored to your business objectives
6
CERTIFY the final product, demonstrating your software adheres to the gold standard for Software Risk Management
4
DESIGN SRM into complete software lifecycle for reliability, safety and security
.1


cigital
Identify
• Identify software-induced business risks associated with building and deploying products that include essential software
– Meetings with key stakeholders to elicit business context/goals, people, product, and process.
• Use of Risk Questionnaire
– Review key technical product specifications—architecture, design, implementation, tests, code
– Develop a preliminary set of risks
12

Synthesize

- **Synthesize** data gathered from stakeholder meetings and technical/project documents
  - Identify and prioritize key business goals and the consequences and costs of not meeting them.
  - Identify critical software risks and determine the likelihood of their impact on the the business goals
  - Identify preliminary mitigation methods

13

Business goals, consequences, and costs

Business goals	*Business Consequence*	*Potential cost*
Availability: Software failures cause the termination of the operation system on the server leading to the shut down and non-availability of the server.	Server Availability is a crucial concern, impacting the service level agreements the server companies have with their customers who operate in 24/7/365 mission-critical business environments. If there are service-level agreements, the server company will share some of its customer losses.	The potential costs to the server company customers can vary from $100K to $1 million per hour of downtime depending on the application. Scaled to 1000 systems in the market place, this cost can potentially be $1 billion.
Time-To-Market: The server system does not meet acceptance criteria when the servers are ready to be shipped.	The server company will lose the crucial first-to-market advantage.	Depending on the revenue goals, the server company can lose millions of dollars/day if it misses the time-to-market window and delays server shipments.
Reliability: The server software fails to perform critical operational functions correctly.	Reliability is a key requirement, which leads to better Total Cost of Ownership (TCO) and reduced costs for maintainability and serviceability.	Support and maintenance costs reduce the overall value of the server software.

Risk severity classification

CONSEQUENCE OF NOT MEETING BUSINESS GOALS	*COST OF THE RISK MATERIALIZING*	*MITIGATION RECOMMENDATIONS*
Catastrophic	The cost to the business of this risk materializing is enormous.	Risk reduction and implementation of mitigation strategies is **mandatory.**
Critical	This risk is tolerable only if the cost of implementing the mitigation strategies is disproportionate to the cost of the risk itself.	Risk reduction and implementation of mitigation strategies is **highly recommended.**
Important	These risks are tolerable only if the mitigation would exceed the improvement gained.	Risk reduction and implementation of mitigation strategies is **recommended.**
Non-Critical	These risks are cosmetic or inconsequential to overall operation of the system.	Risk reduction and implementation of mitigation strategies is **not recommended**.

Mapping of Software risks and business goals

Software Risks → / Business Goals ↓	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Performance **(Catastrophic)**	H	H	H	L	M	L				L
Reliability **(Catastrophic)**	H	H	H	H	H	H	H	M		L
Availability **(Critical)**	H	H	H	H	H	H	H		M	

Likelihood: H: High, M: Medium, L: Low

16

Strategize

- **Strategize** a complete project plan for managing software-induced business risks consisting of:
  - Identify the mitigation methods for software risks
  - Create a comprehensive implementation action plan to mitigate risks
  - Identify the expertise/roles/responsibilities to carry out the mitigation plans
  - Create a schedule (dates/milestones) for implementing the mitigation plan

17

Mitigation strategy

Software Risks → / Mitigation Methods ↓	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Method #1	M	M			M					
Method #2	M	M		M	M	M	M		R	H
Method #3	M	M			M	M	M	H	R	H
Method #4	M	M			M	R			H	H
Method #5	R				R	R		R		
Method #6		R	R	M		H				

M: Mandatory, H: Highly recommended, R: Recommended

18

Design for SRM and Monitoring and Measurement

- **Design for SRM** is an on-going activity which focuses on *architecting-in* mechanisms to prevent/reduce the identified risks
- **Monitor and measure** risks is an ongoing activity to measure progress of mitigation against identified risks and to identify new risks.

19

Summary

- Companies *can* prevent disastrous software risk by planning ahead and focusing on the business impact of software development and deployment.
- Companies should consider creating an independent software risk management function to
  - Determine software risks impacting business goals and the business consequences and costs of those software risks.
  - Prioritize and manage software risks throughout the life-cycle of a project.
  - Provide ROI justification for the mitigation methods employed for preventing/removing software risks.

20

SOFTWARE RISK MANAGEMENT

Kamesh Pemmaraju, kamesh@cigital.com
Cigital, Inc (http://www.cigital.com)

As software technologies continue to evolve in functionality and complexity, we are experiencing the rapid expansion of software into all areas of our business and private lives. Today, software is found in cars, traffic lights, household appliances, communications equipment, transportation systems, hospitals, airplanes, medical devices, next-generation payment cards, business supply chains, and enterprise management systems, to name but a few places. Software has truly become ubiquitous and *essential*. It is hard to imagine a company today that does not use a piece of software in its day-to-day operations. But what happens if a piece of software in a car's braking system fails due to a faulty line of code? What happens if a faulty line of code in an oil refinery delays production? Software risks can harm a company's reputation and revenue. Today's business leaders need to realize the full effects of software risks and how these risks can be prevented.

The consequences of *essential* software failure can be dramatic. At the extreme, the failure of *essential* software in a safety-critical system can result in loss of life. From a business perspective, the financial consequences of *essential* software failure can also be severe:

- The Standish Group estimates that software problems cost U.S. businesses $85 billion in lost productivity in 1998.
- Hershey lost $150M in revenue during Q3 1999 when an enterprise software glitch prevented Halloween candy from being shipped.
- eBay's 22-hour system outage in June 1999 resulted in a revenue loss of $4M and a loss of consumer confidence that lead to a market capitalization drop of $5.7B for the online auction giant.
- The SEC has fielded over 20,000 investor complaints related to software problems in online trading.
- The parent company of bankrupt pharmaceutical distributor FoxMeyer is suing SAP for $500M over enterprise software that allegedly snarled operations.

Brand awareness and confidence are all too easily eroded, and often software problems are to blame:

- H&R Block suffered significant brand damage and credibility loss when a software glitch allowed online clients to view other clients' tax returns.
- CDUniverse's reputation was compromised when its software was exploited by a hacker who stole 300,000 credit card numbers and published the information online, complete with names and addresses.

As companies try to come to grips with such severe consequences, they are faced with equally daunting challenges to "do it right" from a software development perspective, even though not doing it right may risk the entire business. Companies face typical challenges such as highly compressed development lifecycles, fiercely competitive markets, tight budgets, lack of experienced software professionals, employee turnover, and constantly changing requirements, among others.

Most business people understand how to manage these types of challenges: business executives do it every day when they make calculated decisions. Software Risk Management (SRM) provides information that allows business executives to improve their decision making through understanding the risks that software brings to their businesses. By grasping the business proposition – the technology that is being built and the role that it plays in the business model – managers can mitigate the risks associated with building and deploying software-based systems. These issues can be framed in terms of potential payoff and required investment, and sound decisions can thus be made using a marriage of business goals and technology realities.

THE CIGITAL ADVANTAGE[SM] : A SOFTWARE RISK MANAGEMENT METHODOLOGY

While companies may understand the critical business drivers of their software products and the development challenges of the real world, they do not necessarily understand how *software risks* can impact those business drivers.

Fortunately, given the right data about software behavior and a *methodology* to identify and understand risks associated with that behavior, companies can control software-induced business risks and their corresponding business consequences.

The key questions that need to be addressed in order to understand the risks that software brings to a company's business operations include:

- What are the most important software risks and how do these software risks impact critical business drivers?
- How big are these software risks?
- What are their technical and business consequences?
- How much will these software risks cost if they materialize?
- How should these software risks be prioritized and managed?
- What should be done to mitigate these software risks and how much will it cost?

Both business risks and technology risks must be identified, ranked in order of severity and potential impact, and addressed in rank order by well-conceived mitigation techniques. Any sort of severity ranking is clearly a context-sensitive and time-dependent perspective that depends directly on the changing business needs and goals of the system at hand. Starting the process early is important: the earlier in the development process that risks are taken into account, the more efficiently mitigation planning and resource allocation can proceed. Thus, what is required is a clearly defined, well-structured, iterative and full lifecycle Software Risk Management process/methodology that provides clear ROI justification and minimizes the financial impact of negative business consequences. The *Cigital Advantage* is one such Software Risk Management methodology that takes all these factors into account.

Software risk management (SRM) techniques can be applied throughout the software development lifecycle to manage the software-based risks in systems. In so doing, these techniques protect the overall business goals for the product. Using an appropriate SRM approach reduces the likelihood of software failure, thereby increasing lifecycle productivity while still meeting time-to-market demands.

The Cigital Advantage SRM approach includes a series of six inter-related steps that can be used to help understand and mitigate key software risks:

1. **Identify software-induced business risks** associated with building and deploying software products
2. **Synthesize** data gathered from structured questionnaire sessions with key stakeholders of the product and business.
3. **Create a SRM Strategy** drawing on expertise at various technology and business levels to determine critical tradeoffs that exist between technology-driven approaches and a company's business objectives for the software product.
4. **Design for Software Risk Management,** resulting in a software system that is *designed* right from the start to be reliable in real-world conditions, safe in operation, and free from security vulnerabilities.
5. **Measure and Monitor** progress against software-induced business risks, providing insight into testing and validation of the delivery and deployment of the software and ensuring that business risks are understood and managed.
6. **Certify** that the software meets an acceptable SRM standard.

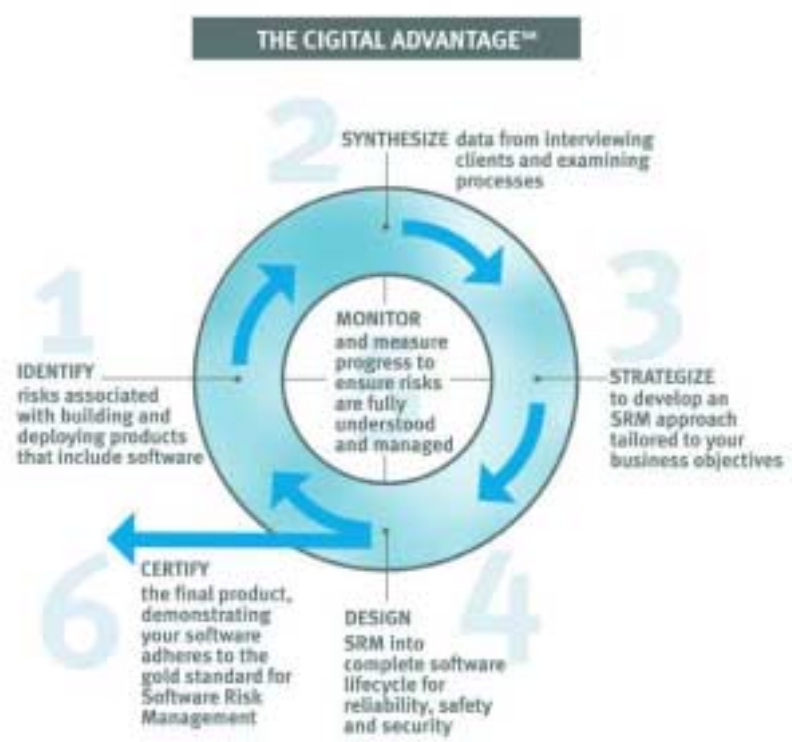


Figure 1: The Cigital Advantage Methodology

Keep in mind that these steps are not carried out as a one-time activity. Rather, the entire process is iterative and risks are regularly reviewed and the SRM strategy updated according to changing business and technology drivers.

APPLYING THE CIGITAL ADVANTAGE

IDENTIFY RISKS

Identifying software-induced business risks is an essential first step to the SRM solution. A Risks Questionnaire (RQ) plays a central role in eliciting discussion of risks during a series of stakeholder meetings. Using the RQ as a guide, discussions are conducted with the stakeholders about their market, business, product, process, and project. The RQ provides a framework for the meetings, making the risk identification process more systematic and repeatable. Though risks will be identified and worked into the overall risk management strategy throughout the duration of the project, the initial set of risks is created through application of the RQ.

The series of risk questionnaire meetings includes following stakeholders:

- **Upper management (LOB or c-level representatives):** This group will provide the best data about the business proposition of the product initiatives. A central tenet of the risk management solution is using information about the business proposition to guide the SRM process. This group is able to change budgets and schedules according to business needs. In business parlance, this group has profit/loss (P/L) responsibility and acts to maximize shareholder value.
- **Project Management:** This group is constrained by budget and schedule. For them, progress is usually measured in terms of time to market and cost of delivery. Though project managers may understand the business proposition, they are not often in a direct position to implement critical tradeoffs.
- **Architects**: This group of technical experts helps design the product according to technical requirements. Business priorities can be misunderstood or misinterpreted by architects. Likewise, risk management specialties such as security and testability are areas where architects require help. Though they wield much power in the technical realm, they may not interact with the business side of the house.
- **System Engineers/Analysts**: This group develops system requirements and serves as the interface between the end users and the project team.
- **Developers**: In-the-trenches-technologists, this group is charged with creating the product designed by the architects.
- **Testers:** Also in the trenches, this group is charged with analyzing and testing the product throughout its lifecycle.

An example set of questions to ask a marketing executive would be:

- Is the target market well-defined?
- Is there a fixed time-window for product delivery?
- What market category will the product address (e.g., custom, shrink wrap, vertical)?
- Has customer dissatisfaction been an issue with this or other related products?
- Are customers involved during the development process?

Following this step, a detailed review of all project-related business/technical documentation is carried out to create a comprehensive list of software-induced business risks.

SYNTHESIS

Synthesis will help in understanding and prioritizing both the critical business goals for the software product and the software risks impacting those business goals. Preliminary cost-justified and ROI-justified mitigation strategies are also developed during this step.

Some examples of typical business goals are availability, time-to-market, reliability, flexibility, and cost. Since the software risks in the product can potentially impact one or many business goals, it is important to develop an understanding of the business costs of these software risks. The key question from a business perspective is: how much will software-induced business risks cost if they materialize?

It helps to create a table that summarizes, in priority order, the top business goals, the consequences of not meeting these business goals, and the potential costs to the company if these goals are not met. The following table shows an example of such a table for high-availability server software:

Business Goals	*Business Consequence*	*Potential Cost*
Availability: Software failures cause the termination of the operation system on the server leading to the shut down and non-availability of the server.	Server availability is a crucial concern, impacting the service level agreements the server companies have with their customers, who operate in 24/7/365 mission-critical business environments. If there are service-level agreements, the server company will share some of its customers' losses.	The potential costs to the server company customers can vary from $100K to $1 million per hour of downtime, depending upon the application. Scaled to 1,000 systems in the market place, this cost can potentially be $1 billion.
Time-To-Market: The server system does not meet acceptance criteria when the servers are ready to be shipped.	The server company will lose the crucial first-to-market advantage.	The server company stands to lose millions of dollars per day if it misses the time-to-market window and delays server shipments, impacting the revenue goal for this server family.
Reliability: The server software fails to perform critical operational functions correctly.	Reliability is a key requirement that leads to better Total Cost of Ownership (TCO) and reduced costs for maintainability and serviceability.	Support and maintenance costs reduce the overall value of the server family.

Table 1: Business Goals, Consequences and Costs

As preliminary mitigation plans are developed, several tradeoffs must be considered. For example, some mitigation activities may add cost to the budget or lengthen the schedule. The cost of the mitigation activity must be weighed against the importance and cost of the risk it

mitigates. Some risks are so catastrophic that the mitigation methods for addressing them are mandatory, while others are not. The following table presents the categories that can be used to classify identified risks. Such classification helps justify the cost of implementing the mitigation plan and helps prioritize the recommended mitigation activities.

Consequences of Not Meeting Business Goals	*Cost Involved if the Risk Materializes*	*Mitigation Recommendations*
Catastrophic	The cost to the business of this risk materializing is enormous.	Risk reduction and implementation of mitigation strategies is **mandatory.**
Critical	This risk is tolerable only if the cost of implementing the mitigation strategies is disproportionate to the cost of the risk itself.	Risk reduction and implementation of mitigation strategies is **highly recommended.**
Important	These risks are tolerable only if the mitigation would exceed the improvement gained.	Risk reduction and implementation of mitigation strategies are **recommended.**
Non-Critical	These risks are cosmetic or inconsequential to the overall operation of the system.	Risk reduction and implementation of mitigation strategies are **not recommended**.

Table 2: Risk Severity Calculation

The next step is to create a mapping between the software risks and the business goals. An example of such a mapping between software risks and business goals is shown in the following table:

Software Risks / **Business Goals**	**Software Risk #1**	**Software Risk #2**	**Software Risk #3**	**Software Risk #4**	**Software Risk #5**	**Software Risk #6**	**Software Risk #7**	**Software Risk #8**	**Software Risk #9**	**Software Risk #10**
Performance **(Catastrophic)**	H	H	H	L	M	L				L
Reliability **(Catastrophic)**	H	H	H	H	H	H	H	M		L
Availability **(Critical)**	H	H	H	H	H	H	H		M	
Maintenance **(Critical)**	L		L	M	L	M		H		
Usability **(Important)**	L	H	H	H		M	H	M	H	
Portability **(Important)**		H	L		L				M	H

Table 3: Mapping of Software Risks to Business Goals

This table shows the likelihood of each software risk contributing to not meeting the business goal. The likelihood levels are: H = High, M = Medium, and L = Low. The severity of not meeting the business goal is indicated in parenthesis.

SOFTWARE RISK MANAGEMENT STRATEGY

This step involves the creation of a comprehensive Software Risk Management (SRM) strategy consisting of a set of recommended mitigation methods. The mitigation methods are selected based on their effectiveness in mitigating the identified risks. Table 4 is an example of the risk areas covered by the different methods as they are applied in the recommended SRM Strategy. The matrix further shows method recommendations in terms of which risks they mitigate and the relative ranking of how important and effective they are for the particular risk. The relative ranking of the methods is presented in terms of *Mandatory* (M), *Highly Recommended* (H), and *Recommended* (R) attributes. Empty squares in the matrix indicate where application of a given method for a given risk area is *Not Recommended*. For all the method recommendations, the rankings were used to develop the overall SRM plan by providing guidance on the depth to which each of the methods should be applied. It is worth noting that each individual method will lose effectiveness and risk coverage if applied without following the integrated SRM strategy and plan, as recommended.

Software Risks / **Mitigation Methods**	**Software Risk #1**	**Software Risk #2**	**Software Risk #3**	**Software Risk #4**	**Software Risk #5**	**Software Risk #6**	**Software Risk #7**	**Software Risk #8**	**Software Risk #9**	**Software Risk #10**
Method #1	M	M			M					
Method #2	M	M		M	M	M	M		R	H
Method #3	M	M			M	M	M	H	R	H
Method #4	M	M			M	R			H	H
Method #5	R				R	R		R		
Method #6		R	R	M		H				

Table 4 Mitigation strategy

DESIGN FOR SOFTWARE RISK MANAGEMENT

Design for SRM means designing the software from the ground-up in order to ensure that risk mitigation methods developed during the first three steps of the methodology can be applied easily and quickly. Some mitigation methods may include testing for quality and security. Software must therefore be designed to *be* testable and secure. The SRM Solution pays close attention to design and architecture.

Software cannot be rushed to market and later made reliable or secure. Rather, it must be carefully *designed* to be reliable *and* secure. And there is no such thing as perfection. Reliability and security always involve tradeoffs and must be weighed against business requirements and

objectives. The architecture phase of software development is where much of the risk management activity takes place.

Risk analysis of the resulting software architecture must also take place on a technical level. Once a design has been created and formalized with an eye toward business risks, that design must be carefully assessed for emergent properties such as reliability, safety and security. Analysis of technical software risk is much more efficient when conducted early, rather than later, in the software lifecycle. Software Risk Management is not a task to be done once and forgotten; it is, rather, a process.

MEASURING AND MONITORING PROGRESS AGAINST RISKS

Even the world's best design can be poorly implemented. The SRM Solution places a critical emphasis on probing and measuring the actual software product throughout its development. Unfortunately, merely following processes like those utilized in support of the Capability Maturing Model or ISO 9000 cannot, alone, deliver software that works. Good process can be helpful, but in the end, it's the *product* – not the process – that must run on a machine. Measuring the product throughout development yields important data on software behavior, even before the software is put into use.

SUMMARY

Essential software systems are becoming more and more common and are beginning to affect deeply both our core businesses and our daily lives. Software failure in essential software systems is unacceptable: serious implications that result from such failure include loss of life and extreme business exposure. Thus, the risks that essential software systems bring to bear must be carefully managed.

By using SRM methodologies such as *The Cigital Advantage*, companies *can* prevent disastrous software risk by planning ahead and focusing on the business impact of software development and deployment. Companies should consider creating an independent software risk management function and deploying SRM methodologies to:

- Determine software risks impacting business goals and understand the consequences and costs of those software risks.
- Prioritize and manage software risks throughout the life cycle of a project.
- Provide ROI justification for the mitigation methods employed for preventing/removing software risks.

Good Software Risk Management practices engendered by the use of structured SRM methodologies like *The Cigital Advantage* and implemented by expert software engineers can help minimize software-induced business risk.

Mr. Pemmaraju has over fifteen years of hands-on experience in all aspects of software development: design, development, and testing of mission/business critical software. He currently works with Cigital (formerly known as Reliable Software Technologies), the leading authority and industry visionary on Software Risk Management (SRM). You can contact him at pemmaraju@cigital.com.

QW2001 Paper 9M1

Mr. Michael J. Hillelsohn
(Software Performance Systems)

Organizational Performance Engineering: Quality Assurance For The 21st Century

Key Points

- Multi-disciplinary, pro-active Organizational Performance Engineering is the future of quality
- Effective Quality Assurance facilitates the next step in a process instead of waiting to correct
- Fresh approaches to quality assurance are needed to apply sound software engineering principles

Presentation Abstract

In too many organizations, quality assurance is relegated to verification and validation activities and is separate from training and software process improvement organizational elements. Proactive quality assurance is actually a multi-disciplinary set of activities that combine these three organizational elements into organizational performance engineering. The approach being advocated uses techniques from quality assurance and control, training, program management, business process reengineering, and software process improvement disciplines to analyze the organization's practices and behaviors and initiate in-process interventions. The philosophy of organizational performance engineering is based on the notion that people really do like to do things correctly and really hate to go back and rework what they have already completed. It is a pro-active approach that emphasizes preventing problems and defects from occurring.

About the Author

Michael J. Hillelsohn is a Director, Product Assurance at Software Performance Systems (SPS) in Arlington, VA. SPS builds secure e-commerce, case management, and network solutions for government and industry clients. Mr. Hillelsohn is a certified quality professional with more than thirty years of experience doing development, management and performance improvement in software and systems development environments. His multi-disciplinary approach combines quality systems and training expertise to improve the performance of organizations and individuals. Mr. Hillelsohn's process-oriented, performance engineering methods facilitate adoption of external frameworks (CMM, ISO, Baldrige) to improve the quality of organizational products and services.

Quality Week 2001

Organizational Performance Engineering:

Quality Assurance for the 21st Century

Michael J. Hillelsohn

SOFTWARE *PERFORMANCE* SYSTEMS

2011 Crystal Drive Suite 710
Arlington, VA 22202
(703) 797-7707 mhillelsohn@goSPS.com

Objectives

- Expand your definition of quality assurance to include all sorts of interventions that improve the quality of the products and services that your organization delivers.
- Design an engineering approach to determining how you are going to be most effective in your organization.
- Think about an implementation strategy for pro-actively impacting the performance of people and processes in your organization.

Quality Assurance Definitions

- (1) *A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or (software work) product conforms to established technical requirements.* (2) *A set of activities designed to evaluate the process by which (software work) products are developed or manufactured.*

IEEE Std 610.12-1990

( ) SW CMM

Quality Assurance Definitions

- *All the planned and systematic activities implemented within the quality system (organizational procedures, processes, and resources needed to implement quality management) and demonstrated as needed to provide adequate confidence that an entity will fulfill requirements for quality.*

ANSI/ISO/ASQC A8402-1994

Quality Assurance Definitions

- *The set of support activities (including facilitation, training, measurement, and analysis) needed to provide adequate confidence that processes are established and continuously improved in order to produce products that meet specifications and are fit for use*

Quality Assurance Institute's CQA Study Guide Version 2

Where We Would Like To Be

- PERFORMANCE - *The activities performed by individuals, groups, and organizations to produce the things they deliver to customers.*

Michael

- ENGINEERING - *The application of a systematic, disciplined, quantifiable approach to structures, machines, products, systems or processes.*

IEEE Std 610.12-1990

Underlying Principles

People prefer to do a good job rather than a poor job

&

People hate to go back and correct things that they have already completed

Planning Activities

- Define quality policy and goals
- Participate in preparing management and development plans
- Support life cycle tailoring
- Write Quality Assurance Plan/Quality Plan
- Personnel and task management
- Tracking and oversight of resources, schedule, activities, products

Policies, Standards, Guidelines

- Write and maintain quality policies
- Research current standards & methods
- Adapt standards/methods to project /organization
- Write implementation guidelines
- Facilitate definition of work instructions
- Train/consult users on implementation
- Tailor standards… as required

Compliance Verification

- Documents
  - Review standard with developer
  - Conduct in-process reviews
  - Participate in inspections
  - Final compliance check
  - Sign off on deliverable/product
  - Compile metrics

Compliance Verification

- Design Diagrams/Code
  - Publish standards that affect design/code
  - Conduct training on walkthroughs/ inspections/ peer reviews
  - Coordinate & attend reviews
  - Compile results of reviews
  - Follow up on action items
  - Verify & validate test activities
  - Review test results

Process Improvement

- Analyze "as-is" environment
- Conduct "how to define a process" training
- Facilitate analysis of process improvements
- Define processes
- Plan implementation of improvements
- Audit the process(es)

Consult & Train

- Determine & anticipate life-cycle training requirements
- Design event driven learning (EDL) sessions
- Provide “just-in-time” EDL
- Maintain training records
- Exercise a consulting role on the project team

Process Teams

- Identify issues to be addressed (data-driven)
- Help form the team
- Train/facilitate/coach the team
- Support analysis of issue
- Facilitate definition of solutions
- Provide methodology for implementation pilot
- Generate standards/guidelines for implementation planning

Conduct Assessments

- Train organization on expectations
- Review compliance/non-compliance criteria
- Establish the baseline culture
- Determine compliance with internal & external standards
  - **Perform gap analysis**
  - **Suggest method to reach compliance**
  - **Facilitate development & implementation of action plan**
- Perform in process audits

Reporting

- Track progress against goals
- Define relevant metrics
- Gather metrics from other disciplines
- Analyze data
- Report results
- Recommend action(s)
- Conduct special studies

A Functional Organization

Senior Executive

Performance Engineering

Product Assurance	System Services
•Quality Assurance/Systems	•Training
•Process Improvement	•Internal (non-HR)
•Requirements Analysis	•External
•Verification & Validation	•Technical Publications
•Configuration Management	•Customer Service(s)

An Example - PE

Plan Requirements Capture	•Brief/Tailor RM Process & Standards
Capture&Identify Requirements	•How to Facilitate Reqs Gathering
Document Requirements	•How to Write Reqs
Conduct Walkthroughs	•Coordinate/Facilitate Walkthrough
Customer Requirements Review	•Gather Data
Write Requirements Spec	•Review SRS Standard
	• Quality Assurance Review

Successful Performance Engineering Is...

- Inter-disciplinary
- Insidious
- Flexible
- Cooperative
- Versatile
- Pervasive
- Systematic
- Based on facts
- Team oriented
- Communication
- Practical
- Focused on internal & external customers
- Supported by senior management

Summary

- Proactive quality assurance entails a multi-disciplinary set of activities
- Performance Engineering is assessing and enhancing the performance of individuals', groups' and the organization's products and services
- Success means making the developers' job more effective and efficient
- Result = achieving quality goals!

QW2001 Paper 9M2

Mr. Brian Lawrence
(Coyote Valley Software)

Choosing Potential Improvements -- Comparing Appoaches

Key Points

- What are the advantages and disadvantages of different improvement approaches
- How to analyze the your situation to help choose what you're going to do
- Examples of how others have made improvement strategy choices

Presentation Abstract

You know that what you're doing right now won't keep working at some point in the future, so you want to try something different. But what should you do? Should you follow reference models such as the CMM or ISO-9000? Or just choose something that sounds good and hope for the best? Maybe your CEO's pick-of-the-day?

There are many possible things you can do to improve your software design process: Inspection, Configuration Management, Testing, Modeling Requirements and Designs, QA, Retrospectives, Project and Risk Management, as well as others. How do you choose which to do? People will tell you the advantages of their favorite approach, and encourage you to head off in that direction. Is that what you should do? All of the choices have potential benefits, and what their purveyors frequently forget to mention is that they all have risks too. Some might not work-they definitely cost time and money-and you might not need some of them.

For example, you can assess your organization against a reference model such as the CMM. But is that the right model for your business needs? When the CMM was originally designed, its designers relied on some big assumptions about the nature of the software businesses that would use it. Those assumptions may not be true for you. And organizational assessment can be expensive.

In this presentation, I will compare improvement approaches using these criteria: "routineness," complexity, constituency, difficulty, and level of effort. I will explain what I mean by each of the criteria, and then offer my evaluation of each of the approaches I've examined. You will have the chance to see how different approaches match up. By examining the relative value of possible approaches and the risks and benefits, you can have a better basis for choosing among them. I will offer you my take on how these different ideas might work out, and where I've seen them both succeed and fail.

Choosing the next improvement effort can be a dicey decision. I don't believe there is any one right way, or one right answer. The proper choice depends on your circumstances and potential capabilities at the moment when you want to attempt the improvement effort. You can better inform your decision by using an evaluation such as this. I recommend that you take my comparison as a starting point, and conduct your own comparison based on what you know of your own organization. That way you can improve the chances that what the improvement effort you set out to do does indeed succeed.

About the Author

Brian has presented at many conferences on a variety of subjects over the years. He has served as a program chair for the SEPG'97 Conference and the 1998 International Conference on Requirements Engineering. Brian teaches and facilitates requirements analysis, peer reviews, project planning, risk management, life cycles, and design specification techniques. Brian serves on the editorial board of IEEE Software and as the editor of Software Testing and Quality Engineering magazine.

Comparing Improvement Approaches

Brian Lawrence
Coyote Valley Software
brian@coyotevalley.com
www.coyotevalley.com
(408) 578-9661

Winter 2001

The Pretext

- You are already doing some things to produce your software:
  - Such as coding!
  - Possibly other things
- You would like to do better:
  - perhaps fewer defects
  - perhaps more predictable, quicker delivery

Some choices for improvement may work better than others, depending on your circumstances.

Two Quotes

"Things are the way they are because they got that way."

- Kenneth Boulding

"We have met the enemy, and he is us!"

- Walt Kelly (from *Pogo*)

Why compare improvement approaches?

- Many (most?) improvement efforts fail because:
  - We choose the wrong thing to target
  - We don't lay the groundwork properly
  - We don't commit the proper resources to do the job well
  - We don't get the right people to participate
  - and...

Improvement is really hard!

Criteria

- "Routineness"
  - Not "we routinely do this."
  - From organizational theory, a routine task has little variation between work put in and results coming out. (known and predictable)
- Constituency - Who participates?
- Complexity - How intricate?
- Difficulty - How hard?
- Size of Effort - How big?

A Caveat!

The following table contains values I put in based on my experience, reviewed and adjusted by some of my colleagues.
Your experience is different, so you might put in different values. Feel free to do so.

Activity Comparison

Activity	Routine?	Constituency	Complexity	Difficulty	Effort Size
Project Charters	Yes	Senior Staff	Simple	Medium	Small
Life Cycles	Yes	Senior Staff	Can be simple	Medium	Small
Inspection	Yes and no	Team members	Simple	Hard	20-30% of team effort
Requirements modeling and management	Never	Wide Cross-functional	Complex	Very Hard	10-70% of total effort
Architecture and design modeling	Never	Designers	Complex	Hard	10-20% of designer's
Project management	Maybe	Manager & Team	Very complex	Hard	All of manager's
Risk management	No	Senior staff & managers	Simple	Medium	Medium
Configuration management	Yes	Dev & Testers	Medium	Not hard	Small
Testing	No	Testers & Dev	Complex	Medium	All of testing
QA	Yes	QA Staff	Simple	Can be Hard	Small
Organizational appraisal (ISO and CMM)	Yes	Very broad	Complex	Hard to get right	Large
Retrospectives	Yes	Team	Simple	Can be tricky	Small

Retrospectives

AKA Post-Mortems

- Advantages
  - Best place to start!
  - Counters the effects of rumor due to *leveling, sharpening, & assimilation*
  - Not very expensive
- Risks
  - Damaging if done badly
  - Can be damaging if recommendations are ignored
  - Someone may have already poisoned the well
  - Tempting not to use a trained facilitator

Project Chartering

AKA ?

- Advantages
  - Omission is a major source of project failure
  - Routine with a knowledgeable facilitator.
  - Not expensive
- Risks
  - Establishes accountability
  - Not seen as needed
  - Can be misinterpreted as casting project in concrete
  - Not many people know how

Testing

AKA QA - Not!

- Advantages
  - Mainstream approach
  - Lots of good knowledge around
  - Very understandable
- Risks
  - Can be very expensive
  - Hard to find qualified staff
  - Can encourage developers to abandon their responsibility
  - Prone to overlooking entire classes of defects, especially requirements and design defects
  - Frequently underestimate level of effort, especially for automation
  - Vulnerable to late changes

Inspection

- Advantages
  - Single most effective quality technique
  - Despite considerable investment, has immense ROI
  - Fosters professionalism in the ranks
- Risks
  - Scary - impossible to hide anything
  - Hard to get right
  - Produces information which can easily be misinterpreted
  - Must have ironclad management support

Requirements Modeling and Managing

- Advantages
  - Greatest source of potential value
  - Can optimize the level of conflict
  - Vastly lowers risk and increases predictability
  - Foundation of most other quality strategies
- Risks
  - Strongly establishes accountability
  - Easy to underestimate level of effort
  - Frequently misunderstood
  - Non-routine, broad constituency, very complex, and difficult
  - Some don’t think they need or can get better requirements

Project Management

- Advantages
  - Vastly improves chances project will succeed
  - Much is known about it
- Risks
  - Hard job
  - Well-known subject with lots of misconceptions
  - Hard to find qualified staff
  - Easy to set off without it, creating a barrier to installing it later

Risk Management

- Advantages
  - Vastly improves chances project will succeed
  - Can be simple and effective
  - Matches reality
- Risks
  - Some are “risk-averse” and don’t even want to talk about it
  - Prone to lapse into incongruent interactions (blaming, placating)
  - Relatively new to software
  - Dependent on Project Management & other things

Some Observations

- If you don't know how what is came to be, setting a course may be difficult and unpredictable.
  - Understand the *development* context
  - Understand the *business* context
- Consider choosing improvements based not just on potential value, but also on your chances of succeeding.
- Remember that all change is hard.

Further Reading

Walt Kelly, *The Best of Pogo*, Simon & Schuster, 1982

Jerry Weinberg, *Quality Software Management,* a 4 volume trilogy, Dorset House, 1989-95.

Jim Highsmith, *Adaptive Software Development*, Dorset House, 1999

Malcolm Gladwell, *The Tipping Point*, Little Brown, 2000.

Norm Kerth, "The Ritual of Retrospectives," *Software Testing & Quality Engineering*, September 2000, Vol. 2, No. 5

Bob King, "When Assessments are Relative," *STQE*, January 2001, Vol. 3, No. 1.

III, "Immunizing Against Predictable Project Failure," *STQE*, January 2001, Vol. 3, No. 1.

Brian Lawrence & Payson Hall, "The Problem of Project Management," *Cutter IT Journal*, Vol 12, No. 5, May 1999.

QW2001 QuickStart 2Q

Mr. Kent Beck
(Author)

Extreme Programming Explained

Key Points

- Programmers need their own tests to maintain speed and flexibility.
- These tests may improve quality enough that QA is no longer needed as a Great Wall to protect the customers against the depredations of the Mongrel Programmer Hordes.
- QA can then take the initiative in enhancing communication between business and development

Presentation Abstract

Extreme Programming (XP) violates the prevailing Taylorist assumptions of conventional software engineering. Who was Fred Taylor and why would he make such a crummy software engineering manager? What is an alternative?

About the Author

Kent Beck is the godfather of XP. He also pioneered CRC cards, the HotDraw drawing editor framework, the xUnit testing framework and (with Erich Gamma) its open source Java variant JUnit, the rediscovery of test-first programming, and patterns for software development. He leaves on 20 rural southern Oregon acres with his wife, five children, two dogs, and a variable number of domestic fowl.

Frederick Winslow Taylor

- "In the past man was first. In the future the system must be first."

Taylorism

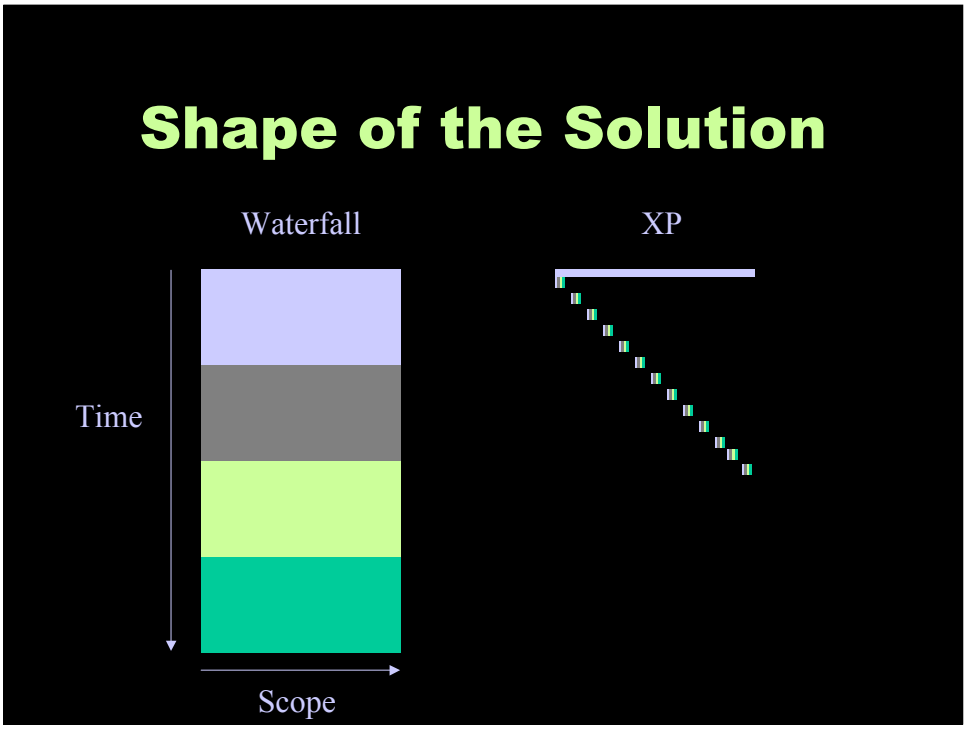
- Time studies
- Separate planning
- Instruction cards
- Selection of workmen
- Task assignment
- Quality control
- Differential rate

New Paradigm

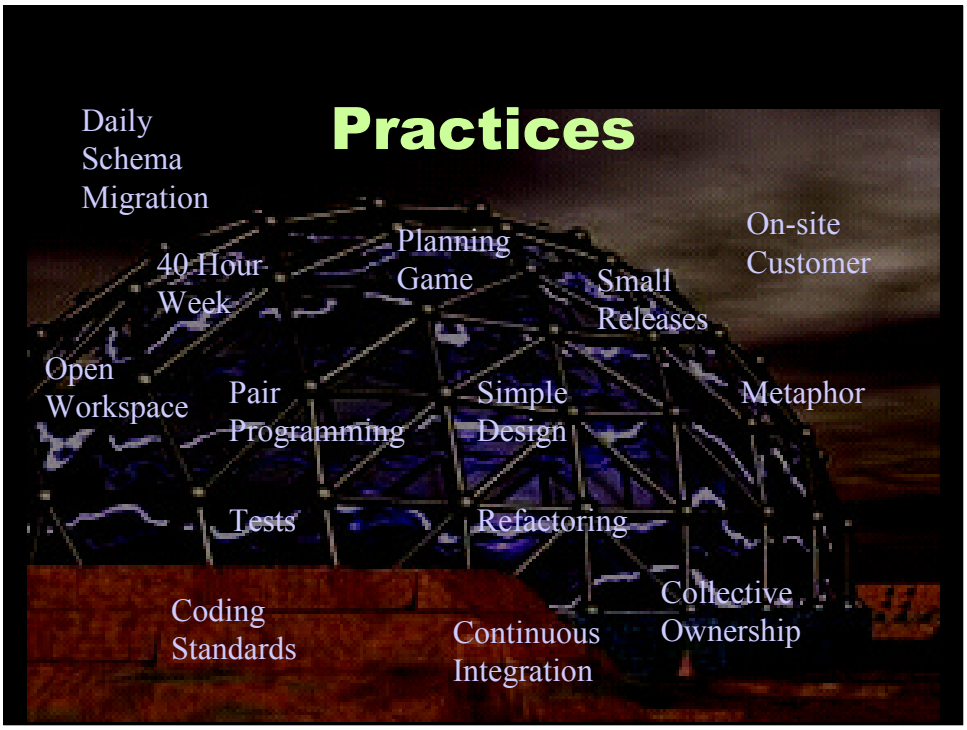
Making things

Conversation

Shape of the Solution



Practices



Learning To Steer

URLs

- www.junit.org
- www.xprogramming.com
- www.extremeprogramming.org
- www.cs.utah.edu/~lwilliam

QW2001 QuickStart 3Q

Mr. Tom Gilb
(Result Planning Limited)

Planguage: A Defined Language for Clearer Requirements and Design

Key Points

- A new requirements-and-design specification language
- Focus on stakeholder-driven value and quality
- Control over cost and time

Presentation Abstract

A formal planning language suitable for all aspects of software engineering planning, requirements, design, project planning, risk analysis, organizational improvement, quality control; has been specified. Planguage is unique. There is no other remotely similar alternative. One distinguishing characteristic is that all qualitative stakeholder values are expressed quantitatively. It is defined in free texts on a website, It has been used in practice for years in many multinational corporations. It resembles a programming language in character, but it is a higher level of specification which is particularly good at specifying the very things which programming languages are poor at specifying: quality, costs, risks, and system level relationships. Planguage is a solid and precise foundation for deriving tests from requirements and design. From this talk you will get an overview and s amples of Planguage, which you can follow up from free website materials.

About the Author

Tom Gilb was born in Pasadena in 1940, emigrated to London 1956, and to Norway 1958, where he joined IBM for 5 years, and where he resides when not travelling.

He has mainly worked within the software engineering community, but since 1983 with Corporate Top Management problems, and 1988 with large scale systems engineering. He is an independent teacher, consultant and writer. He has published eight books, including the early coining of the term "Software Metrics" (1976) which is the basis for SEI CMM Level 4. He wrote "Principles of Software Engineering Management" (1988, now in 13th printing, with 3 chapters on Evolutionary delivery methods), and "Software Inspection" (1993). Both titles are really systems engineering books in software disguise. His pro bono systems engineering activities include several weeks a year for US DoD and Norwegian DoD, and environmental (EPA) and Third-World Aid charities or organizations.

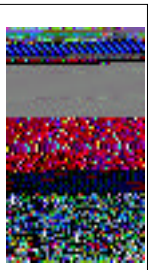
His clients include Hewlett Packard, Boeing, Microsoft, Ericsson, Alcatel, Nortel,

Oracle, Sun, British Aerospace, UK Civil Aviation Authority, Litton PRC, Siemens, Medtronic and many others.

Planguage:

A defined Language
for Clearer
Requirements and Design"

Quality Week, San Francisco
Quickstart
Wednesday 30th May 2001, PM
Tom Gilb
URL www.result-planning.com



Planguage: a defined language for clearer requirements.

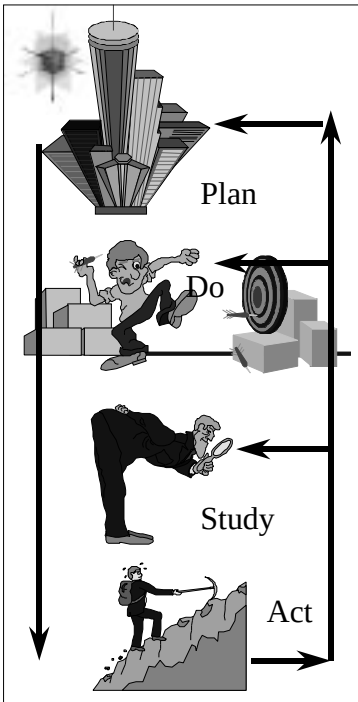
- a new requirements-and-design specification language
- focus on stakeholder-driven value and quality
- control over cost and time

Summaries

Tom Gilb
is best summarized at www.result-planning.com. He has published 8 books, including Principles of Software Engineering Management. His new book, defining 'Planguage' is forthcoming in 2001, and is free on his web site.

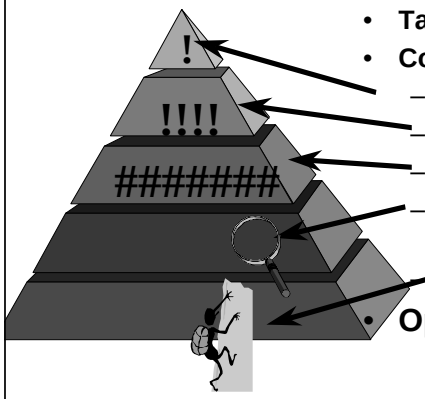
"Planguage' is
A formal planning language suitable for all aspects of software engineering planning, requirements, design, project planning, risk analysis, organizational improvement, quality control; has been specified.
Planguage is unique.
There is no other remotely similar alternative.
One distinguishing characteristic is that all qualitative stakeholder values are expressed quantitatively.
It is defined in free textbooks on a web site,
It has been used in practice for years in many multinational corporations.
It resembles a programming language in character,
but it is a higher level of specification
which is particularly good at specifying the very things which programming languages are poor at specifying:
quality, costs, risks, and system level relationships.
Planguage is a solid and precise foundation for deriving tests from requirements and design.
From this talk you will get an overview and samples of Planguage,
which you can follow up from free web site materials.

The Planguage Cycle



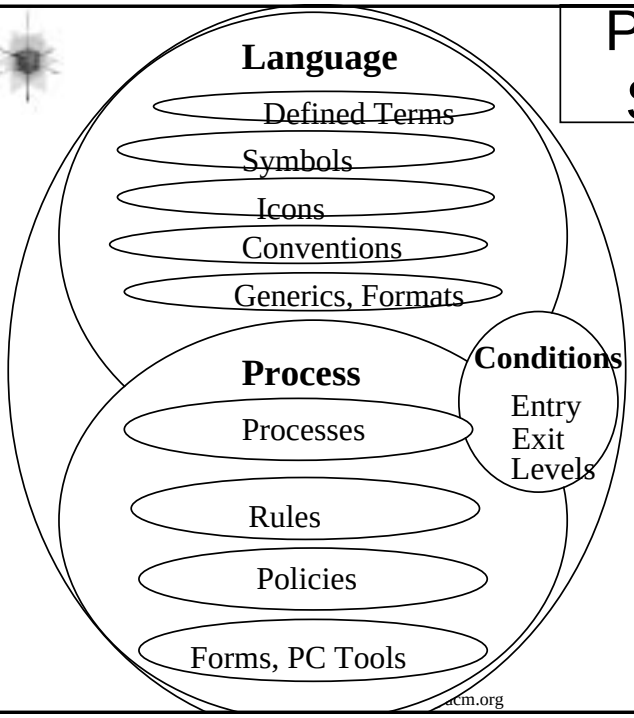
- **Define Requirements**
- **Design to meet requirements**
- **Quality Control of design/plans**
- **Build the design**

‘Planguage’ Summary



- **Meta-Process**
- **Generic**
- **Tailorable**
- **Cornerstones**
  - **clear requirements**
  - **multiple qualities & costs**
  - **quantified design accounting**
  - **Specification Quality Control/Inspection**
  - **Evolutionary project management**
- **Open-ended**

Planguage Structure



Quality Defined for Comp. Eng. book

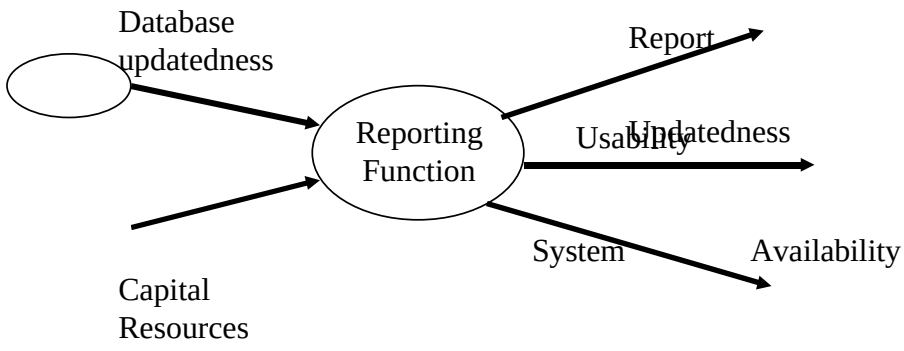
- **Quality (noun) Concept *125**
- **A quality is a stakeholder-valued attribute of a system.**
- **If no stakeholder is interested in the attribute, then we would not be interested in classifying it as a 'quality'.**
- **All systems have a large number of quality attributes in practice. That is, they have a large number of 'dimensions of goodness' or 'valued characteristics' which are the concern of some stakeholders.**
- **It is fundamental to systems engineering, and management, that we identify our critical stakeholders, and their critical needs, in terms of requirement levels of selected attributes. In short we must understand our stakeholder's quality requirements.**
- **The concept of a 'quality' is also needed in order to distinguish these characteristics from other central system engineering system descriptors; such as functions, costs, designs, constraints and all other concepts.**

Quality is distinguished by us, from these others, in one or more of the following ways: (from Concept *125 CE Glossary version Jan 26 2001 TG

- it is valued to some degree by some stakeholders in the system
- it is variable (along a definable scale of measure)
- it is capable of being specified quantitatively
- it can be measured in practice
- more of it is generally valued by stakeholders, especially if the increase is free or lower cost than the value of the increase.
- it can never be perfect, in the real world
- it is independent of the particular means (designs) for reaching a particular level
- it can be a complex notion, consisting of many elementary quality concepts.
- it can be traded off to some degree, given limited resources for producing qualities, for other quality levels which are valued more by a defined stakeholder.
- as quality levels increase towards perfection, the resources (a 'cost' concept) needed to support those levels tend towards infinity.
- there are some levels of a particular quality which may be outside the state of the art, at a defined time and circumstance.

System dimensions

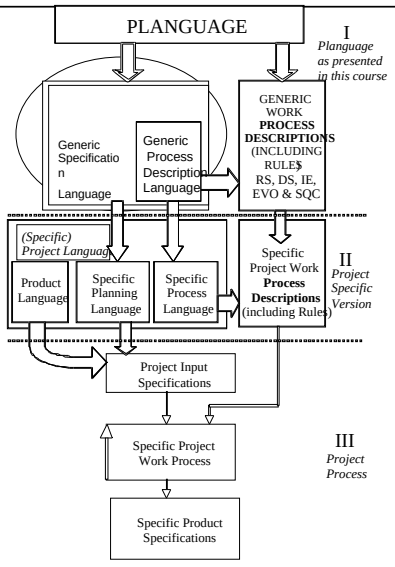
- **Resources** **'Qualities'**



Planguage Process Overview

- **Software Engineering Tools**
  - **Requirements Engineering:**
    - **A defined planning language, quantified quality**
  - **Design:**
    - **quantitative impact estimation of design on requirements**
  - **Specification Quality Control (Inspection)**
    - **quantified approach to engineering documentation**
  - **Evolutionary Project Management:**
    - **frequent feedback and learning-based management**
    - **A 'revolution' in testing approach:**
      - **Continuous system integration**

Planguage Components



Standards: Purposes

- Train newcomers
- Capture wisdom and experience for all
- Lay basis for systematic process improvement
- Lay basis for stable systems: predictable output
  - Less unnecessary individual variation
  - Basis for statistical process control (SPC, CMM5)
- Presentation to clients (how professional we are)
  - A differential to client's own practices, so they want to use us
- To capture general customer needs and specific customer needs in a systematic way
- A basis for auditing processes (do we really follow our best practices?)
- A basis for Inspection (Spec QC)
  - -measurement of specification quality versus standards
  - Decision to exit and enter engineering processes based on objective economics
  - Inspection is a major teaching and motivation device for good practices

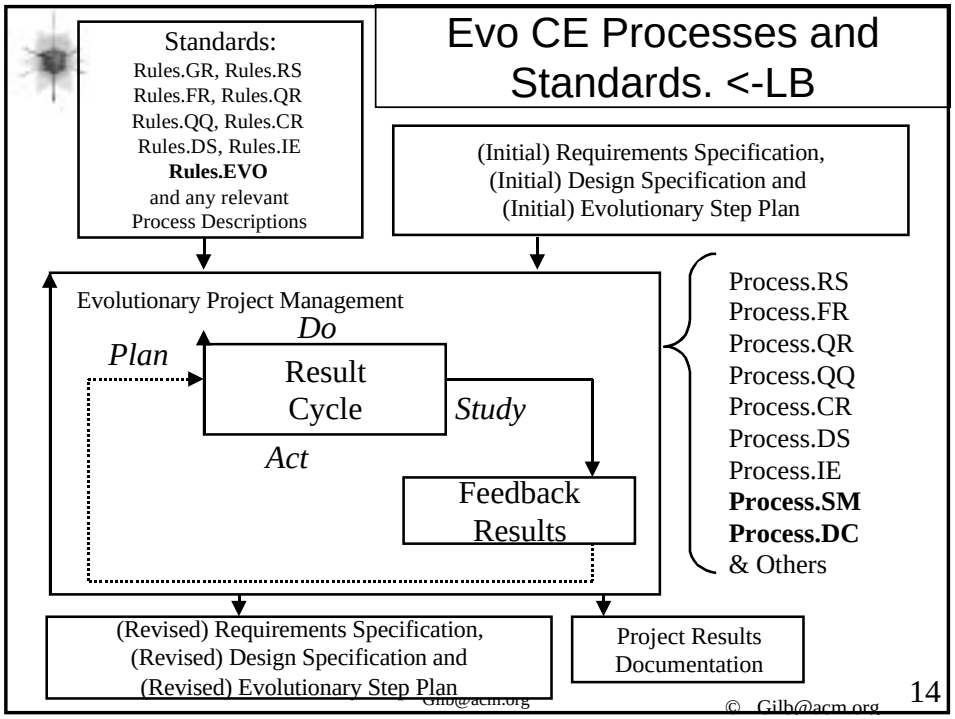
Standards Types

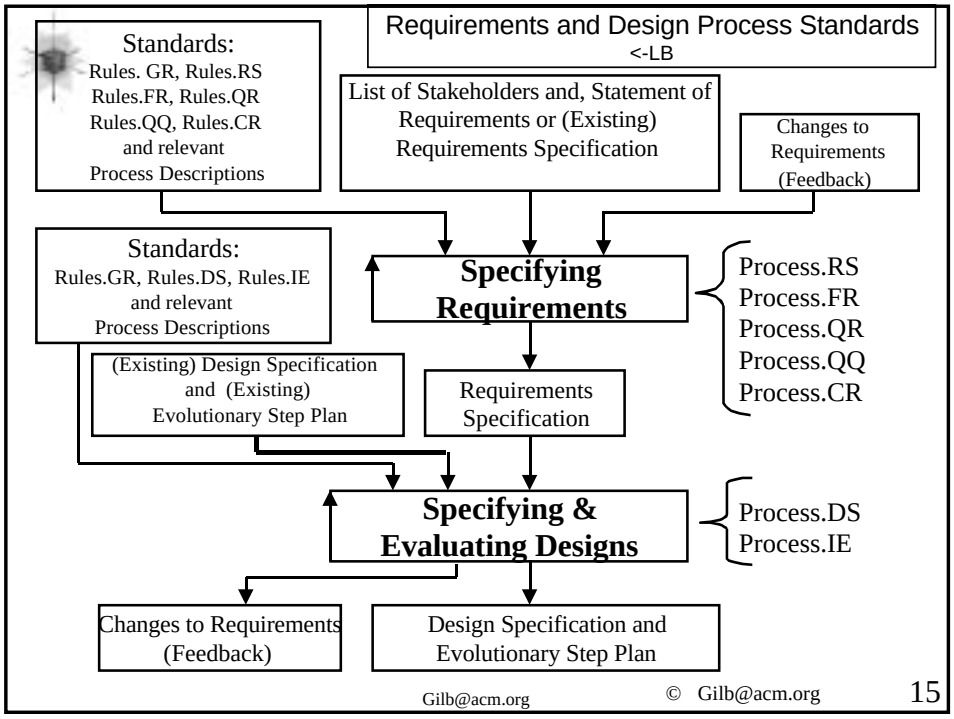
- **Models: best practice examples realistic**
- **Templates: predefined structure or lists**
- **Forms: information collection procedure**
- **Rules: required specification method, content, format**
- **Process Descriptions: how should we do our work?**
  - **Entry Conditions: when are we ready or not ready to work?**
  - **Procedure description: what sequence and activity to do work?**
  - **Exit Conditions: when are we finished? Objective numeric conditions.**
- **Checklists: help to interpret rules and find defects during inspections.**
- **Rates: recommended speeds of working for optimum human performance (example: speed of checking pages/hour)**
- **Defined terms: Glossary for precise communication**
- **Courseware: slides exercises etc.**

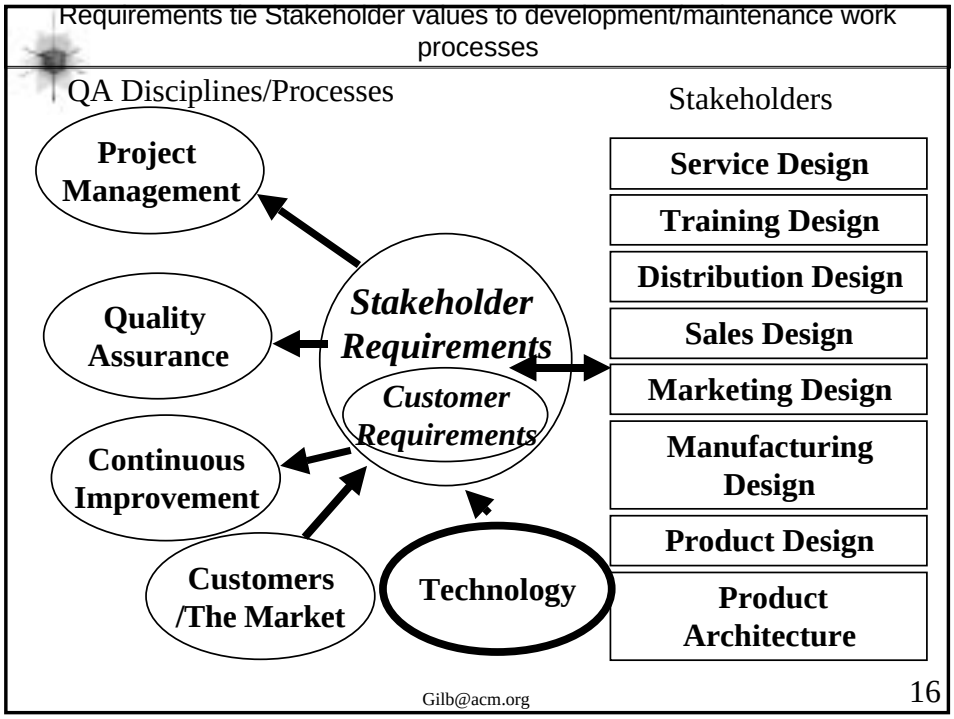
Quality Control Types

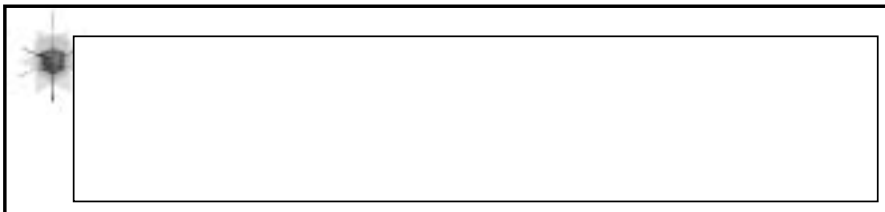
- Individual Examination: Check your own work (for example against rules)
- Peer Reviews
  - Buddy Checking (a friend checks your work)
  - Inspections: have we followed our standards? (Current focus)
    - Sampling to measure
    - 100% to clean up
  - Content Reviews: is the work good enough?
- Customer Reviews
  - Do they understand the work?
  - Do they like the work
  - Do they formally approve the work
- Testing
  - Detail to be supplied, not my concern now TG
- Field
  - Initial field trials
  - Longer term field experience and feedback

Evo CE Processes and Standards. <-LB




Requirements and Design Process Standards
<-LB
Standards:
Rules. GR, Rules.RS
Rules.FR, Rules.QR
Rules.QQ, Rules.CR
and relevant
Process Descriptions
List of Stakeholders and, Statement of Requirements or (Existing) Requirements Specification
Changes to Requirements (Feedback)
Standards:
Rules.GR, Rules.DS, Rules.IE
and relevant
Process Descriptions
Specifying Requirements
Process.RS
Process.FR
Process.QR
Process.QQ
Process.CR
(Existing) Design Specification and (Existing) Evolutionary Step Plan
Requirements Specification
Specifying & Evaluating Designs
Process.DS
Process.IE
Changes to Requirements (Feedback)
Design Specification and Evolutionary Step Plan
Gilb@acm.org
© Gilb@acm.org
15


Requirements tie Stakeholder values to development/maintenance work processes
QA Disciplines/Processes
Stakeholders
Project Management
Quality Assurance
Continuous Improvement
Customers /The Market
Stakeholder Requirements
Customer Requirements
Technology
Service Design
Training Design
Distribution Design
Sales Design
Marketing Design
Manufacturing Design
Product Design
Product Architecture
Gilb@acm.org
16



- *4.14 Stakeholder*
- *An interested party having a right, share or claim in the system or in its possession of qualities that meet their needs.*

← **ISO/IEC 152881, TC /SC /WG , Secretariat**

Stakeholder: (Planguage Glossary)

- **People, group, or any object**
  - **which has some direct or indirect 'interest' in the outcome of a defined process or product.**
  - **We would also select stakeholders as ones we have some interest in listening to.**
  - **Stakeholders have requirements which are critical or profit-impacting for your project**

For example they might have an interest in

- ***1. Setting the objectives for a process.***
- ***2. Evaluating the quality of the product***
- ***3. Using the product or system, even indirectly***
- ***4. Avoiding problems for themselves as a result of our product or system.***
- ***5. Being compatible with another machine or software component.***

The Planguage parameter term 'Stakeholder'

- ***can be used to specify one or more stakeholders explicitly.***
  - *Stakeholder = {End User, Help Desk, Installer}*
- ***We can attach stakeholder information to any elementary specification,***
  - Plan [Stakeholder = Novice User] 10 minutes
- ***or to a*** set ***of specifications,***
  - Scale [Installers] time for successful installation
  - Must 20 minutes, Plan 10 minutes, Wish 5 minutes.
- ***as appropriate.***

Stakeholder Types: Example from real customer requirements definition about 1996, USA

- **Government FCC**
- **Telecompany Corporate**
- **DEVELOPER**
- **MANUFACTURER**
  - **See detail next slide of probable values/requirements**
- **OPERATOR (like AT&T)**
- **DISTRIBUTION**
- **LEASING/PURCHASE**
- **PHONE USER:**
- **System Owner (in office)**
- **MAINTENANCE: Employees of system owner**
- **Responsible Site Administrators**
- **Responsible Installers**
- **Repair Centers**

Manfacturing Stakeholder (detail)

Example from real customer requirements definition about 1996, USA

- **MANUFACTURER: some potential requirements areas**
  - **Lynch-Town or elsewhere?**
  - **Like to manufacture in this country, avoid tolls taxes**
  - **Need to invest in capital equipment**
  - **Just in time purchasing and manufacturing**
  - **Re-use of existing components**
  - **Ease of manufacture of components**
    - **Ease of setup**
    - **Training**
    - **Small runs**
  - **Ease of Assembly**
  - **Ease of tailoring to special orders**
  - **Ease of testing products (in direction of minimization of need)**

(2001) Real Example Stakeholder spec

BT OPP Integration:

Summary :The XXX-999. would integrate both 'Push Server' and 'Push Client' roles of the BT Object Push Profile.
Type: Architectural Constraint.(requirement)
Source: 3.4.3 Integration of BT OPP… in BT Application Requirements Study Version 0.1 March 9 2001

Stakeholders: " who we are writing this particular requirement for"

Phonebook, Scheduler, Testers, <Product Architect>, Product Planner, Software Engineers, User Interface Designer, Bluetooth Team Leader, **Our Co. Bluetooth engineers, Bluetooth Developers from other Our Co. product departments which we interface with, the supplier of the software {Texas Instruments, Condat**.

Description:

Comply:

A defined [XXX-999. software]
Acts in accordance with to the <specification> defined in the
Defined subject [for both Push Server and Push Client roles of the BT Object Push Profile (OPP).] ,
in the following defined way: .
for [[BT, XXX-999.]:
Official certification is actually and correctly granted; before
{developer or supplier or any real integrator, whoever it really is doing the integration}
has completed their task correctly.
This includes correct proven interface to any other related modules specified in the Specification.

- **Impacts Section. For BTT OP Integration**
  - **Impact A:**
    - **Impact Assertion: <100% of <Interoperability> objective with other BT devices that support OPP on time is estimated to be the result>. <Information about measurement, basis for estimate should also be given>.**
    - **Interoperability: Defined As: Certified that this device can exchange information with any other Bluetooth device.**
    - **Assumption: there are some quality requirements for BT certification regarding probability of connection and transmission etc. we do not remember what they.<-TG**
    - Risks:
      - 1. We do not 'understand' (do dot have information in hand here) fully the BT Certification requirements, so we risk that our design will fail certification. <-TG
    - **Sources:**
      - **Specifications of the Bluetooth System volume 1 version 1.1, Promoters Members of the Bluetooth SIG, Inc. ("Bluetooth SIG"), February 2001**
        - » Precise reference <to be supplied by Andrea>
      - **Specifications of the Bluetooth System volume 2, version 1.1, Promoters Members of the Bluetooth SIG, Inc. ("Bluetooth SIG"), February 2001**
        - » Precise reference <to be supplied by Andrea>

What's New in Planguage ?

- **Total quantification of all quality requirements**
  - **'Scale: relative eng. Hours to port to new env. '**
- **Quantified estimates of design impacts**
  - **Design-x --> 30% Plan [30,000 h MTBF]**
- **Quantified Spec Quality Control (Inspection)**
  - **Exit OK: Max. 0.2 Major defects/page remain.**
- **Multiple-dimension quantified project management**
  - **All qualities and costs impact per 2% cycle**

12

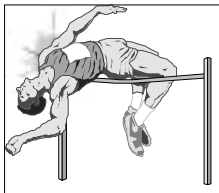
Contents: ‘Competitive Engineering’ (the Planguage handbook)

- **Defined systems engineering language: “Planguage”**
  - **covers: requirements, design, spec QC, project management**
  - **Specification Rules defined for QC purposes**
  - **Engineering processes defined (with Entry , Exit)**
  - **100 Principles defined**
  - **425+ integrated concepts defined**
  - **User-tailorable, continuous improvement**
  - **free handbook on web, no strings attached.**
    - Get manuscripts, papers, cases and slides free on www.result-planning.com

Principles: Design engineering process.

- 0. THE PRINCIPLE OF 'IDEAS ARE ONLY AS GOOD AS THE REQUIREMENTS SATISFIED'.

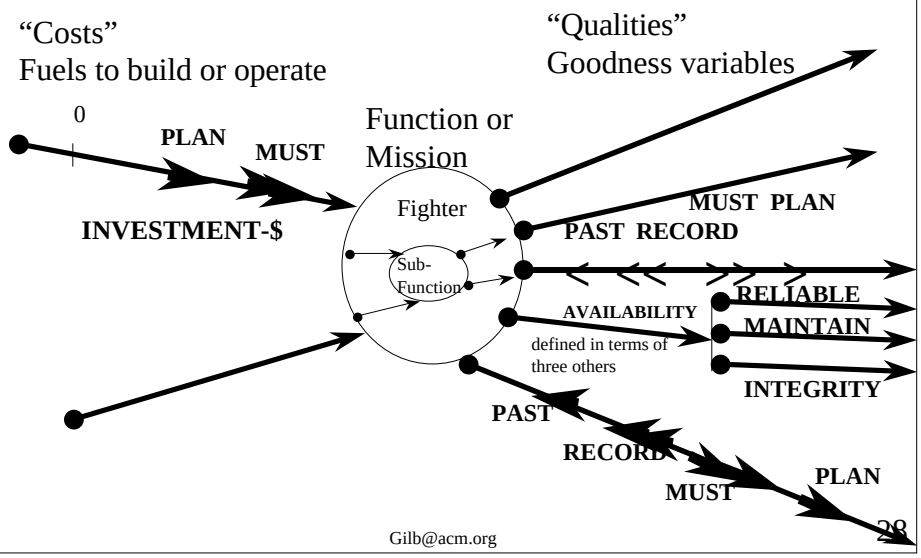
- Design ideas cannot be judged or validated except with respect to all quality and cost requirements they must satisfy.



1.THE PRINCIPLE OF 'REALITY BEATS THEORY'.

- Design ideas are only as good as their actual implementation, not their intent.

Some Planguage Graphical Icons



Part 1. Requirements Engineering

- **Requirements as 'End states' not means.**
- **All variable quality ideas quantified**
- **Advanced specification of *quality levels*, "when" , 'where" and "ifs" for a requirement.**
- **Absolute testability of all requirements**
- **Configuration management is built in**
- **"Uncertainty" and "risk" is explicitly specified**
- **Intimately tied to "design", QC, project control**
- **'Learning feedback' from delivery cycles**

Blank Requirement Template. V=012603

Requirement Tag:
Ambition:
Type: Requirement
Stakeholders: { }
Version:
Owner:
Scale:
Meter [ ]
====Benchmarks ========== the Past
Past [ ] <--
Record [ ] <--
Trend [ ] <-
===== Targets ========== the future value and needs
Wish [ ] <-
Must [ ] <--
Plan [ ] <--
Stretch [ ] <-

Electronic Requirements template with hints:

V=012503

<name tag of the objective>

Ambition: <give overall real ambition level in 5-20 words>

Type: <quality|objective|constraint>

Stakeholder: { , , } "who can influence your profit, success or failure?"

Scale: <a defined units of measure, with [parameters] if you like>

Meter [<for what test level?>]

====Benchmarks ========== the Past

Past [ ] <estimate of past> <--<source>

Record [<where>, <when record set> <estimate of record level>] <-- <source of record data>

Trend [<future date>, <where?>] <prediction of level> <- <source of prediction>

===== Targets ========== the future value and needs

Wish [ ] <- <source of wish>

Must [ ] <-- <source>

Plan […] <target level> <-- Source

Stretch [ ] <motivating ambition level> <- <source of level>

Other useful Parameters
see CE book index and Glossary for detail. V 14April01

- **Assumptions:**
- **Authority:**
- **Source:**
- **Risks:**
- **Dependencies:**
- **Impacts:**
- **Impacted By:**
- **Resources:**
- **Priority:**
- **Responsible:**
- **Test Plan:**
- **Test Cases:**
- **Sponsor:**
- **Initiator:**

Integratability (Real Example)

- **Gist: (Ca.): better ease of integration than most competitors**
- **Stakeholder: {Independent Software Vendors, Systems Integrators, Ourselves, …}**
- **Scale: time it takes a defined number of people to <integrate>**
- **Must [Independent Software Vendor, Highest Complexity Task, 1 person …] <24 hours> ?? Guess**
- **Plan [Independent Software Vendor, Highest Complexity Task, 1 person …] 1 hour**

An example of definition: "Parameter Types : definitions"

TEST-EFFECTIVENESS <-Tag of defined spec.

Ambit<-Parameter type ***portion of defined defect types are*** Definition> ***ading using our test processes?***

SCALE: % <-Parameter type FECTS identified by DEFINED TEST PROCESSES within DEFINED EFFORT using DEFINED TOOLS <-Definition>

METER: <-Parameter type sampling of test reports by QA>

PAST [EC<-Parameter type BRANCH COVER<-Definition>STING, 80% level coverage, SRA Tools] 50% + or - 30%?? <- Tom Gilb wild Definition> **illustrative guess to p<-Definition> better information.**

PLAN [ANY DESIGN DEFECT, {SQC and Our Tests}, 1 work hour per Definition> **NO SPECIAL TOOLS] 30% ?? <- swag tg**

DEFECTS:DEFINED: any difference from formal requirements.

Definition>

DESIGN DEFECT: DEFINED : any difference from specified design.

An example of definition: Test Effectiveness Measures

TEST-EFFECTIVENESS

Ambition: ***enhance greatly portion of defined defect types are we finding using our test processes?***

SCALE: % of DEFINED DEFECTS identified by DEFINED TEST PROCESSES within DEFINED EFFORT using DEFINED TOOLS

METER: <sampling of test efforts by QA>

PAST [LOGICAL BUGS, BRANCH COVERAGE TESTING, 80% level coverage, SRA Tools] 50% + or - 30%?? <- Tom Gilb wild illustrative guess to provoke better information.

PLAN [ANY DESIGN DEFECT, {SQC and Our Tests}, 1 work hour per 100 LOC, NO SPECIAL TOOLS] 30% ?? <- swag tg

DEFECTS:DEFINED: any difference from formal requirements.

DESIGN DEFECT: DEFINED: any difference from specified design.

Requirements Analysis (real) Example:
An aircraft company bid for $60 million CAD/CAM Bd.of Dir.

A special effort is underway to improve the *timeliness* of Engineering Drawings. An additional special effort is needed to *significantly improve drawing quality* .
This Co.. establishes a Natural Work Group on Engineering Quality (EQNWG) to lead Engineering to a breakthrough level of quality for the future. To be competitive, Our Co.. must *greatly improve productivity*. Engineering should make major contributions to the improvement. The simplest is to reduce drawing errors which result in the AIR (After Initial Release) change traffic that slows down the efficiency of the manufacturing and procurement process. Bigger challenges are to help make CAD/CAM a universal way of doing business at Co.., effective use of group classification technology, and teamwork with Manufacturing and suppliers to develop and implement truly innovative design concepts that lead to *quality products* at lower cost. The EQNWG is expected to develop "end state" concepts and implementation plans for changes of organization, operation, procedures, standards and design concepts to guide our future growth. The *target of the EQNWG is breakthrough in performance* not just "work harder". The group will phase their conceptualizing and recommendations *to be effective in the long term* and to influence the large number of drawings now being produced by TRANSPORT CARRIER and Model -11 design teams

Requirements Classification

organize

OBJECTIVES

DESIGNS

PLANS

FUNDING

RESPONSIBILITY

METHOD

Functional Classification

Quality Requirement Classification

clarify Ambition level

Functional Areas "WHAT we do"
F1: Engineering Drawings
F2: "TRANSPORT CARRIER .. design team"
F3: "Model -11 design team"

Quality Objectives. "ENDS-- how well we do what we do'"
Q-LT-EFFECT: "to be effective in the long term "
Q-PERFORM: "The target of the EQNWG is breakthrough in performance."
Q-DRAW-PLAN: "Significantly Improve Drawing Quality "
Q-PROD: "Productivity."
Q-QUALITY: "quality products"
Q-TIME: "Timeliness"

Al Says ….

“Perfection of means

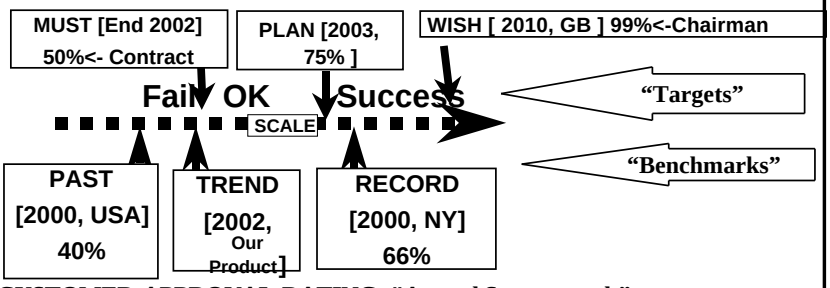
And confusion of ends

Seem to characterize our age..”

Albert Einstein, found on the www 2000

Http://albert.bu.edu BostonUniversity

Scale-of-Measure Concepts



- **CUSTOMER-APPROVAL-RATING: "Annual Survey result”**
- **Ambition: be almost as good as state of the art in customer happiness.**

SCALE: Percentage 'YES' answers on average of all questions.

METER: The Annual Gallup Survey for our market niche.

PAST[2000, USA] 40%, [1999, Europe] 55.5%<-Euro Gallup

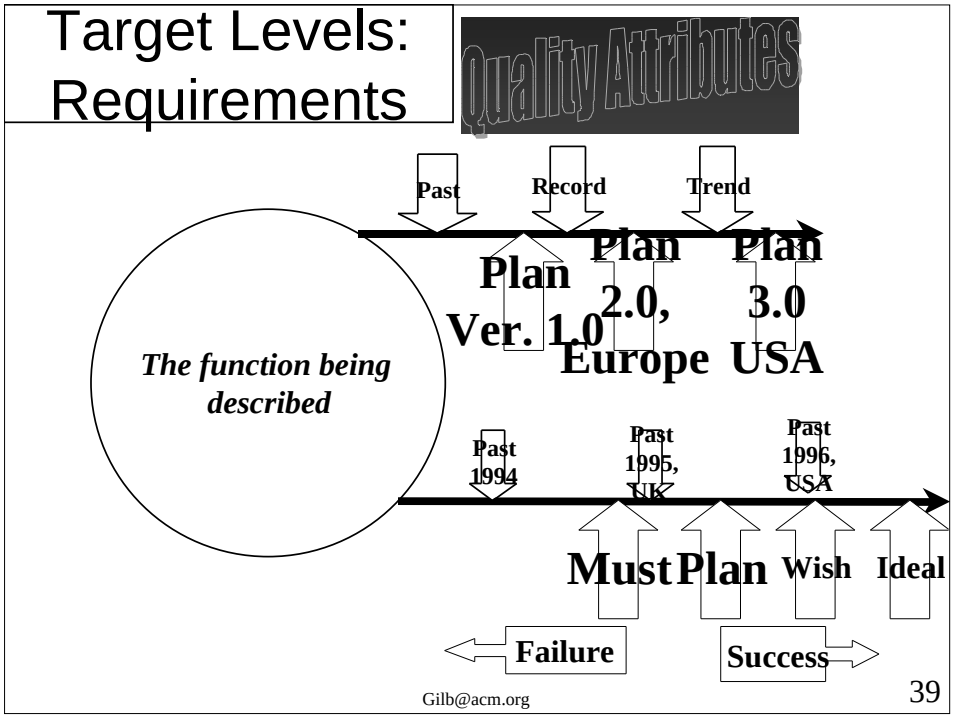
RECORD[2000, New York] 66%, [1994, Oslo] 70%

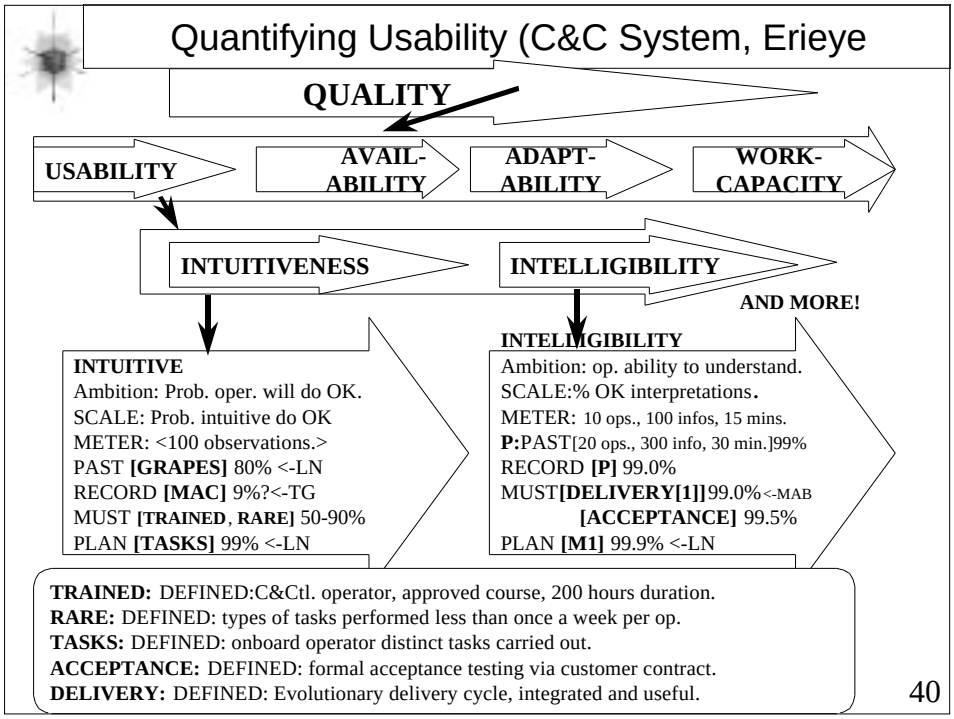
TREND [2002, Our Product] 35% <-Tom’s Guess

WISH [Long term future] 99% <-Board Policy.

MUST[End 2002, USA] 50% <- Corp. Marketing Plan page 6.

PLAN[2002,Worldwide] 52%, [2003] 75%, [2004] 95% <- Five year plan 5.4.


Target Levels: Requirements
Quality Attributes
Past
Record
Trend
Plan Ver. 1.0
Plan 2.0, Europe
Plan 3.0 USA
The function being described
Past 1994
Past 1995, UK
Past 1996, USA
Must
Plan
Wish
Ideal
Failure
Success
Gilb@acm.org
39


Quantifying Usability (C&C System, Erieye
QUALITY
USABILITY
AVAIL-ABILITY
ADAPT-ABILITY
WORK-CAPACITY
INTUITIVENESS
INTELLIGIBILITY
AND MORE!
INTUITIVE
Ambition: Prob. oper. will do OK.
SCALE: Prob. intuitive do OK
METER: <100 observations.>
PAST [GRAPES] 80% <-LN
RECORD [MAC] 9%?<-TG
MUST [TRAINED, RARE] 50-90%
PLAN [TASKS] 99% <-LN
INTELLIGIBILITY
Ambition: op. ability to understand.
SCALE:% OK interpretations.
METER: 10 ops., 100 infos, 15 mins.
P:PAST[20 ops., 300 info, 30 min.]99%
RECORD [P] 99.0%
MUST[DELIVERY[1]]99.0%<-MAB
[ACCEPTANCE] 99.5%
PLAN [M1] 99.9% <-LN
TRAINED: DEFINED:C&Ctl. operator, approved course, 200 hours duration.
RARE: DEFINED: types of tasks performed less than once a week per op.
TASKS: DEFINED: onboard operator distinct tasks carried out.
ACCEPTANCE: DEFINED: formal acceptance testing via customer contract.
DELIVERY: DEFINED: Evolutionary delivery cycle, integrated and useful.
40

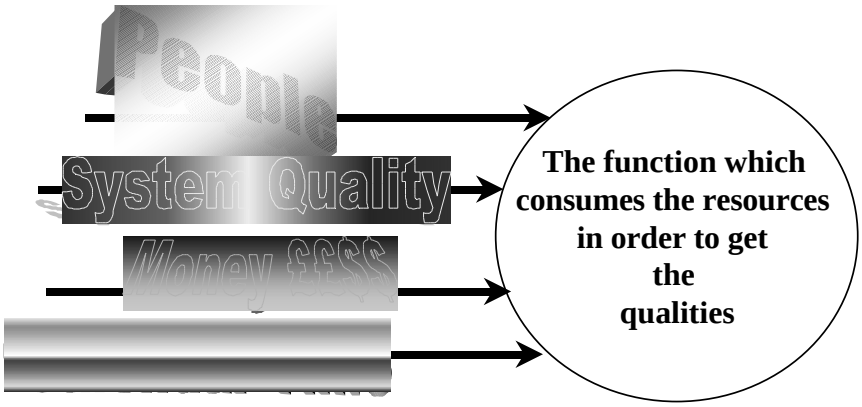
Uncertainty Notation

- **There is a wide variety of notation to express uncertainty, risk of deviation, fuzzy thought etc.**
  - **[Qualifiers] can reduce uncertainty by limiting the conditions for which a specification is valid**
  - **Past [USA] <50%> < fuzzy brackets>**
  - **Past [USA] 50% ±20%**
  - **Past [USA] 50% to 60%**
  - **Past [USA] 50%? "Or ??"**
  - **Plan [USA, IF Copyright Valid] 50%**

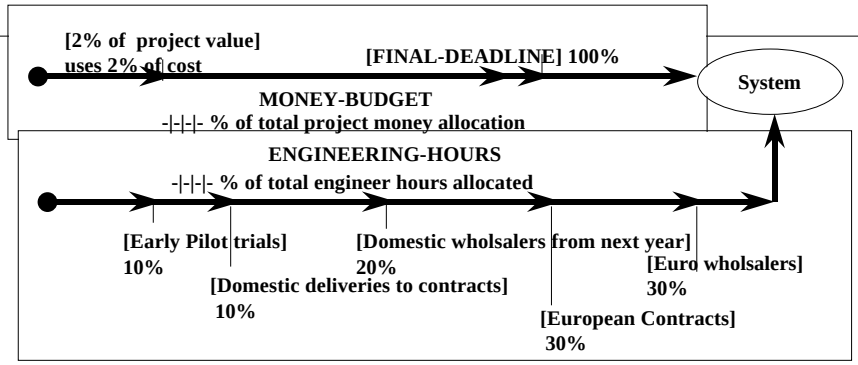
Assumptions

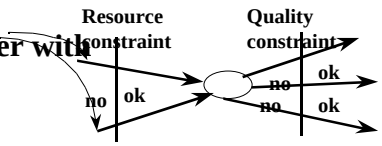
- **Assumptions are any condition for other specifications to be valid.**
- **A1: Assumption: the weather is suitable.**
- **Must [Suitable Weather] 55%.**
- **Plan [2001, If Suitable Weather OR Indoors]1.**
- **S22: State: Off.**
- **Stage1: Basis: Unit Tests AND QC Exited.**

‘Resources’: system inputs



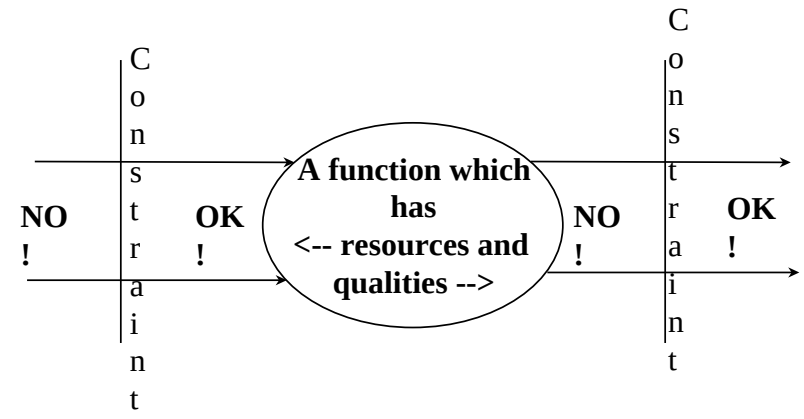
Example: Budget Specification



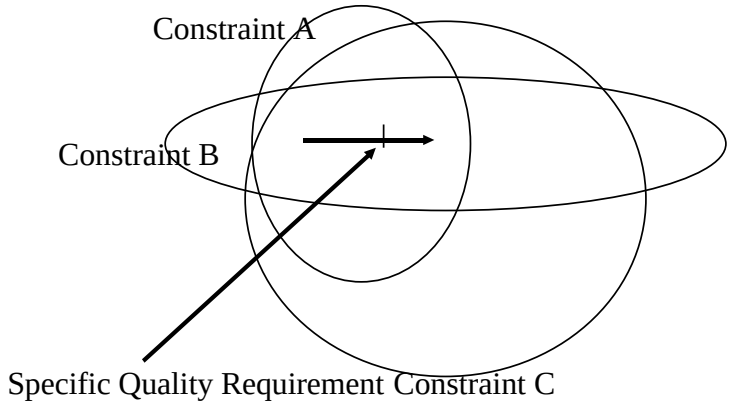


Multiple resource budgets together with multiple quality requirements and global constraints.
They can all be specified, designed and evaluated simultaneously!

Some global Constraint Concepts



A global Constraint restricts the designer; then the other requirements need to be considered



Resource Constraints (Real examples) 1

Installation $ Cost:

SCALE: Total Installation Cost of all involved parties.

Total Installation Cost: DEFINED:

{education of customer people, involvement of customer people during installation, during planning, TeleCo, Loss of Service in a PBX, Special Tools for Strange Cabling, any other thing even if not on this list!}

MUST [per installation, USA, Release [1]] $ <??> Maximum twice DECT Installation costs.

PLAN [per installation, USA, Release [1]] $ similar (within 20%) to DECT Installation costs.

Per User Price:

Note: this is a price-border constraint, the actual price targets may vary from time to time and market to market.

SCALE:$ per user price to customer for total Base Station {CE and RH}.

PLAN [30 to 250 users system, USA, Release [1]] $700 <--RSW 2

PLAN [more than 250 users OR larger building OR tougher than normal radio , USA, Release [1]] >$700 <--RSW 2.

Resource Constraints (Real examples) 2

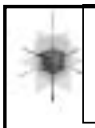
- **Installation Time:**
  - **Ambition: must not be more than of an unlicensed system.** <--**RSW 3**
  - **SCALE: Work Hours:**
  - **PLAN**
  - **SCALE: Calendar Days.**
  - **PLAN**
  - **NOTE: a detailed table of these timings was included on MRS 4.5.3**

- **Subscriber Cost**
  - **Note: this is a cost-border constraint, the actual cost targets may vary from time to time and market to market.**
  - **SCALE: $ cost for a [defined # of users] system per subscriber, including TK and SW licenses cost to TeleCo.**
  - **MUST [100 users, USA, Release [1]] $400 <--RSW 2, page 2 Cost Assumptions.**

Gilb@acm.org

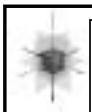
Legal Constraints (Real examples)

- **E911 [USA]:**
  - **Any user must be able to get emergency by dialing 911<--MRS 4.6.1 and laws.**
  - **[NOT USA] the corresponding emergency number must be able to be used.**
  - **←Marketing and sales and distribution**
- **Sales-Process: <--RSW 2**
  - **Ambition: different from XXX 88XX**
  - **We can sell to a distribution channel (internal or external), who sells to customer**
  - **(OR) We can sell to an operator (sale or leasing who sells to distribution channel.**
- **Sales-Category:**
  - **the product will be sold as a Wireless PBX/Key system. <--RSW 2**
- **Replacement:**
  - **the product will be offered as a replacement for a Fixed or Cordless Private System. <--RSW 2**
- **Coexist:**
  - **if not Replacement, the system will be offered as a wireless office system that will coexist with an existing PBX <--RSW 2**



"Political" Constraints (Real examples)

- **Operator-Acceptance**: The product must be accepted by the operator.
- **Buyer-loves-it**: the office or company which buys it, must "love" it.
- **TeleCo Documentation**: we will map this documentation onto existing company documentation categories as far as possible ← TW.



Rules Frameworks

Rules Section

- Purpose of rules:
  - To formally define the best practices for specification
  - To be a vehicle for teaching to newcomers
  - To be used in inspections to determine 'defect'
  - To enable systematic measurement of specification quality
  - To enable measurement of process change or improvement
  - To enable measurement of processes (like 'design')
  - To enable economic evaluation of engineering activity
    - Does it pay off
    - Is it getting more or less efficient
    - Are the changes working or not?
    - To give individual engineer some consciousness of their economics of work process
  - Not to constrain creativity in doing even better things!
    - But maybe these ideas should be captured somewhere in standards (models, rules, checklists, templates, forms) so others can benefit from them!

Policy for Rules

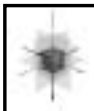
- **POLICY:**
  - **Version 4 April 2001, Origin Date: April 2 2001 Tampere.**
  - **Owner IW**
  - **Identification: Company XX.Policy.Rules**
  - **Rules should be brief.**
  - **Rules should be significant (engineering important)**
  - **Rules should be very well written for intelligibility**
  - **Rules should be supported by a variety of devices: models, checklists, templates, forms (and not try to do everything**
  - **Rules should serve the larger long term stakeholder value and economic interests (we should invest in things which have a good return on that investment).**

How to get Rules Accepted?

- **They are official Company XX standards of best practice**
- **They are acknowledged to not be perfect**
- **Anybody can argue for improvement to the process owner**
- **They are taught at induction training**
- **They are justified by the formal 'justification'**
- **People learn from inspections that their peers respect these rules**
- **A formal numeric Exit (also Entry) condition is set for release (OK) of any ones specifications, for example "Maximum One Major Defect/Logical Page remaining." This sends a clear daily message about taking rules seriously.**

Rules for Rules

- Version: 2 April 2001, Owner: IW, ID: Rules: RFR
- 1. No rules set for Inspection shall ever exceed 1 physical page
  - Justification: force us to keep them brief, significant, useful
- 2. Rules should be justified. Why are they good?
- 3. Rule examples should be given (one liners).
- 4. Some information about rules needs to be kept in the Rules Master File, some is used selectively in different connections (a Rules List, Teaching Aids, Meetings to discuss rules changes etc.)
- 5. Information in Master File should include:
  - Filename, owner, Version, Date, Rule identification, Rule name, Rule specification, rule justification, rule examples (good and bad), rule severity classification, rule change proposals, cross reference to checklist questions supporting the rules, references to literature describing the ideas of the rules, data about rule violations (frequency of defects as collected by inspection data), rule source, rule authority…. Any information pertaining to the rules in addition to this.
- 6. Information for Teaching should include
  - Rules Specification, Rule justification, Examples
- 7. Information for Inspection should include:
  - Rule Identification, Rule Name, Rule specification, Examples



Rules TEMPLATES

Rule Template

- **Rule set Identification: <file name, intranet location>**
- **Rules owner: <email>**
- **Scope:**
- **Version:**
- **Originated: <date>**
- **Updated: <date>**
- **Rule Prefix: Company XX.Rules.<name of these rules>**
  - **<rule number>, <Rule Name>: <Rule Specification>**
  - **<Rule Example>**
- Interpretation note.
- Reference to Checklist Questions:

Rules: Owner Set Template (the Rules database)

Version 10 April 2001

- **Filename**
- **Owner**
- **Version**
- **Date**
- **Rule identification**
- **Rule name**
- **Rule specification**
- **Keywords:**
- **Links:**
- **Rule justification (Rationale, Why this rule?)**
- **Rule examples (good and bad)**
- **Rule severity classification**
- **Rule change proposals**
- **Cross reference to checklist questions supporting the rules**
- **References to literature describing the ideas of the rules**
- **Data about rule violations (frequency of defects as collected by inspection data)**
- **Rule source**
- **Rule authority**
- **Any information pertaining to the rules in addition to this.**

EXAMPLES OF USE OF RULES

Rules: Owner Set Example

Version 2 April 2001

- Filename <where you store this rules detail in intranet>
- Owner: IW
- Version: 0.1
- Date: 2 April 2001
- Rule identification: Company XX.Rules.Data Sheet.1
- Rule name, User View
- Rule specification:
  - The Data Sheet is limited to describing things the User can see and test.
- Rule justification: Because the Use has no way of verifying the other types of data, so they will just confuse the User.
- Rule examples (good and bad),
  - Examples: Types to include: Functional Description, Signal Description, Electrical Characteristics, Address Map Description, Register Description, Connection of Production Test Signals.
    - NOT to include: Implementation Details like Internal Structure and Signals, Detailed Production Test information.
- Rule severity classification: Major
- Rule change proposals: -
- Cross reference to checklist questions supporting the rules: -
- References to literature describing the ideas of the rules: Courseware LED24 Data Sheet, …...
- Data about rule violations (frequency of defects as collected by inspection data): none 2001 yet
- Rule source: Quality Manager [IW]
- Rule authority: Quality Manager.
- Any information pertaining to the rules in addition to this.

Requirements Rules
(an example of a powerful rule)

Source Priority Management manuscript page 8, ideals are on page 6 of Priority management

1. Quantify all things (qualities and costs) that vary ('increased', 'better')

Rationale: to give engineering clarity as basis for control of the specification.

Example: Scale: Mean Time to Learn, Plan [Teenage User] 30 minutes

Version: March 20 2001, 11:40

References: Priority Management Chapter Part 1 page 68, Competitive Engineering (Chapter on Quantifying Quality), Principles of Software Management. (especially Templates, Chapter 19)

Reference copy Generic Rules
(THESE ALSO APPLY TO REQUIREMENTS)

- **General Rules**
  Version June 22nd 2000 (apply to any plan) Owner: <process responsible>
- **G1: Reference Name:**
  - **Unique reference tag Capitalized for each elementary 'specification.**
- **G2: Clarity**
  - **Specs should be clear enough to measure or test, and clear to the intended readership.**
  - **Readership: shall be defined for each document.**
- **G3: Unambiguous**
  - **Specifications should be immediately unambiguous, as intended by the spec author, to the intended readership.**
- **G4: Source references**
  - **Each individual specification shall explicitly and in detail give the source (person or paragraph) of the spec.**
  - ***Rationale: {quality control, priority, acceptance, consensus}***
- **G5: Rationale (justification, impact)**
  - **Each spec of set of specs shall have a statement which directly explains what we are expecting as a result of doing it.**
- **G6: Single Instance**
  - **Specification shall have only one valid 'master' instance, to which all other uses will refer.**
  - ***Rationale: avoid confusion and multiple variations, automatic update, recognizability.***
- **G7: Fuzzy indication**
  - **When we are conscious that a term or terms need further clarification or definition we will explicitly inform the reader, usually using fuzzy brackets.**
- **G8: Assumptions:**
  - **All underlying assumptions shall be brought out and explicitly stated.**
  - **Rationale: risk analysis and testing of the truth of such assumptions.**
- **G9: Use The Planning Language**
  - **The FM Version of The Planning Language (Planguage) will be the guide to style, consistency and definition of terms.**
  - **Interim guide is Gilb's: Competitive Engineering, at www.result-planning.com.**

Example of Functional Specification

DATAB:
Gist: Deep Database Diagnostics.
Type: Functional Requirement.
Version: 25 Feb 2001: 15:43
Owner: Stakeholder : Quality Assurance Division
Linked To: ACC.D.MOP
Sub-functions: none
Specification:
Deep database diagnostics. <Various levels of checking> Not including <on mission>.

Assumptions: ***A0: it is cheaper to automate this function than to do analysis manually, and it is faster and more reliable.***
A1: the sub-system will be able to run in the background and monitor database quality.
A2: it will be able to be run user-parameter driven to sample particular classes of database records, data elements and relationships.
A3: it can be used integrated with the automatic recovery system.
Risks: Failure to update this function in parallel with the database structure.
Impacts: {System Recovery, Bug Maintenance, Database Integrity}
Priority:
This function must be available to some degree in first customer use releases. It will also be used in pre-release systems testing to some undefined degree.
Dependencies: the database system itself must be defined and operational.
Test:
This function shall be used in system testing and an early version of it can and should be made available in parallel with the development of the database itself. The function shall be tested by insertion of artificial database defects, and shall discover 100% of these.
Costs:
the cost of developing and maintaining this function is assumed to be between 10% and 50% of the cost of building and maintaining the database software in total.
Implementor: The Database Team
Function Intranet Location: ACC. Software.DB-Diagnosis

Part 2. Design Management {Means, Strategies, Tactics, Techniques}

- **Based on:**
  - **satisfaction of all quality requirements, resource and other constraints.**
- **Integrated with**
  - **the Evolutionary project management cycles, learning and adjust estimates.**
- **Tables relate**
  - **all designs to all requirements**
- **Tables and quantification**
  - **enable QC**
- **Design assertions**
  - **based on evidence, sources**

"Specific": Specification Rules

- **Rule set Identification: Data Sheet <intranet tbd>**
- **Rules owner: IW@Chip Company.com**
- **Version: 2 April 2001**
- **Scope: Data Sheets, Engineering Inspections, Engineers who write Data Sheets.**
- **Rule Prefix: Chip Company.Rules.Data Sheet.<#>**
- **1.** User View**: the Data Sheet is limited to describing things the User (Glossary) can see and test.**
  - **Examples: Types to include: Functional Description, Signal Description, Electrical Characteristics, Address Map Description, Register Description, Connection of Production Test Signals.**
    - **NOT to include: Implementation Details like Internal Structure and Signals, Detailed Production Test information.**
  - **Severity: Major. More Detail: <????, models, templates, courseware>**
- **2.** Value Added**: every statement should have some clear value added for the User.**
  - **Examples: a necessary fact, not presented otherwise at all. An example to help Users understand better. A view showing relationships; graphical diagrams to aid visualization of timing and operation.**
    - **NOT VALUE ADDED: <obvious> things, <redundant> things.**
  - **Severity: minor. More Detail: <????>**
- **3.** Cost Effective**: Everything specified should be implementable in a Cost Effective (see Glossary!) manner. Severity: Major.**
- **4.** Stakeholder Loyalty**: The Data Sheets will be Correct and Complete interpretations of requirements all classes of Customer-level Stakeholders. No essential-to-customer aspect shall be forgotten or corrupted. The Readership shall be able to understand exactly which of their previously specified requirements are implemented by our direct cross-reference to their requirements.**
  - **Example: Source: Customer Requirements Version 2-April-2001 page 16 5.1.6 Timing.**
- **5.** Standards Extent**: all telecommunications and industry standards used, will be referenced precisely, and a precise list of the elements of those standards which we propose to apply, and a precise list of those standards elements which we propose to NOT apply will be specified. The designer, the rationale and the sources shall be given.**
  - **Example: IEEE Std 498, Apply Pgf 1, 2, 23, NOT pgf 3, 5, 50. <-Engineer Tomberg**
    - **Rationale: we must apply some of these requirements because our customer demands it. <- Requirements 3.4.5**

Blank "Design specification" template. Version March 23 2001

Strategy Tag (official name):

- **Version:**
- **Owner:**
- **Gist:**
- **Type: <strategy & design>**
- **Stakeholders:**
- **Specification (definition):**
- **Real Expected Impact:**
- **Primary objective,**
- **Other objectives,**
- **Costs}**
- **± Uncertainty of Impact Estimate: ±___%**
- **Impact % on Specific Goal**
- **Primary objective: ___%**
- **Other objectives: ____ % on Objective _____**
- **Costs:**
- **Evidence:**
- **Source (of evidence):**
- **Credibility 0.0 low to 1.0 high**
- **Risks:**
- **Assumptions:**
- **References:**
- **Competitive Efforts:**
- **Market Targets:**
- **Alternative Strategies:**
- **Web Location of master specification:**

Specification Rules for 'Designs'

Version 22 June 2000/April 15 2001TG, owner <process responsible>

Design/Strategies/Initiatives: Defined As: means to impact the Objectives.

S1 (Use General Rules) - next slide

General Rules, Version June 22th 2000 (apply to any plan) Owner: <?>

S2: Template: Use the suggested template. "Design Specification Template"(previous slide) .

S3.: Model: see best practice model for other insights: "#2 Initiative June 22"

S4: Spec: The specification must be detailed enough and clear enough to understand the impacts of the design in terms of value delivered and costs.

S5. Real Impacts: The impacts are initially estimated on the scale of measure defined for a particular objective. So you need to specify the expected change from a defined baseline for the implementation of the design.

S6: (% Impacts) Impacts can also be expressed in terms of % progress on the real scale from the current level (0%, usually a Benchmark such as Past level), to the target level (usually a Plan level, 100% if on time).

S7 (Costs). All relevant cost aspects should be estimated as well as possible.

S8 (Risks) All potential risks which can *negatively influence* the estimated impact need to be stated. This is to permit pro-active planning to contain those risks.

S9 (Assumptions). Any assumptions which the 'impact, and timing-of-impact' rests on, need to be specified; again to that we can actively make sure these assumptions hold.

S10 (Credibility): the credibility of estimates basis shall be made on scale of 0.0 (none) to 1.0 (Perfect). (scale is in Gilb Competitive Engineering)

Company Glossary: what and why?

- **Initial Purpose: to give a standard for exact interpretation of terms used in standards, especially Rules.**
- **Terms which do not have obvious correct and complete interpretations.**
- **Terms which we expect to reuse several times, need to be defined once very well to avoid repetition and multiple updates**
- **Defined terms are Capitalized as a simple signal to the reader that there is a formal definition they should be aware of and use.**

Glossary Entry Template:

Version 1.0 April2001

- **<term main reference name>**
- **Definition: <write an unambiguous clear definition here.>**
- **---------- useful extra specifications, these are an option, until rules demand them----------**
- **Owner: <which person or function can update this glossary?>**
- **Links: <hyperlinks, web links TO RELATED TERMS>**
- **Index: <terms to index>**
- **Synonyms: <list exact equivalent terms>**
- **Antonyms:<list opposite meaning terms>**
- **Related Terms: <list closely related terms>**
- **Non-English Terms:**
- **Concept Number (*nnn):**
- **Graphical Icon:**
- **Version number:**
- **Date of last update:**
- **Source: <where could we look up more extended information about the term?)**
- **Authority: <what expert or authority is there for this term, like IEEE standard>**
- **Standard: <reference to specific standards like ISI, IEE, IEEE etc.>**

Glossary Rules for Competitive Engineering book manuscript

GLOSSARY Rules (MASTER)

Version July 9 2000 ←TG (A specific set of Rules for the Glossary) Note April 2001 these are dated in relation to the real CE manuscript.

BASIS: the Rules for Editing and writing the entire Book manuscript "Generic CE Rules MASTER" apply to the Glossary.

GC1. Terms are *numbered* to identify concepts independently of language used to identify them.

GC2. The number is preceded by * sign and may have descriptive words after a dot. e.g. *001.aim.

GC3. The *-numbers will all be found in the index.

This '*999' device is intended to help with consistency of using and understanding this method, in spite of translations, and special adaptations.

GC4. All definitions will be *proceeded* by the main term in bold type as well as its asterisk number in bold type. These are the Planguage symbols for a master definition of a term. This will be on a separate line, the definition (in regular type) immediately below. This definition should use the format:

Defined: <then the words defining the concept in regular typeface>.

GC5. All terms which are defined in this Glossary, formally with an asterisk number, *may* be written with Capital Letters, anywhere in the book text, to emphasize that they have a formal definition elsewhere. *Note these rules are designed to work with both* regular *and italic text.*

GC6. The definition *itself* will be in regular type, on lines below the bold term tag, preceded by the keyword "Defined:". Except emphasized words which can be in *italics*, and optional bold type or Capitalized words (those which are defined in this text). See GC9.

GC7. *Commentary,* which is not part of the essential definition, will be written in *italic type.* The main body of this commentary will start on a separate paragraph, indented.

At least one full line of space will be given between the term-definition and the commentary.

GC8. Emphasis, in a normal type sentence will be by means of *italics*.

GC9. An alternative way of pointing out that a term is defined in Planguage is to note the number in parenthesis immediately after the term. E.g. "procedure (*115)".

GC10. All formal Planguage 'Parameters' (examples {Scale, Defined, After}) will always have the first letter of all words in the term Capitalized. They may also, for emphasis, CAPITALIZE the ENTIRE TERM.

GC11. Synonyms, abbreviations and the like will be indexed using the format:

***number.<term> (<type of term>). The index shall apply to the *number, and may also be done for the term.**

GC12: Grammar class: whenever a term can have more than one grammar class, the one defined shall be stated explicitly: Example: Delivery (noun) *xxx

Examples of Terms defined in a real company situation in connection with Rules definitions

April 2001

City between 2 lakes

Generic Rules: Engineer sub-set:
THIS IS A SAMPLE OF THE RULES THAT NEEDED DEFINED TERMS THAT FOLLOW

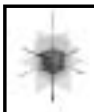
- Rule set Identification: Specifications <intranet tbd>
- Rules owner: IW
- Version: 4th April 2001
- Scope: All technical written specification.
- Rule Prefix: S-CO.Rules.Generic.<#>
- 1. Clear: All specifications must be clear enough to test; to formulate a specific test for proving correct delivery. It is about 'precision'.
  - Interpretation note: the investment in clarity must have a clear and probable 'payback' in terms of value for stakeholders, or economic and time saving in the long term. In other words do not overdo this unnecessarily.
- 2. Unambiguous: All specification must be Unambiguous to the Intended Readership. There is one and only one valid interpretation the 'correct' one as intended by the engineer. It is about 'confusion'.
  - Test [for Ambiguity] : if two or more people of the Intended Readership were to write their interpretations independently of each other, the interpretations should be essentially identical in engineering substance and result (properties of quality, cost, function). Alternatively the author can judge any one interpretation written or oral (for example at an Inspection meeting) as correct according to their intent.
  - *BACKGROUND COMMENT: THESE WERE APPROVED BY VOTE 100% BY CLASS PARTICIPANTS*

The consequence of a few rules...

- Use the page Datasheet Circuit:LED 24 page 8/20 version 7.9.2000 version 1.1.
- Using 10 minutes
  - Mark on the sheet all potential violations (issues -> defects) of the **Unambiguous/Clear rules** , classify as probably (M) ajor or (m)inor
  - 10 ¤¤¤ cash tax free to finder of most Majors!
  - **Reports Majors:10, 20, 15, 20, 12, 13, 12, 7, 4, 7, 12, 11, 11, 16, 7, 14, 6, 12, 8, 11, 19, 7, 8, 3**
  - Estimate the number of Majors on the entire page,
  - Highest score 20
  - Team score probably about ~ 2(3?) (for small teams of 2 -5 people) x 20 =40 for 22 people I alter the estimate to 3x 20 =60±15, also we used only 15 minutes, Optimum effectiveness (maximum find) needs about 1 hour: so add at least 50% 60 +50% ~= 90.
  - Now this kind of SQC (Spec Quality Control) is 30% to 90% effective , the highest % take 5 to 8 years of culture improvement. The lower number is more realistic and conservative. 33.3% 3x 90= 270 Majors ±100
  - And this is only for TWO RULES: if 20 rules maybe we have 1800 Majors/page??
  - If we fixed all 90 we found, we woud still have 180 (not found yet) +18 not fixed correctly = ~200 Majors per page.
  - If about 1/3 of these caused delay or fault (~67) and the delay was about 10 hours then this page, after correcting would delay you project by 10 x 67 = 670 engineering hours,
  - We have a 20 page data sheet so the total project delay projected is about 20 x 670 hours =
  - ~ 14,000 (@ 2,000 hours year) 7 work years lost as a result of this sloppy work.
  - EXPERIENCE: at GE, IBM, Ericsson we have proven this calculation works
  - ASSUMPTION: these are really Majors: the defects can really cause delay!
    - In this case we suspect the documentation is ignored (with good reason!).
  - CASE STUDY: RON RADICE (IBM) PAPER FOR LUIS IN THE CD
  - TECHNICAL NOTE: BELLCORE (HON PENCE 93) 42% BUGS DUE SPECS DEFECT, TRW (78) 62% BUG IN MILITARY SPACE SOFTWARE DUE TO SPECIFICATION DEFECTS GIVEN TO CODERS.!

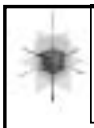
Company Glossary (real example) " User, Cost-effective, Intended Readership"

- **User:**
  - **Defined As: Any person at the customer (not Synopsys) end who will use the data sheet for any purpose whatsoever.**
  - **Known set of Users [Data Sheet] includes:**
    - **HW Engineers, SW Engineers, System Engineers, Engineering Management.**
- **Cost Effective:**
  - **A cost-effective design is one where the value to customer versus the costs of implementation, and production are:**
  - **A. acceptable to the Customer**
  - **B. Competitive with regard to our competition**
  - **C. The lowest cost design alternative we can offer, with satisfaction of any other related objectives and constraints, such as performance, reliability etc.**
  - **Reference to further information:**
    - **<specify how to calculate value and costs in practice!!!! TG>, Template for evaluation of any design strategy.**
    - **See Gilb: Principles of Software Engineering Management Chapter 11 Solution Evaluation,**
  - **Version April 4 2001**
- **Intended Readership:**
  - **The total set of all potential readers of the specification which we would want to understand the specification correctly. This set should be listed and agreed for each type of document and available either in the document heading or in an intranet location specified.**



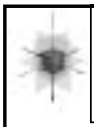
Glossary examples: continued (Unambiguous..)

- Unambiguous:
  - A specification has one one possible interpretation by the Intended Readership: the exact one intended by the document author.
  - Test [Ambiguity] : if two or more people of the Intended Readership were to write their interpretations independently of each other, the interpretations should be essentially identical in engineering substance and result (properties of quality, cost, function). Alternatively the author can judge any one interpretation written or oral (for example at an Inspection meeting) as correct according to their intent.



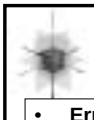
Glossary Example Continued : "Correct Complete, Customer-Level Stakeholders"

- **Correct:**
  - **Defined As: A specification is correct if it is in perfect agreement with correct and official Sources.**
  - **Example: a design specification is correct when it is addressing the official cross-reference-by-the-design requirements as well as it claims to do (Impact analysis: values and costs). A formal technical requirement specification is correct when it is completely consistent with the customer wishes; and would be acknowledged by the same customer to be a correct interpretation of their initial requirements or wishes.**
- **Complete:**
  - **Defined As: A specification is complete when it addresses and interprets all aspects of the sources it claims it is using (by direct source reference to them). Not only when it addresses the referenced source specifications; but it must laso satify the condition that it completely addresses the source specifications which is logically should be using as engineering process inputs, according to either formal process definitions (Rules in particular) or common sense and real life observation.**
  - **Examples:**
    - **If a specification document is found to be missing any one specification it should have, according to the Rules for that type of document or specification, then that missing specification in the engineering work product is a defect.**
    - **If the Rule is 'All Quality Requirement from the customer must be expressed quantitatively" and the work Product specification is still"Bad Data Robust' then the specification is Not complete.**



Glossary Examples Continued: "Customer-level Stakeholders".

- Customer-level Stakeholders.
  - Definition: Project or product stakeholders with critical or profitable requirements, and which are at the level of customer, user, rather than Synopsys internal/ our suppliers.
  - Example [Customer-Level] {Customer Designers, Customer Managers, Manufacturers, ??}



Glossary examples for 'bad' stuff :)

- **Error:**
  - **act committed by a human which is wrong by some defined standard of action (a rule or process for example)**
- **Defect:**
  - **a written specification which is wrong according to some defined standard (a rule).**
- **Major:**
  - **a defect severity classification which implies potentially non-trivial costs downstream as a result of this defect type. Avg. Major cost ~10 engineering hours, if 'triggered (~25%to 35% probability)**
- **minor:**
  - **a defect severity classification meaning not major.**
- **Issue:**
  - **a specification identified as potentially being a defect, but we are not yet certain.**
- **Fault:**
  - **A potential malfunction in a system or product.**
- **Bug:**
  - **Informal synonym for 'fault'. It is also used to describe occurrence of a 'malfunction'.**
- **Malfunction:**
  - **A real system (even a prototype) fails to act according to some definition of how it should act (example requirements or design or both)**

Impact Estimation Principle: Multiple Purposes

SEQUENCE
by risk or value

EVALUATE
all critical attributes

PRESENT
facts & basis

PRIORITIZE
by overall objective value to cost

COMPARE
alternatives

Attribute Tag	Point	100% PLAN or MUST	IDEA-1	IDEA-2	Risk Sum (10.)	SUM (9.)	Safety Factor 2 Deviation (11.)
RELIABILITY	300 Hrs.	3000 h	50%±0	20%±80	±80	70%	-130%
USABILITY	20 min.	10 min.	10%±40	60%±90	±130	70%	-130%
Sum Qualities		(12.->)	60	80			
CAPITAL	0	1 mill	50%±20	10%±20	±40	60%	-10%
MAINTENANCE	1 mill/year	100%00 per yr.	50±20	100%±80	±100	100%	-50%
Sum Costs		(13.->)	50	110			
Quality/cost ratio		(14.->)	1.2 (60/50)	0.73 (80/110)			

FIND WEAKNESS
in useful quality or excessive costs

ARGUE
for or against alternatives

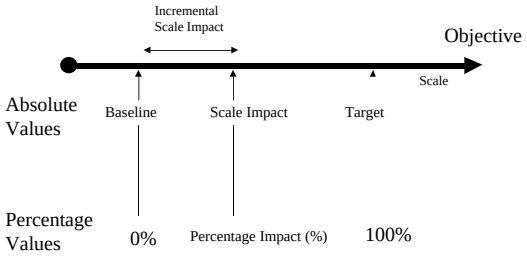
DESTROY
false beliefs with facts

CUT FAT
by understanding value to cost ratio

SEE RISK
by documenting uncertainty

Gilb@acm.org

Impact Estimation: Basic Concepts



Source: Lindsey Brodie, Editor of Competitive Engineering May 2000

Gilb@acm.org

"Impact Estimation" concepts: full table

All in %!

Tags of proposed TOTAL SET of strategies (defined elsewhere) for meeting the quality objectives, within resource constraints.

Strategies-> Objectives	***A1***	**B4**	**CD**	**DX**	Sum
AVAILABILITY "99.9%->99.98%" "PAST->PLAN"	0%	100%	50%	-5%	145%
PORTABILITY "80%->95%"	1	1	1	1	4%
USABILITY " 3 mins.->1"	**60±20 %**	**99**	**41**	**200**	400%
BUDGET **"0->1 million"**	100	10%	?	-	110?
EMPLOYEES **"0->32 people"**	0	30%		9±5	
Benefit/Cost->	**0.6**	**5.0**			

Gnb@acm.org

"Impact Estimation" concepts: detail

Tags of proposed TOTAL SET of strategies (defined elsewhere) for meeting the quality objectives, within resource constraints.

***Rough* sum of effects of all strategies on a single attribute's planned level.**

Quality and Benefit Objectives

Resource Budget tags

Strategies-> Objectives	**A1**	B4	CD	DX	**Sum**
AVAILABILITY "99.9%->99.98%" "PAST->PLAN"	0%	100%	50%	-5%	**145%**
PORTABILITY "80%->95%"	1	1	1	1	**4%**
USABILITY " 3 mins.->1"	60 ±20	99	41	200	**400%**
BUDGET "0->1 million"	00	10%	?	n.a.	**110?**
EMPLOYEES "0->32 people"	0	30%		9±5	
Benefit/Cost->	**0.6**	**5.0**			

Clearly not good enough design yet

Safety margin 4X

Explicit uncertainty estimate

Estimation language:
0% = no effect with respect to PAST level.
100% = expected to meet PLAN level.
negative effect= makes things worse than PAST level.
? = no basis for an estimate.
n.a. = not applicable.

Sum Benefits / Sum resources = rough relative goodness of a strategy with respect to all objectives.

Evidence
USABILITY:A1

Design method A1 in all competitive products and in our lab prototypes shows user learning time to be under two minutes. <- Lab Report U-92

Objective statement, example

Strategy Definition Example

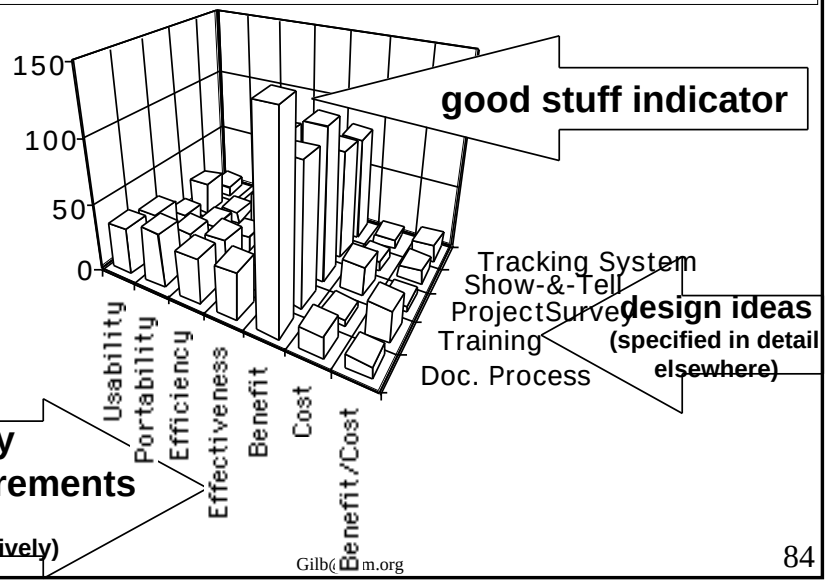
A1: Graphical interfaces using minimal language, no codes, maximum pictures, maximum user tailoring, maximum learning about particular users.

b@acm

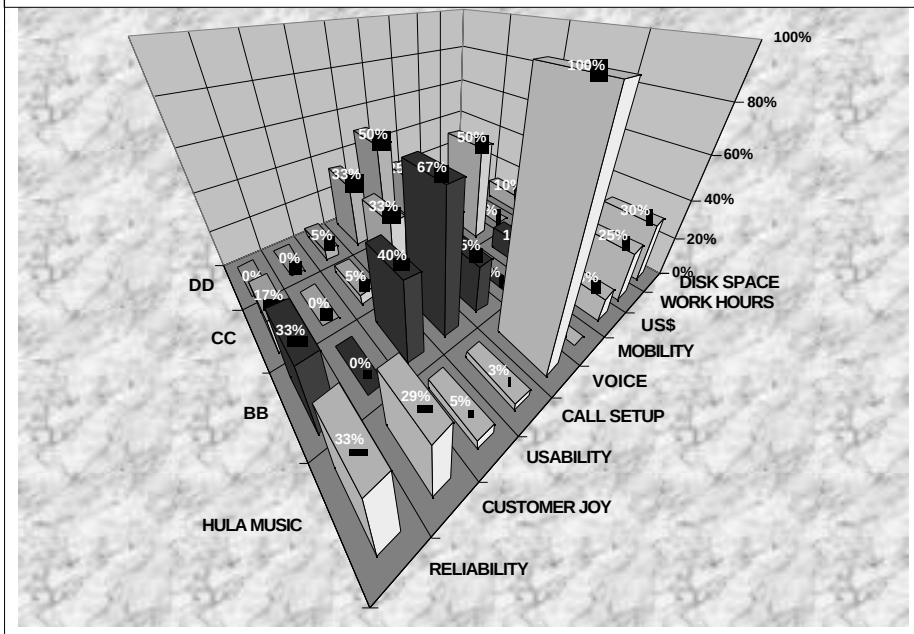
Credibility Rating Scale

Credibility Rating	Meaning
0.0	Wild guess, no credibility
0.1	We know it has been done somewhere
0.2	We have one measurement somewhere
0.3	There are several measurements in the estimated range
0.4	The measurements are relevant to our case
0.5	The method of measurement is considered reliable
0.6	We have used the method in-house
0.7	We have reliable measurements in-house
0.8	Reliable in-house measurements correlate to independent external measurements
0.9	We have used the idea on this project and measured it
1.0	Perfect credibility, we have rock solid, contract-guaranteed, long-term, credible experience on this project and, the results with this idea are unlikely to disappear

Impact Table 3D Display:
How good are design ideas compared to your objectives?



Skyscraper Impact Estimation Format



Gilb@acm.org

Impact Estimation Example

Design Ideas➔ Objectives +.-|-|-.#	**IDEA-1** Impact Estimates ➔.#*	**IDEA-2** Impact Estimates ➔.#*	Sum **(3)** Percentage Impact =.+.%.#	Sum **(4)** Percentage Uncertainty =.+.%.±	Safety Factor ±.*.2 Deviation**(5)**
RELIABILITY **300 -> 3000 hours MTBF** +.%.#	1650hr ±0 **(1)** 61%±0 **(2)**	840hr ±240 31%±9%	92%	±9%	-108%
Incr. Scale impact: +.-|-|-.# **USABILITY** **20 -> 10 minutes** Incremental % est.: +.%.# Sum Qualities **(6)** =.O➔*.%	1min. ±4 10%±40% 71%	6 min. ±9 60%±90% 91%	70%	±130%	-130%
+➔O-|-|-# +.➔O.-|-|-.#.± **CAPITAL** **0 -> 1 million US$**	500K ±200K 50%±20	100K ±200K 10%±20	60%	±40%	-10%
MAINTENANCE **1.1M -> 100K/year US$** +.➔O.#.% & ± Sum Costs **(7)** =.➔O*.%	0 K$/Y ±180K 0%± 18% 50%	1 M$/Y ±720K 100%±72 % 110%	100%	±90%	-50%
Quality-To-Cost Ratio **(8)** **O➔* / ➔O***	1.42 (71/50)	0.83 (91/110)	Idea-1 = 71% better		

An Impact Estimation for 'Learning'

TASK-HELP: Gist: the set of ideas below.
ONLINE-SUPPORT: Gist: provide an optional alternative user interface, with the user-task information for defined task(s) embedded into it.
ONLINE-HELP: Gist: integrate the user-task information for defined task(s) into the user interface as a 'Help' facility.
PICTURE-HANDBOOK: Gist: produce a radically changed handbook that uses pictures and concrete examples to *instruct*, without the need for *any* other text.
ACCESS-INDEX: Gist: detailed *keyword indexes* will be made, using *experience* of *at least ten* real users learning to carry out the defined task(s). What do *they* want to look things up under?

Ill. 8.1.1 More-detailed design specification

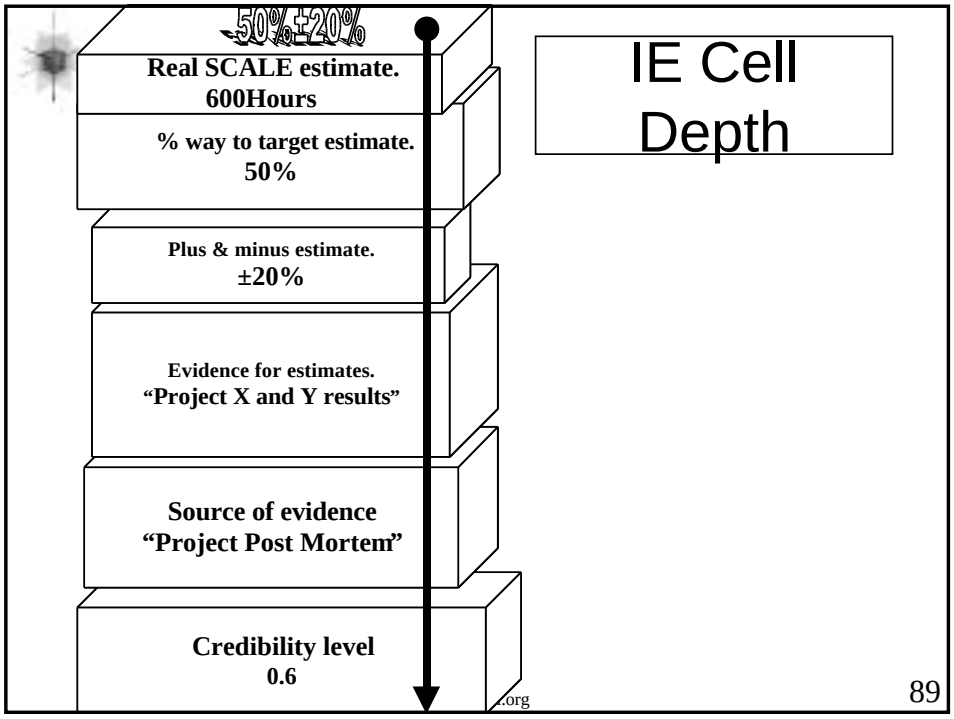
'Task Help' Design Ideas-> Objective	**ON-LINE SUPPORT**	**ON-LINE HELP**	**PICTURE HANDBOOK**	**ON-LINE HELP + ACCESS INDEX**
LEARNING Past[60min]->Plan[10min] Scale Impact	5 min.	10 min.	30 min.	8 min.
Scale Uncertainty	±3min.	±5 min.	±10min.	±5 min.
Percentage Impact	110%	100%	67% (2/3)	104%
Percentage Uncertainty	±6%	±10%	±20%?	±10%
Evidence	Project Ajax, 1996	Other Systems	Guess	Other Systems + Guess
Source	Ajax report, p.6	World Report p.17	John B.	World Report p.17 + John B.
Credibility	0.7	0.8	0.2	0.6
Development Cost	120K	25K	10K	26K
Quality- To- Cost Ratio	110/120 = 0.92	100/25 = 4.0	67/10 = 6.7	104/26 = 4.0
Notes: Time Period is two years.	Longer time-scale to develop			

87

Impact Estimation Purposes (lots of purposes!)

1. Evaluating a *single* design idea. *How good is the idea for us?*
2. Comparing two or more design ideas to find a *winner*, or set of winners. *Use IE, if you want to set up an argument against a prevailing popular, but weak design idea!*
3. Gaining an *architectural overview* of the impact of *all* the design ideas on *all* the objectives. *Are there any negative side effects ?*
4. Obtaining *systems engineering views* of particular components, or particular quality aspects. *Are we going to achieve the reliability levels ?*
5. *Analyzing risk*; evaluating the designs with regard to negative uncertainty and minimum credibility.
6. Planning *evolutionary project steps* with regard to value and cost.
7. Monitoring for *project management accounting purposes*, the progress of single evolutionary delivery steps, *and the progress to date* compared against the requirement specifications or management objectives.
8. *Predicting* future costs, project time-scales and quality levels.
9. Understanding *organizational responsibility* in terms of attributes and costs by organizational function (Steve Poppe's application).
   In 1992, Mr. Steve Poppe pioneered this use at executive level while at British Telecom North America.
10. Achieving rigorous quality control of a design, before approval.
11. *Presenting ideas* to committees, management boards, senior managers, review boards, and customers for approval.
12. Identifying which parts of the design are the *weakest link (risk analysis). If there are no obvious alternative design ideas, they should be tried out earliest, in case they do not work well (risk management). This impacts scheduling.*
13. Enabling *configuration management* of design, design changes, and change consequences.
14. Permitting *delegation of decision-making* to teams. *Teams can achieve better internal progress control using IE, than they can from repeatedly making progress reports to others, and acting on other's feedback. (See point 11. IE can be used to report progress to management, as opposed to seeking permission.)*
15. Presenting overviews of *very large, complex* projects and systems by using *hierarchical* IE tables. *Aim for a one page top-level IE view for senior management.*
16. Enabling cross-organizational co-operation by presenting overviews of how the design ideas of different projects contribute towards corporate objectives. *Any common and conflicting design ideas can be identified. This is important from a customer viewpoint ; different projects might well be delivering to the same customer interface.*
17. Controlling the design process. *You can see what you need, and see if your idea has it by using the IE Table. For example, which design idea contributes best to achieving usability? Which one costs too much?*
18. Strengthening design. *You can see where your design ideas are failing to impact sufficiently on the qualities; and this can provoke thought to discover new design ideas or modify existing ones.*
19. *Helping informal reasoning and discussion* of ideas by providing a framework model of how the *design* is *connected* to the *objectives* in our minds.
20. Strengthening the specified objectives. *Sometimes you can identify a design idea that has a great deal of popular support, but doesn't appear to impact your objectives. You should investigate the likely impacts of the design idea with a view to identifying additional stakeholder objectives, which may be the underlying reason for the popular support. You might also identify additional types of stakeholders.*

44



Impact Estimation Policy

Impact Estimation Policy

1. All designs or strategies which can have significant (5% or more) impact on any one critical quality or cost requirement of a project or product, must be included in an Impact Estimation table.

2. The design ideas must be detailed and clear enough to clearly support the estimates made, irrespective of who would make or evaluate the estimates.

3. An Impact Estimation table, with all related design and requirements specifications, shall be quality controlled with respect to all relevant Rules, and the Major defect level will be low enough to Exit, and the estimated Major defect level will in any case be stated on the cover page.

4. Significant proposed changes to the design or architecture shall be accompanied by an impact estimation showing the net impact of the change. The quality of specification and QC shall be as in the policy above.

Gilb@acm.org

US Army IE Table Persinscom

STRATEGIES → OBJECTIVES	Technology Investment	Business Practices	People	Empow-erment	Principles of IMA Management	Business Process Re-engineering	SUM
Customer Service ?→0 Violation of agreement	50%	10%	5%	5%	5%	60%	185%
Availability 90% → 99.5% Up time	50%	5%	5-10%	0	0	200%	265%
Usability 200 → 60 Requests by Users	50%	5-10%	5-10%	50%	0	10%	130%
Responsiveness 70% → ECP's on time	50%	10%	90%	25%	5%	50%	180%
Productivity 3:1 Return on Investment	45%	60%	10%	35%	100%	53%	303%
Morale 72 → 60 per mo. Sick Leave	50%	5%	75%	45%	15%	61%	251%
Data Integrity 88% → 97% Data Error %	42%	10%	25%	5%	70%	25%	177%
Technology Adaptability 75% Adapt Technology	5%	30%	5%	60%	0	60%	160%
Requirement Adaptability ? → 2.6% Adapt to Change	80%	20%	60%	75%	20%	5%	260%
Resource Adaptability 2.1M → ? Resource Change	10%	80%	5%	50%	50%	75%	270%
Cost Reduction FADS → 30% Total Funding	50%	40%	10%	40%	50%	50%	240%
SUM IMPACT FOR EACH SOLUTION	***482%***	***280%***	***305%***	***390%***	***315%***	***649%***	
Money % of total budget	15%	4%	3%	4%	6%	4%	
Time % total work months/year	15%	15%	20%	10%	20%	18%	
SUM RESOURCES	***30***	***19***	***23***	***14***	***26***	***22***	
BENEFIT/RESOURCES RATIO	**16:1**	**14:7**	**13:3**	**27:9**	**12:1**	**29:5**	

91

Sample Objective/Strategy Persinscom

Customer Service:
Ambition: Improve customer perception of quality of service provided.
Scale: Violations of Customer Agreement per Month.
Meter: Log of Violations.
Past [1991] Unknown Number ←State of PERSCOM Management Review
Record [NARDAC] 0 ? ← NARDAC Reports 1991
Must : <better than Past, Unknown number> ←CG
Plan [1991, PERSINCOM] 0 "Go for the Record" ← Group SWAG

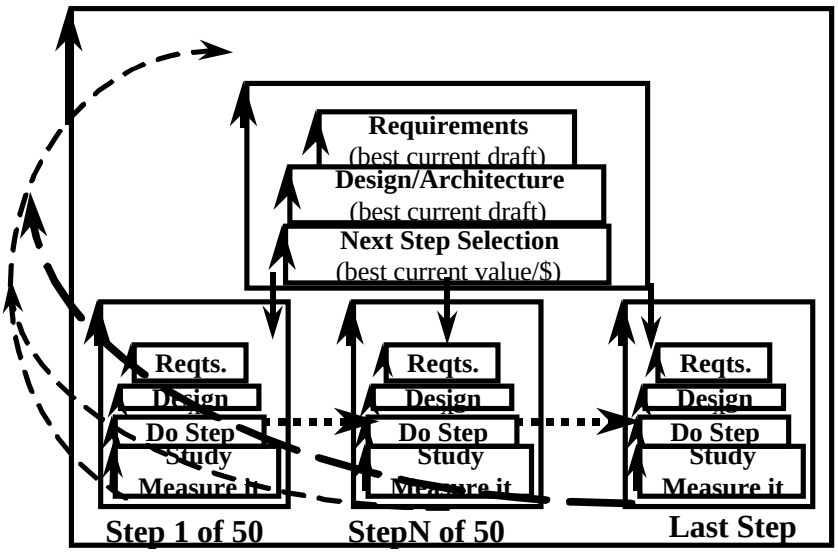
Technology Investment:
Exploit investment in high return technology. Impacts: productivity, customer service and conserves resources.

92

Design Comparison.
Can you compare apples and oranges?

Requirements	Apple	Pear	Strawberry	Carrot
Vitamins	Vitamins 30%	Vitamins 80%	Vitamins 65%	Vitamins 57%
Poison Sprays	Poison Sprays 60%	Poison Sprays 25%	Poison Sprays 85%	Poison Sprays 40%
Domestic Produce	Domestic Produce 95%	Domestic Produce 44%	Domestic Produce 15%	Domestic Produce 60%
Current Price	Current Price .35	Current Price .55	Current Price .67	Current Price .22
Wastage %	Wastage 10%	Wastage 20%	Wastage 40%	Wastage 60%
Quality to Cost Ratio	Quality to cost ratio 4.0	Quality to cost ratio 1.99	Quality to cost ratio 1.54	Quality to cost ratio 1.92

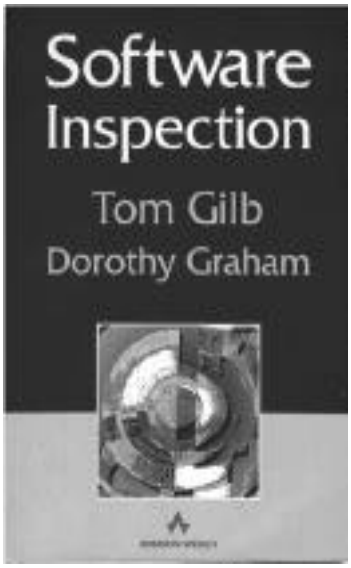
Design feedback adjustment integrated into the delivery cycle



Part 3. Specification QC

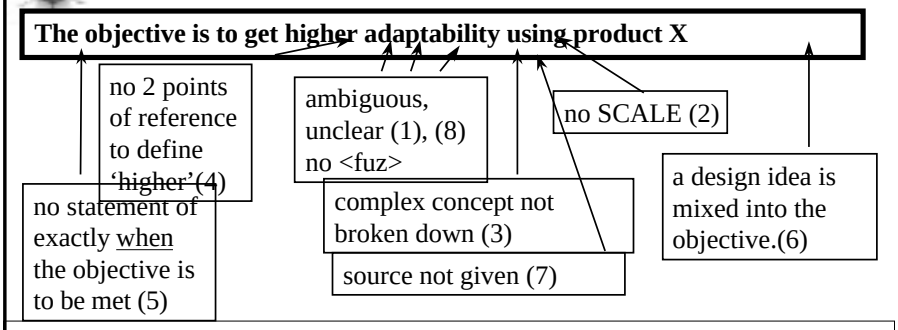
- **Based on Objective 'rules' of specification**
- **Focus on 'Major' downstream costs reduction**
- **An engineering process measuring method**
- **Basis for continuous improvement of engineering process (95% defect avoidance)**
- **Proven to reduce project time 50% (rework)**
- **Proven in Aircraft Engineering, Telecoms**
- **Superior to 'checking', 'reviews', 'approvals', 'meetings'**

A systems level Specification QC process based on Statistical Process Control methods (Deming)



- **Focus on**
  - **Measurement, by sampling, of a specification's conformance to 'best practice' standards (rules and exit levels)**
  - **Not about 'bug removal'**
  - **Predictor of bugs in test and in field**

Defects in a Statement



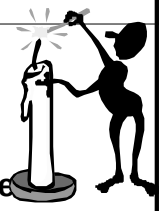
RULES FOR QUALITY OBJECTIVES: Tag: RULES.QOBJ
QOBJ.1. They should be unambiguously clear to the intended reader.
QOBJ.2. They shall specify a SCALE of measure to define the concept.
QOBJ.3. They shall break down complex concepts into a set of measurable concepts.
QOBJ.4. To define 'relative' terms like 'higher' they shall specify at least two points of reference on the defined SCALE.
QOBJ.5. They shall specify exactly when a quality level is to be available.
QOBJ.6. They shall not mix design ideas in the specification of objectives.
QOBJ.7. The process input (like contract, standard, marketing plan) of the requirement shall be given.
QOBJ.8. Fuzzy unclear concepts shall be marked with <angle brackets> for improvement.

'Editing' to follow the rules

(this might not be a 'good' plan but it contains no 'defects'

- **Adaptability:**
  - **Maintainability:**
    - **SCALE: Clock time to fix a bug and validate fix.**
    - **PAST [Product X, last year] 5 hours <- *Internal stats.***
    - **PLAN [Product Y, At Launch] 10 minutes <- *Mkt. Dir.***
  - **Portability: *<- Marketing Plan Dec 7th. M.P.***
    - **SCALE: Conversion cost for [defined ports].**
    - **PAST [Prod. X, Any UNIX, 1996] 100 hours/1000 Lines**
    - **PLAN [Prod. Y, Any UNIX, 2001] 20 hours/1000 Lines**

10 Top Advanced SQC/Inspection Principles



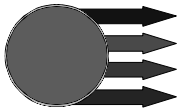
- **Pr1. *Prevention* is more effective than Cure**
- **Pr2. *Avoidance* is more efficient than removal**
- **Pr3. *Feedback* teaches effectively**
- **Pr4. *Measurement* gives facts to control the proce**
- **Pr5. Priority to the *Profitable***
- **Pr6. *Forget perfection*, you can't afford it!**
- **Pr7. Teach *fishing*, rather than 'give fish'**
- **Pr8. *Framework for Freedom* beats bureaucracy**
- **Pr9. *Reality* rules**
- **Pr10. *Facts* beat intuition**

Advanced SQC/Inspection Objectives

- **Central Objectives**
  - **1. Engineering Process Control**
  - **2. *Measuring* Specification Quality**
  - **3. Reduce Project Time & Cost**
- **Secondary Objectives**
  - **4. Identify and Remove Major Defects**
  - **5. Reduce Service/Maintenance Costs**
- ***NOT* Objectives**
  - **Approve document '*content*'**
  - **Remove *minor* defects**
  - **'*Improve*' Quality**

Main ‘Specs QC’/Inspection Objectives

1. Time-to-Delivery
2. Measurement
   - •document quality
   - •doc. process quality
   - •QC value/cost
3. Release “downstream”
4. Identify defects
5. Fix defects
   - avoid new defect injection
6. Improve process
   - product producers
   - QC process itself
7. On-the-job training
8. Motivation
9. Help Engineer
10. Effectiveness (Quality)
11. Efficiency (Productivity)
12. Train QC team leader
13. Certify the team leader
14. Motivate Managers
15. Reduce Maintenance Costs
16. Relieve Project Leader.
17. many others

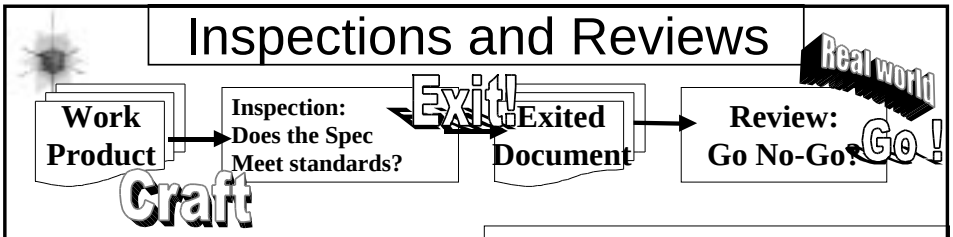


Difference to ‘Conventional’ Inspections

- **Conventional Inspection**
  (IBM, Fagan, 1973)
  - No sampling
  - Inflexible bureaucracy
  - Focus on ‘Cleanup’
  - Focus on ‘software code’
  - Poorly documented process
  - “My Way”<--MEF/IBM
  - Do the defined process!
  - ‘Interpret’ the document

- **‘Advanced’ Spec QC**
  (Gilb & Graham, Software Inspections)
  - Sampling to measure specs (much cheaper!)
  - ‘Intelligent Inspections’
  - Focus on Time & Control
  - Systems, upstream focus
  - Richly documented (Book)
  - ‘Our Way’ & ‘Your Way’
  - Do what pays off, *only*
  - Check against Rules, Sources

Inspections and Reviews

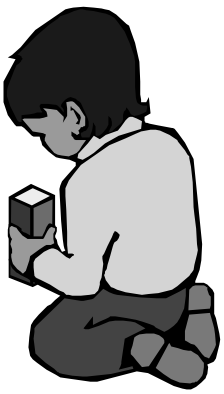


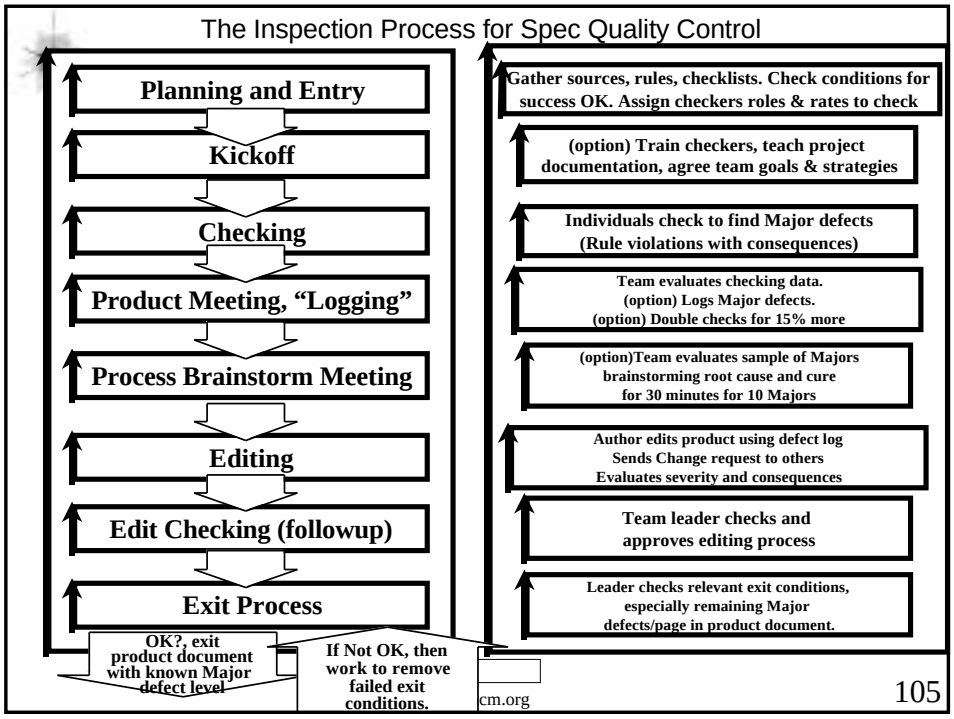
- Inspections
  - Judgement based on conformance to standards
  - Well written, clear, complete, trustworthy
  - Can be carried out by any of 'intended readership'
  - Should be done to guarantee decision-makers a good basis for a decision.

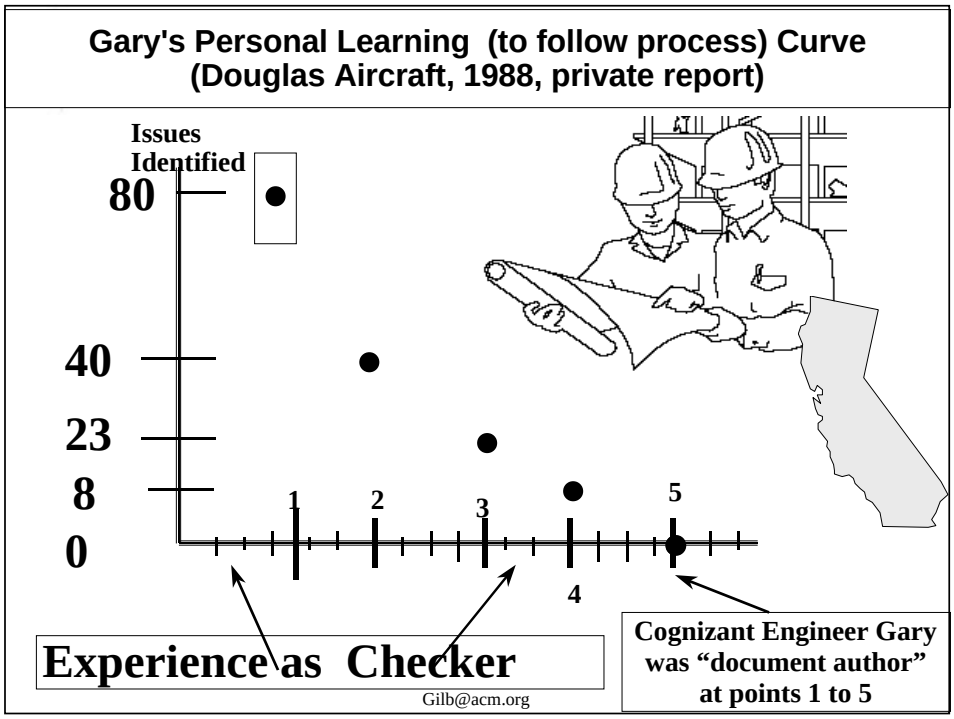
- (Go No-Go) Reviews
  - Judgement based on goodness in real world
  - Content, not format;
  - Value, not clarity
  - Approval by authorized 'managers'
  - Should not be Entered if document not Exited from Inspections

Paradigms

- **P1. Engineering process control**
- **P2. Cleanup is ineffective**
- **P3. Teamwork beats ego**
- **P4. Data beats guessing**
- **P5. Real Time Control**
- **P6. Author Responsibility**
- **P7. Checker Consultants**
- **P8. Author is Client**
- **P9. Optimize Checking speed**
- **P10. Quantified Gatekeepers**
- **P11. Rules Rule Objectively**
- **P12. Structure satisfies objectives**
- **P13. What works is right**



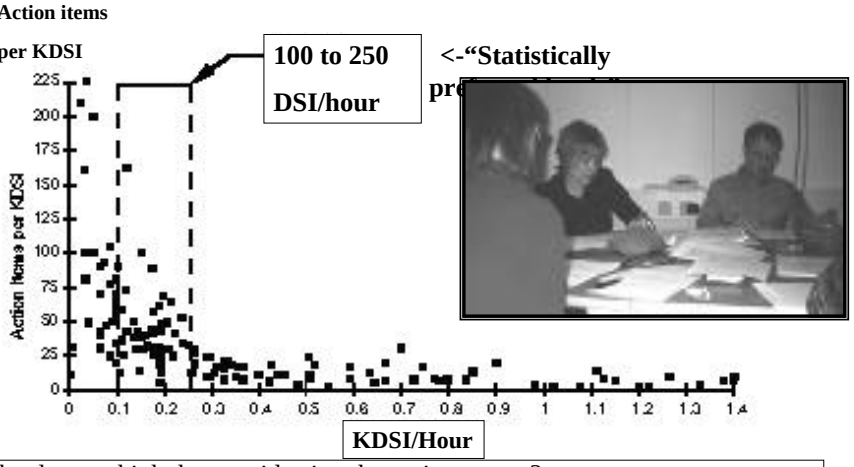

The Inspection Process for Spec Quality Control
Planning and Entry
Kickoff
Checking
Product Meeting, "Logging"
Process Brainstorm Meeting
Editing
Edit Checking (followup)
Exit Process
Gather sources, rules, checklists. Check conditions for success OK. Assign checkers roles & rates to check
(option) Train checkers, teach project documentation, agree team goals & strategies
Individuals check to find Major defects (Rule violations with consequences)
Team evaluates checking data.
(option) Logs Major defects.
(option) Double checks for 15% more
(option)Team evaluates sample of Majors brainstorming root cause and cure for 30 minutes for 10 Majors
Author edits product using defect log
Sends Change request to others
Evaluates severity and consequences
Team leader checks and approves editing process
Leader checks relevant exit conditions, especially remaining Major defects/page in product document.
OK?, exit product document with known Major defect level
If Not OK, then work to remove failed exit conditions.
cm.org
105


Gary's Personal Learning (to follow process) Curve
(Douglas Aircraft, 1988, private report)
Issues Identified
80
40
23
8
0
1
2
3
4
5
Experience as Checker
Gilb@acm.org
Cognizant Engineer Gary was "document author" at points 1 to 5

All vital documentation

- **Can be measured for adherence to standards**
- **engineering process can be indirectly measured**
- **powerful training device**
- **gives objective release (exit) and accept (entry) criteria for engineering process**

Fault Density versus Checking Rate: Raytheon 95<-- TR 017 1995 at SEI website SEI.cmu.edu

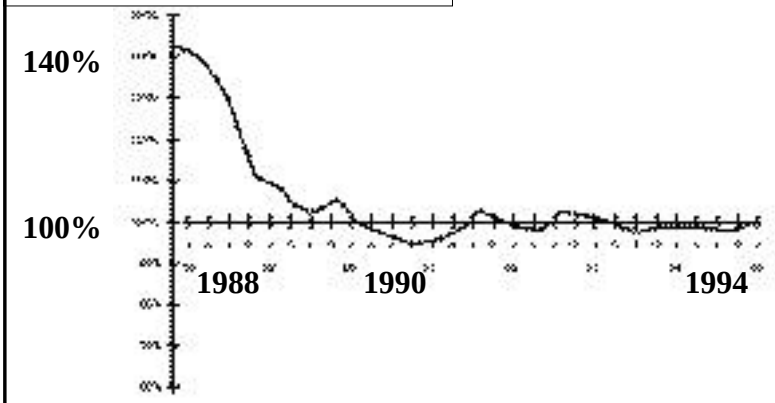


Why do you think they avoid using the optimum rate?

Hint: "Our process mandates 100% Inspections coverage"

Achieving Project Predictability: Raytheon 95

Cost At Completion / Budget %



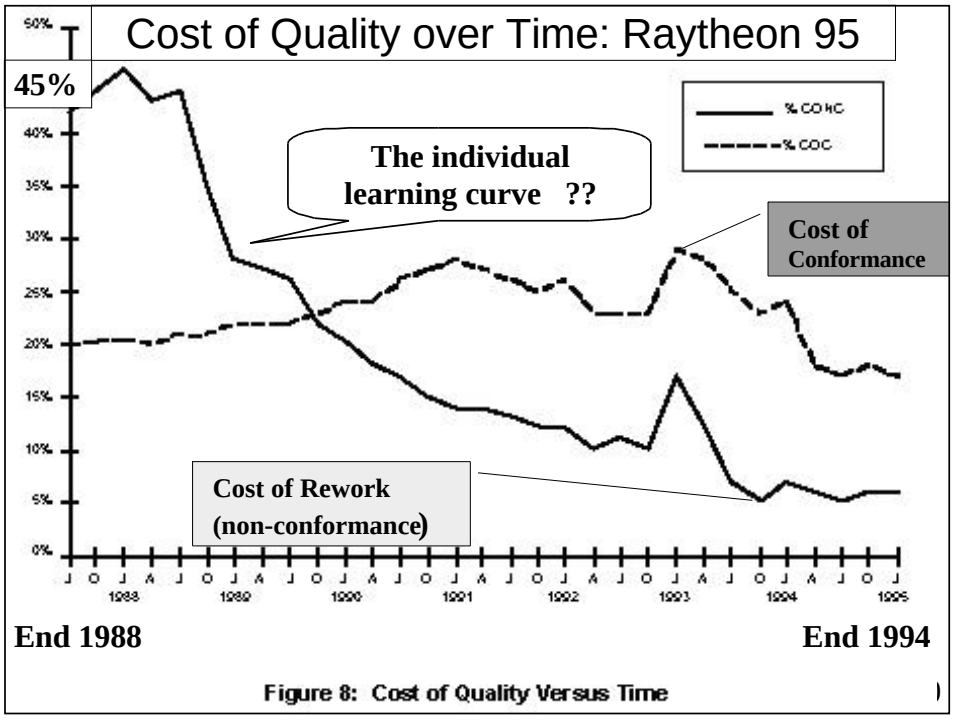
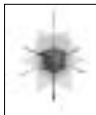


Figure 8: Cost of Quality Versus Time

A Sampling Case Study

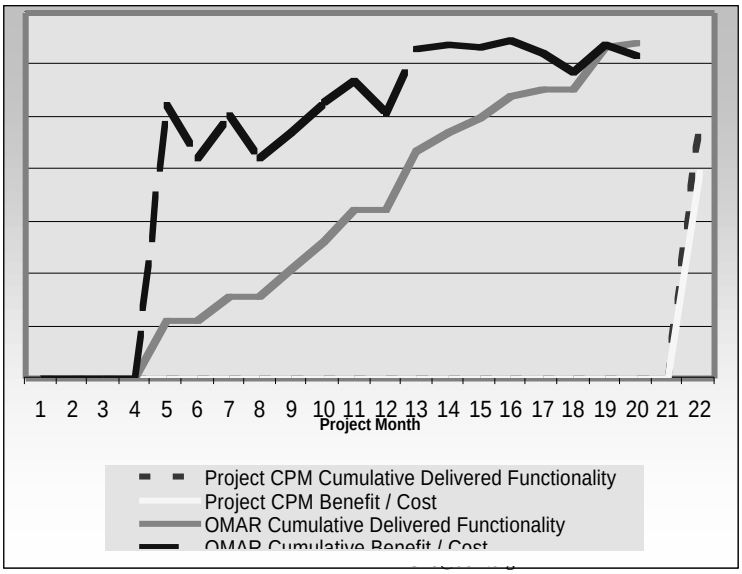
- **1986 Northern Europe**
  - **Air traffic control trainer system for export**
  - **80,000 pages contracted pseudocode before code**
  - **40,000 pages already written**
  - **Project seriously late already (customer informed)**
  - **About 7 management signatures approving the 40,000 pages (pseudocode for coders)**
  - **SQC of a sample of three pages**
    - **chosen by random numbers**
    - **declared to be representative**
    - **19 Major defects found in half day QC/Inspection by the 7 managers**
    - **director checks the defect log and confirms**



Part 4. Evolutionary Project Management

- **Proven industrially by IBM FSD, HP, JPL, Microsoft**
- **DoD moving towards it (Mil Std 498, Jt Std 016, IEEE stds)**
- **Based on feedback to requirements and design**
- **Based on 2% to 10% of normal delivery cycles**
- **Project Progress = *delivered* user *results***
- **Almost no literature or teaching, 'unknown' (Peter J Morris)**
  - **"The Management of Projects" (Telford, 1994 London)**
- **Major culture and project management 'shift'**
- **Design to cost, design to schedule**
- **Risk control, on time, under budget**
- **User-participative all the way**
- **See Alan MacCormack: How Internet Companies Build Software, Winter 2001, MIT Sloan Management Review, p75 on.**

The OMAR Project (UK) Value Early



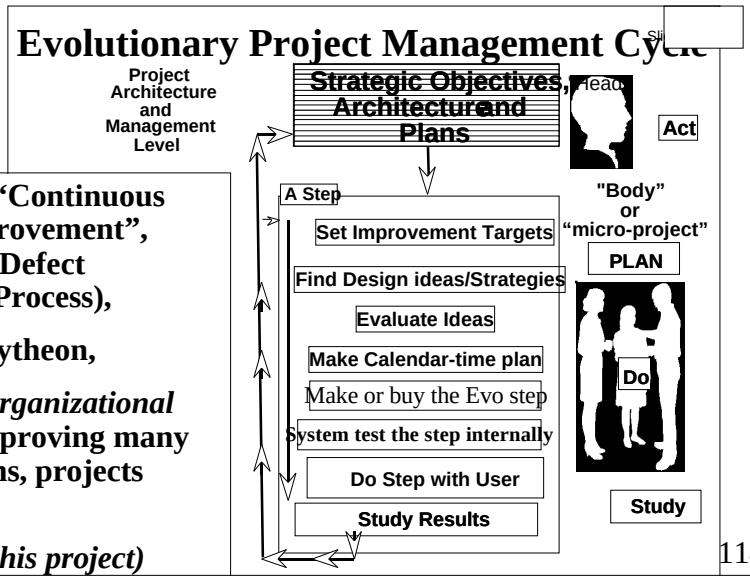
Case study by

stuart.woodward@natural-metrics.freeserve.co.uk

"Evolutionary Project Management", IEEE Computer, Oct 1999, P.49-57

113

Evo Cycles



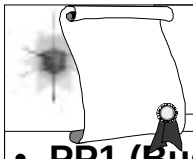
Evo allows "Continuous *Project* Improvement", while DPP (Defect Prevention Process),

used by Raytheon,

focuses on *organizational learning*, improving many people, teams, projects later;

(maybe not this project)

114



Evolutionary Results Delivery Method Project Planning Policy

- **PP1.(Budget) No project cycle shall exceed 2% of total budget before delivering measurable results to a real environment.**
- **PP2. (Deadline) No project cycle will exceed 2% of total project time (one week for a year's projects) before it demonstrates practical measurable improvement, of the kind targeted.**
- **PP3.(Priority) Project cycles which deliver the most planned results to customers, for the resources they claim, shall be delivered first, to the customer.**

The Step Definition: Template

- **Step Name: <a tag> [more detail like <which product>, <which area of application>]**
  - **Stakeholder: <who are you going to give value to??>**
  - **Implementor: <who is in charge of implementing this step>**
  - **Step Content:**
    - **<step tag> Gist <step gist>**
      - **Tasks**
        - » **<list of step tasks>**
  - **Step Value:**
    - **<numeric or rough estimate of value to stakeholder in terms of formal objectives planned level and scales>,**
    - **at least value on scale 0 (none) to 9 (highest)**
  - **Step Cost:**
    - **<Estimates of time and other costs (engineering hours) which are budgetted or constrainde by the Evo 2% policy>**
    - **At least cost on scale 0 (dirt cheap) to 9 (high and unpredictable)**
  - **Step Constraints:**
    - **< any legal, political, economic, security constraints imposed on implements>**
  - **Step Dependencies:**
    - **<anything which must be in place, finished, working properly, for us to be able to start this evostep or to complete it> <--<who says this is true?>**

The Step Definition: (people): Scribe Guide Tom

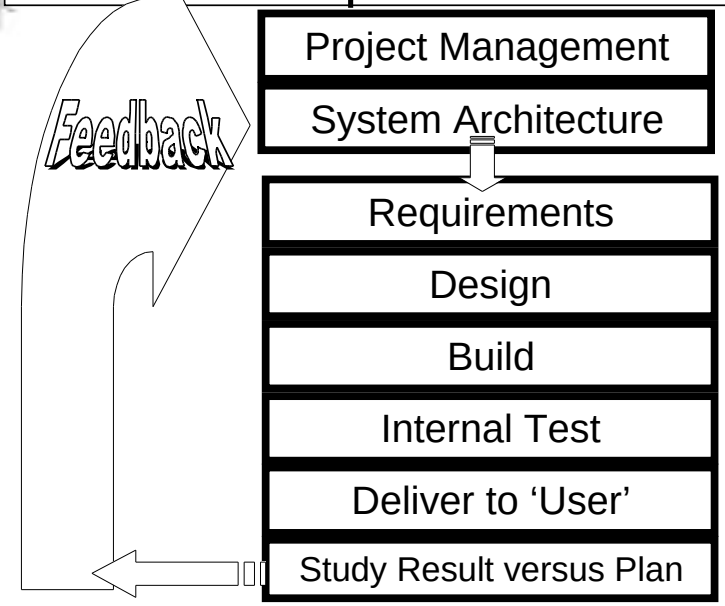
- **Step Name: Tutorial [Product 999, Basic]**
  - **Stakeholder: Marketing, XX (<agreed, Next Friday>)**
  - **Implementor: <XX>**
  - **Step Content:**
    - **HCTD :<Hard Copy Text document> <-- Can do 1 week MMM**
      - **Basic minimal functions.**
        - » **Step by Step Instructions, in English**
        - » **Focus on sales aspects, not how to do it (not yet, in this step)**
        - » **Go to specific web sites**
        - » **Pinpoint some characteristics of what we see on the terminal**
        - » **Compared with what we see on a PC or other terminal**
        - » **what instructions should be on the terminal to begin**
        - » **Intended audience: Marketing Guy**
        - » **Questionnaire for Stakeholder**
        - » **Process for Testing with stakeholder (example observation, times)**
      - **No illustrations, just text.**
  - **Step Value: (to TTT, Saleability) : <some possibility of value>,**
    - **Stakeholder Developers: value of feedback on a tutorial.**
  - **Step Cost: 10 hours per page, < 10 hours <--MMM**
  - **Step Constraints: must be deliverable within 1 calendar week.**
    - **At Least 3 hours of TTT's time for input and trial feedback**
  - **Step Dependencies:**
    - **<feature list of WWW and 77777 WWW Browser> <--MMM**

Step Impact Estimation and Accounting

	Cycle 1 I1[CA]		Cycle 2 I1[NY] & I4[CA]		Cycle 3 I1[DC] & I5		Cycle 4 I3[AZ]	
	Cycle	Sum	Cycle	Sum	Cycle	Sum	Cycle	Sum
GOAL-Q	30%	30%	40%	70%	10%	80%	30%	110%
GOAL-I	0%	0%	60%	60%	25%	85%	25%	110%
COST-C	2%	2%	5%	7%	4%	11%	39%	50%
Bene/Cost	30/2	30/2	100/5	130/7	35/4	165/11	65/39	220/50

- **An IE table for project management planning or feedback. It specifies the projected or actual impact of any set of design ideas done at a particular implementation cycle.**

Simple Evo Model



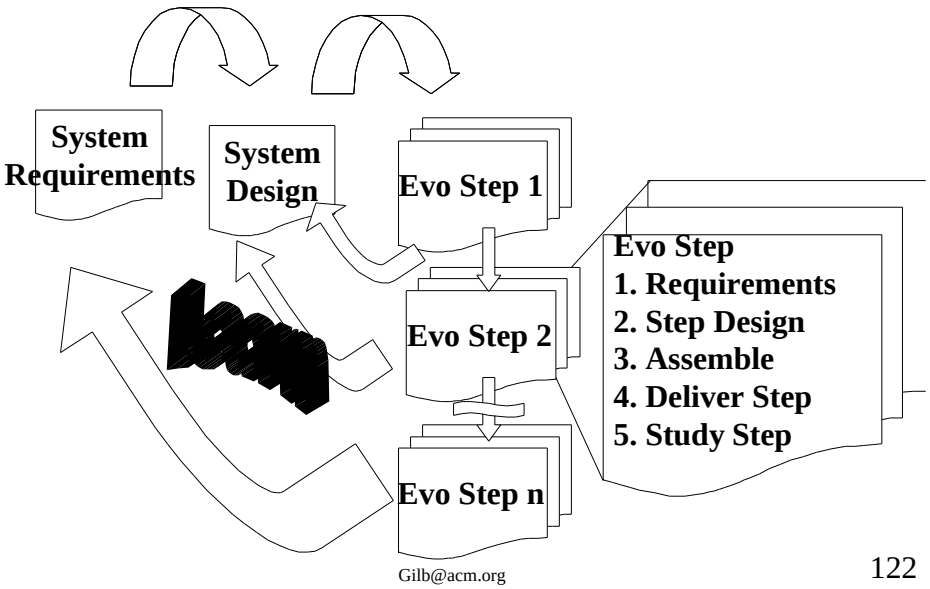
Basic Principles of Results Delivery

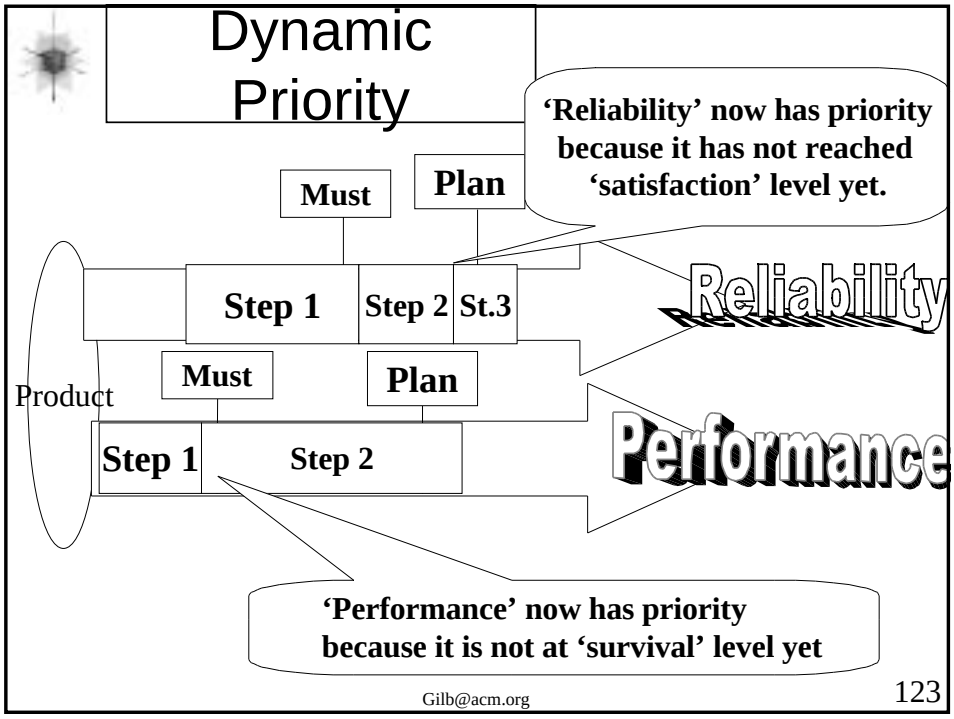
- **RD1. Any Project can be managed better using process control.**
- **RD2. Any change can be delivered as a series of smaller changes.**
- **RD3. No person knows all the results of a design in advance.**
- **RD4. No person can know what all the goals should be, in advance.**
- **RD5. You must be prepared to compromise intelligently with reality.**
- **RD6. Early delivery means early payback.**
- **RD7. The customer is always right, even when they change their goals.**
- **RD8. There is no real end to a project, if we have competition.**
- **RD9. You cannot foresee every change, but you can foresee change itself.**
- **RD10. Useful results are your only justification for existence.**

Evo Cousins: Process Control

Evo	**DPP Defect Prevention CMM 5**	**Inspection (Doc. Qual. Ctl.) CMM 3**	**SPC Statistical Process Control**
Project oriented	***Organization orientation***	***Document, and work process***	***General, but manufacturing bias***
Meet project goals or change them	*Improve general org. ability work*	*Approve work, clean work, analyze tasks*	*Improve work processes*
Project Manager	***Quality Director***	***Inspection process owner***	***Quality Control***

"Evo" model





Step Impact Estimation and Accounting

	Cycle 1 I1[CA]		Cycle 2 I1[NY] & I4[CA]		Cycle 3 I1[DC] & I5		Cycle 4 I3[AZ]	
	Cycle	Sum	Cycle	Sum	Cycle	Sum	Cycle	Sum
GOAL-Q	30%	30%	40%	70%	10%	80%	30%	110%
GOAL-I	0%	0%	60%	60%	25%	85%	25%	110%
COST-C	2%	2%	5%	7%	4%	11%	39%	50%
Bene/Cost	30/2	30/2	100/5	130/7	35/4	165/11	65/39	220/50

- **An IE table for project management planning or feedback. It specifies the projected or actual impact of any set of design ideas done at a particular implementation cycle.**

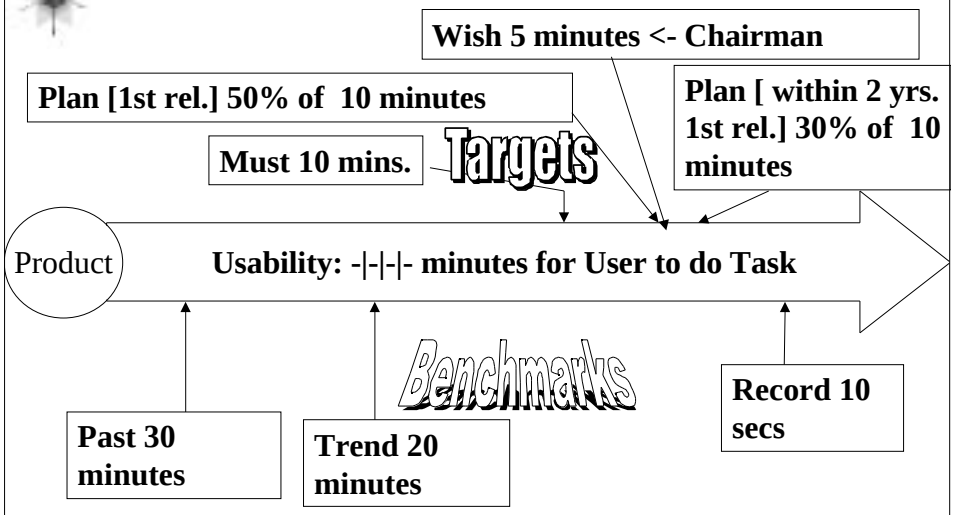
Gilb@acm.org

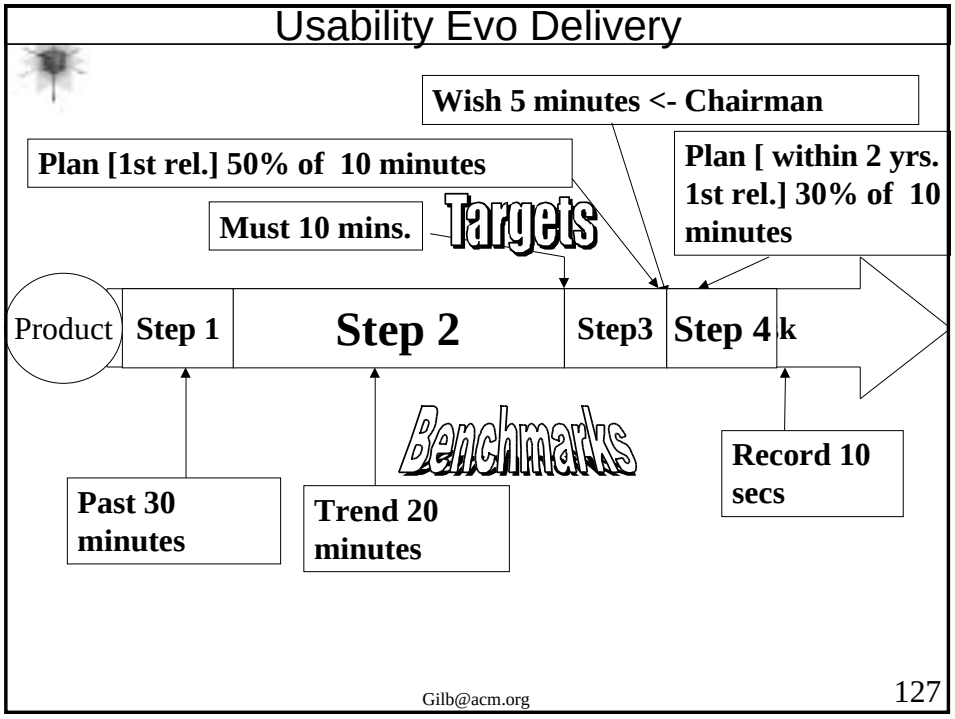
124

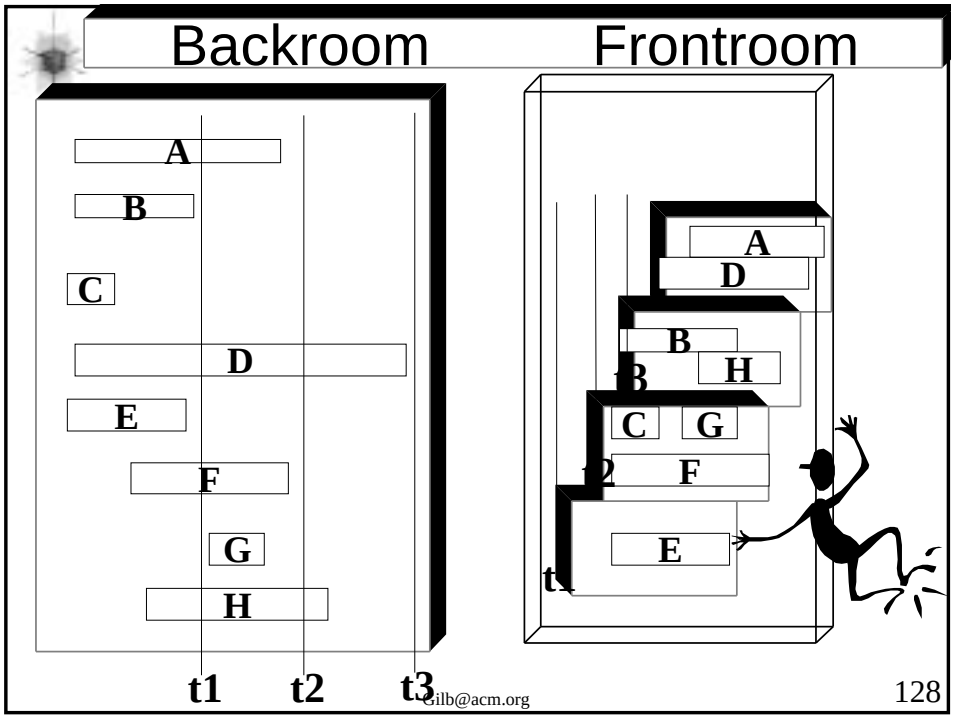
Impact Table for Step Management

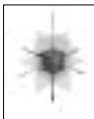
	Step #1 **A**: {Design-X, Function-Y}	Actual	Difference. - is bad + is good	Total	Step #2 **B:** {Design Z, Design F}	Actual	Difference	Total	Step #3 Next step plan
Reliability 99%-99.9%	50% ±50%	40%	-10%	40%	100% ±20%	80%	-20%	120%	0%
Performance 11sec.-1 sec.	80% ±40%	40%	-40	40	30% ±50%	30%	0	70%	30%
Usability 30 min.-30 sec.	10% ±20%	12%	+2%	12%	20% ±15%	5%	-15%	17%	83%
Capital Cost 1 mill.	20% ±1%	10%	+10%	10%	5% ±2%	10%	-5%	20%	5%
Engineering Hours 10,000	2% ±1%	4%	-2%	4%	10% ±2.5%	3%	+7%	7%	5%
Calendar Time	1 week	2 weeks	-1week	2 weeks	1 week	0.5 weeks	+0.5 wk	2.5 weeks	1 week

Usability Example Graphically




Usability Evo Delivery
Wish 5 minutes <- Chairman
Plan [1st rel.] 50% of 10 minutes
Plan [within 2 yrs. 1st rel.] 30% of 10 minutes
Must 10 mins.
Targets
Product
Step 1
Step 2
Step3
Step 4
k
Benchmarks
Past 30 minutes
Trend 20 minutes
Record 10 secs
Gilb@acm.org
127


Backroom
Frontroom
A
B
C
D
E
F
G
H
t1
t2
t3
Gilb@acm.org
128



Step Comparison Table

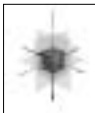
	Step Candidate **A**: {Design-X, Function-Y}	Step Candidate **B:** {Design Z, Design F}
Reliability 99%-99.9%	50%	100%
Performance 11sec.-1 sec.	80%	30%
Usability 30 min.-30 sec.	-10%	20%
Capital Cost 1 mill.	20%	5%
Engineering Hours 10,000	2%	10%
Performance/Capital Cost Ratio	80/20= *4.0*	30/5= *6.0*
Quality/Cost Ratio	120/22=**5.46**	150/15=**10.00**

Bill on Milestone Approach

- "the milestone approach is a major practice for us"
- Bill Gates in CUSUMANO95 , 18
- "Microsoft Secrets"

Mills on Project Control

- ***"Software Engineering began to emerge in FSD (IBM Federal Systems Division, from 1996 a part of Lockheed Martin) some ten years ago [about 1970] in a continuing evolution that is still underway.***
  - ***Ten years ago general management expected the worst from software projects – cost overruns, late deliveries, unreliable and incomplete software.***
  - ***Today [1980] , management has learned to expect on-time, within budget, deliveries of high-quality software.***
- ***A Navy helicopter ship system, called LAMPS, provides a recent example.***
  - ***LAMPS software was a four-year project of over 200 person-years of effort,***
  - ***developing over three million, and integrating over seven million words of program and data for eight different processors distributed between a helicopter and a ship,***
  - ***in 45 incremental deliveries.***
  - ***Every one of those deliveries was on time and under budget.***
- ***A more extended example can be found in the NASA space program,***
  - ***where in the past ten years, FSD has managed some 7,000 person-years of software development, developing and integrating over a hundred million bytes of program and data for ground and space processors in over a dozen projects.***
  - ***There were few late or overrun deliveries in that decade, and none at all in the past four years."*** ***Harlan Mills [IBM80, page 415].***



An example of a typical one-week Evo cycle at the HP Manufacturing Test Division during a project. [MAY96]

	Development Team	Users
Monday	• System Test and Release Version N • Decide What to Do for Version N+1 • Design Version N+1	
Tuesday	• Develop Code	• Use Version N and Give Feedback
Wednesday	• Develop Code • Meet with users to Discuss Action Taken Regarding Feedback From Version Nû1	• Meet with developers to Discuss Action Taken Regarding Feedback From Version Nû1
Thursday	• Complete Code	
Friday	• Test and Build Version N+1 • Analyze Feedback From Version N and Decide What to Do Next	

Step Risk Analysis (range of experiences)

	Step Candidate **A**: {Design-X, Function-Y}	Step Candidate **B:** {Design Z, Design F}
Reliability 99%-99.9%	50%±50%	100%±20%
Performance 11sec.-1 sec.	80%±40%	30%±50%
Usability 30 min.-30 sec.	-10%±20%	20%±15%
Capital Cost 1 mill.	20%±1%	5%±2%
Engineering Hours 10,000	2%±1%	10%±2.5%
Worst Case B/C ratio (1 to 3)	(0+40-10)/(21+3) = *1.25*	(80-20+5)/(7+12.5) = *3.33*
Best Case B/C ratio	(100+120+10)/(19+1) = **11.5**	(120+80+35)/(3+7.5)= **22.38**

Step Choice with 'Credibility' (of evidence)

	Step Candidate **A**: {Design-X, Function-Y}	Step Candidate **B:** {Design Z, Design F}
Reliability 99%-99.9%	50%±50%	100%±20%
Performance 11sec.-1 sec.	80%±40%	30%±50%
Usability 30 min.-30 sec.	-10%±20%	20%±15%
Capital Cost 1 mill.	20%±1%	5%±2%
Engineering Hours 10,000	2%±1%	10%±2.5%
Worst Case B/C ratio	(0+40-10)/(21+3) *=1.25*	(80-20+5)/(7+12.5) *=3.33*
æWorst WorstA case considering estimate credibility factor	0.8 x 1.25= ***1.00***	0.2 x 3.33= ***0.67***

A Credibility=0.8 (High)

B Credibility=0.2 (Low)

References, detailed papers, slides, manuscripts

"Competitive Engineering"

The new Planguage book

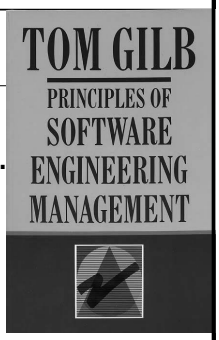
Addison Wesley Longman, 2001

Tom Gilb

Iver Holtersvei 2 N-1410 Kolbotn, Norway.

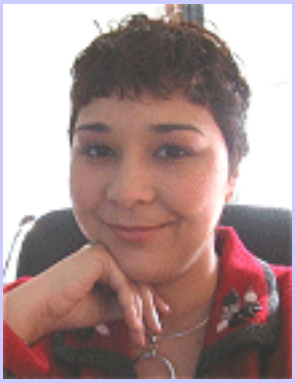
Home +47 66801697

Gilb@acm.org



www.Result-Planning.com

is my homepage, for my slides and book manuscripts



QW2001 QuickStart 6Q

Ms. Jeanette Folkes
(Modem Media)

WebTesting 101

Key Points

- How to build a quality assurance department, from people to process
- How to test web applications
- Tips & tricks to speed up the testing and test planning process

Presentation Abstract

The recent proliferation of interactive web site development in virtually every industry has created challenges for Quality Assurance Test departments. These challenges demand innovative solutions. This presentation will focus on those challenges and hopefully offer some insight for those new to web testing.

This presentation will discuss the challenges of web testing from a management and a production standpoint. Ms. Jeanette Folkes has been involved with testing for 8 years and her last 3 positions have been to build and establish a Quality Assurance presence for interactive agencies.

Ms. Folkes will outline solutions for developing a company wide test methodology, recruiting and training a skilled test team, and scheduling adequate time for testing. She will discuss which areas have been the most and the least receptive to integrating quality assurance.

In addition, she will address the challenges of testing Interactive web sites from a production standpoint. She will offer solutions for rapid test planning and test case execution, and will explore the limitations of automated testing in an interactive environment. This presentation will also offer suggestions on how and when to build your automated test scripts, using WinRunner for examples.

About the Author

Jeanette Folkes - Jeanette is the Director of Quality Assurance at Modem Media. Her experience has stemmed from supporting and managing a helpdesk, writing documentation, providing user training and managing a training group. She has studied to become a webmaster, has worked as a tester for 4 years and has been responsible for building and managing QA departments for the last four years. Her recent accomplishments have been to define and establish the Quality Assurance

departments at Cushman & Wakefield for Citibank, at Grey Advertising for Grey Direct, at Ogilvy & Mather for Ogilvy Interactive and currently at Modem Media. Jeanette is married and lives in New York.

MODEM MEDIA

Web Testing 101

Jeanette Folkes - Director, Quality Assurance

DATE: May 31, 2001

LOCATION: San Francisco

PRESENTED TO: Quality Week 2001

MODEM MEDIA

I. How to build a QA department

II. How to test web applications

III. Tips to speed up the process

… from people to process...

© 2001 MODEM MEDIA. Confidential and Proprietary

MODEM MEDIA

Background

MODEM MEDIA

Locations Worldwide

- Norwalk
- New York
- San Francisco
- Toronto
- Sao Paulo
- London
- Munich
- Hong Kong

MODEM MEDIA

Client List

- 3 Com
- Allianz Versicherung
- Amazon.co.uk
- America's Baby
- Asia Printing
- AT&T
- Avon
- Braun
- Cathay Pacific
- Christies
- Citibank
- Coca Cola Japan
- Coke/Diet Coke
- Commercial General Union
- CP Hotels (Fairmont)
- CSFB*direct*
- DaimlerChrysler
- Debis
- Delta Air Lines
- Deutsche Bank
- Dyson Home Appliances
- E*TRADE
- E-Bookers
- Edgewood Creek
- eHarlequin
- Elida Faberge Ltd.
- EPSCO
- Europaeische Reiseversicherung
- FT (Financial Times)
- GE
- General Motors
- Gessy Lever
- Globo.com
- GM
- Harper Collins
- Hong Kong Jockey Club
- HSBC
- IBM
- Infinite Supply
- Intel
- J&J (Vistakon)
- JCPenney
- John Hancock

MODEM MEDIA

Client List

- Kodak
- Koito Manufacturing
- Kraft
- Leirum
- Lloyds
- Meio e Mensagem
- Mercedes-Benz
- Michelin
- New Tokyo Dental
- Nihon Shiatsu
- Nippon Koei Consulting
- Oracle
- Philips
- Renaissance Asset Management
- Scotia Bank
- Siemens
- Sony Entertainment
- Starwood
- SUNDAY
- Swell.com
- Swire Properties
- Thomas Cook
- Tokyo Nissan
- Tokyo Sports and Recreation
- Tokyo Welfare
- UBS Warburg
- Unilever
- Valvoline
- Vodafone
- Websense
- Weight Watchers
- Wendy's
- X-Stream

MODEM MEDIA

Jeanette Folkes

- **Modem Media**
  - Director, Quality Assurance
- **OgilvyInteractive**
  - QA Manager
- **Grey Direct**
  - QA Manager
- **Cushman & Wakefield / Citibank**
  - QA Manager for Java application
- **Wall Street Access**
  - Training Manager
  - Director of Office Automation
  - WebMaster
- **WSP&R**
  - Trainer
- **Transammonia**
  - Help Desk Analyst
- **The NPD Group**
  - QA Tester for proprietary client server application
  - Project Manager

MODEM MEDIA

I. How to build a QA department

MODEM MEDIA

Considerations

- Quality Assurance growth
- Methodology
- Team
- Lab
- General process
- Documents
- Integration

MODEM MEDIA

Quality Assurance Growth

- Centralized in Norwalk
- Decentralized in Jan 2000
- NYO staff September 2000
- Automated tools purchased September 2000
- QA lab established October 2000
- QA Methodology adopted February 2001
- Dedicated NY testing staff ???

MODEM MEDIA

QA Methodology

- **What is Quality?**

  It is the word used to describe how well a product or service satisfies the customer's needs, wants and expectations.

  - Test plans, outlines of the areas to be tested, are written.
  - Test cases, documentation of the actual steps, are written.
  - Automated testing is used to improve testing.
  - Exceptions are logged in an exception tracking database.
  - Reports are generated.
- **What are you responsible for?**
  - **Copy**
  - **Images**
  - **Links**

MODEM MEDIA

QA Team

- **QA Director hired 9/00**
- **QA Manager**
- **Senior Tester**
- **Tester**
- **Junior Tester**
- **Consultants on an as needed basis**

… resource sharing
… cross-training

MODEM MEDIA

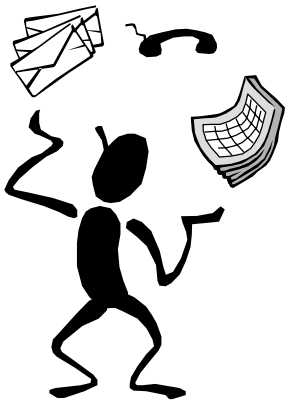
Skill Sets - Director / Manager

- Establish QA Methodology
- Build department
- Ensure TQM best practices
- Budgets
- Client interaction
- Day-to-day issues

MODEM MEDIA

Skill Sets - Senior Tester

- Responsible for all automated testing
- At least 4 years testing
- At least 2 years testplanning
- Familiarity with HTML, plug-ins, software
- Basic programming



MODEM MEDIA

Skill Sets - Tester

- Responsible for all testplanning
- At least 2 years testing
- Familiarity with basic HTML
- Good software background
- Flexible hours

MODEM MEDIA

Skill Sets - Jr. Tester

- Responsible for all manual testing
- Entry level position
- Familiarity with basic HTML
- Own web site
- College editor
- Has own computer
- Good software background
- PROOF their resume
- Flexible hours (evenings and weekends)

MODEM MEDIA

Interview Questions

- What is HTML?
- What is a URL?
- What is / how do you clear cache?
- What is a firewall?
- What is an ethernet connection?
- Name 3 different browsers.
- Name 4 ISP's.
- What are 5 means of connectivity?
- What is a cookie?
- What is stress testing?
- What are the latest browser versions?
- What is performance testing?
- What is a history file?
- What is cache?
- What is a hyperlink?
- What is Java?
- What is streaming media?
- What is a search engine?
- Name 4 search engines?
- Example of a plug-in?
- What is load balancing?
- How do you view Java console
- What is black box testing?

MODEM MEDIA

Sample test page

Can you spot the errors?

RiceRoses - Filled

Satin Roses meeasure 11" in height and come with a baby's breath spray of pearls and green leaf on each stem.

Roses come in several different colors to coordinate with theme or color scheme.

Roses can filled with our standard fillings - Birdseed, Paper Confetti or Potpourri, or our deluxe fillings - Heart Shaped Biodegradable Rice, Candy Conversation Hearts, Heart Shaped Silver Metallic Confetti or Opal Iridescent Confetti.

(Not all locations allow you to throw birdseed or rice. Check with your local establishment before ordering.)

Orders of 100 or more filled Roses receive a FREE basket.

Small baskets can be purchased for $12.50.

Request a free sample!

www.OceansOfRoses.com

Standard Rose	$1.25
Tulle	.20
Standard Filling	.20
Biodegradable Heart Shaped Rice	.30
Conversation Hearts	.35
Silver Heart Shaped Metallic Confetti	.40
Opal Iridescent Confetti	.55
Personalized Ribbon	.45

MODEM MEDIA

Sample test page

Can you spot the errors?

RiceRoses - Filled

Satin Roses meeasure 1" in height and come with a baby's breath spray of pearls and green leaf on each stem.

Roses come in several different colors to coordinate with theme or color scheme.

Roses can filled with our standard fillings - Birdseed, Paper Confetti or Potpourri, or our deluxe fillings - Heart Shaped Biodegradable Rice, Candy Conversation Hearts, Heart Shaped Silver Metallic Confetti or Opal Iridescent Confetti.

(Not all locations allow you to throw birdseed or rice. Check with your local establishment before ordering.)

Orders of 100 or more filled Roses receive a FREE basket.

Small baskets can be purchased for $12.50.

Request a free sample!

www.OceansOfRoses.com

Standard Rose	$1.25
Tulle	.20
Standard Filling	.20
Biodegradable Heart Shaped Rice	.30
Conversation Hearts	.35
Silver Heart Shaped Metallic Confetti	.40
Opal Iridescent Confetti	.55
Personalized Ribbon	.45

MODEM MEDIA

Team hours

- 10:00am - 10:00pm, Monday - Saturday
- Consultants on an as needed basis

- Staggered shifts
  - Monday - Friday
    - 10:00 - 6:00
    - 12:00 - 8:00
    - 2:00 - 10:00 (Jr.)
  - Tuesday - Saturday
    - 10:00 - 6:00 (Jr. / Tester)

MODEM MEDIA

Continuing Education

- Quality Assurance Institute certification
  - Certified Quality Analyst (CQA)
  - Certified Software Test Engineer (CSTE)
- Research
  - How do I ….. ?
- Training
  - General QA
  - Product specific
- Industry Group Affiliations

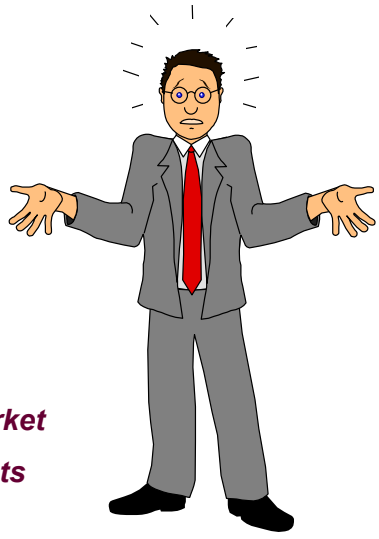
MODEM MEDIA

Testing lab

- Who is your audience?
  - Browser
  - Operating system
  - Screen resolution
  - Screen colors
  - Plug-ins
  - Type of computer
  - Connection

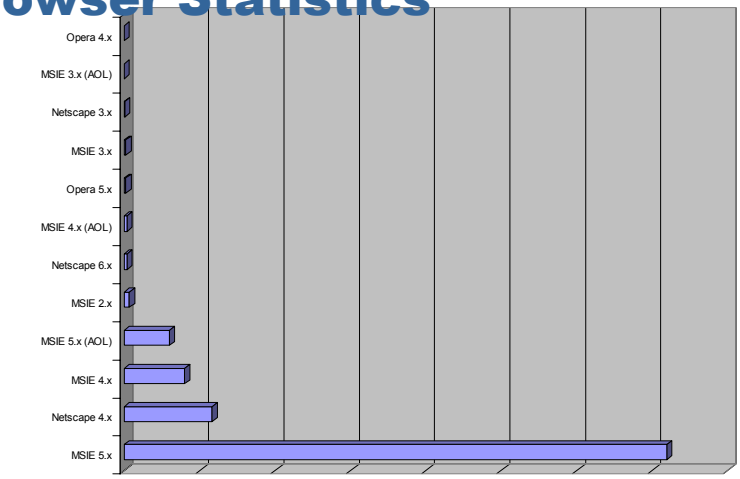
… Stat Market

… Live Stats



MODEM MEDIA

Browser Statistics

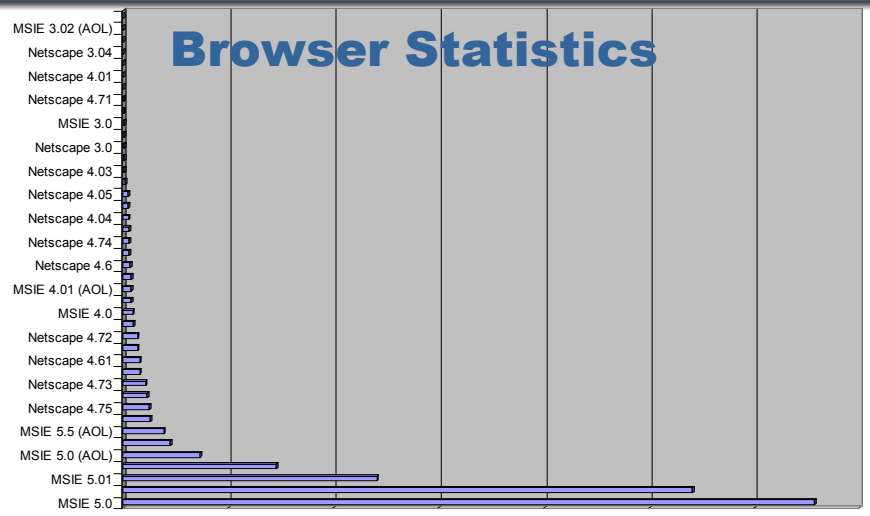


	MSIE 5.x	Netscape 4.x	MSIE 4.x	MSIE 5.x (AOL)	MSIE 2.x	Netscape 6.x	MSIE 4.x (AOL)	Opera 5.x	MSIE 3.x	Netscape 3.x	MSIE 3.x (AOL)	Opera 4.x
%age	71.99%	11.68%	8.05%	6.02%	0.72%	0.41%	0.40%	0.26%	0.25%	0.15%	0.05%	0.01%

Source: StatMarket, 2001

MODEM MEDIA

Browser Statistics

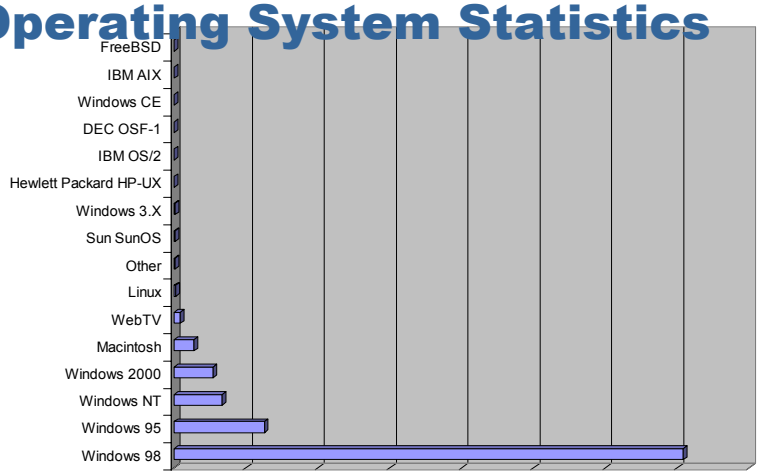


	MSIE 5.0	MSIE 5.01	MSIE 5.0 (AOL)	MSIE 5.5 (AOL)	Netscape 4.75	Netscape 4.73	Netscape 4.61	Netscape 4.72	MSIE 4.0	MSIE 4.01 (AOL)	Netscape 4.6	Netscape 4.74	Netscape 4.04	Netscape 4.05	Netscape 4.03	Netscape 3.0	MSIE 3.0	Netscape 4.71	Netscape 4.01	Netscape 3.04	MSIE 3.02 (AOL)
%age	32.85	12.07	3.68%	1.95%	1.27%	1.09%	0.80%	0.71%	0.47%	0.40%	0.36%	0.31%	0.29%	0.26%	0.08%	0.07%	0.07%	0.04%	0.03%	0.02%	0.01%

Source: StatMarket, 2001

MODEM MEDIA

Operating System Statistics

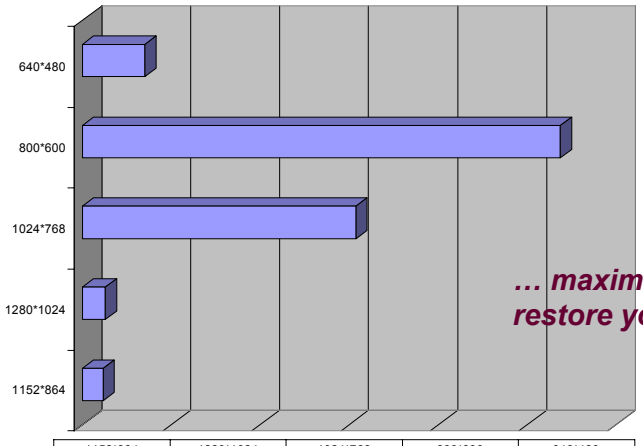


	Windows 98	Windows 95	Windows NT	Windows 2000	Macintosh	WebTV	Linux	Other	Sun SunOS	Windows 3.X	Hewlett Packard HP-	IBM OS/2	DEC OSF-1	Windows CE	IBM AIX	FreeBSD
%age	70.85%	12.58%	6.70%	5.44%	2.83%	0.90%	0.25%	0.15%	0.13%	0.12%	0.01%	0.01%	0.01%	0.01%	0.01%	0.01%

Source: StatMarket, 2001

MODEM MEDIA

Screen Resolution



	1152*864	1280*1024	1024*768	800*600	640*480
%age	2.31%	2.57%	30.79%	53.74%	7.05%

… maximize and restore your window

Source: StatMarket, 2001

MODEM MEDIA

Connection Speed

- Dial up - 56k
- T1
- Web TV
- DSL
- Cable

… especially important for audio or video

MODEM MEDIA

Hardware

- PC vs. Mac
- Memory
- Processor
- Dual boot
- Imaging
- Who has access to lab

… No tweaking!!!!

MODEM MEDIA

Software

- One version of IE
- Every version of Netscape
  - install in separate directories
  - maintain consistent naming convention
- Defect tracking
- Adobe Acrobat
- Link checker

… No tweaking!!!!

MODEM MEDIA

General Process

- Test plans are written *
- Test cases are written *
- Automated testing is used to improve testing *
- Automated testing is used to improve turnaround *
- Exceptions are logged in an exception tracking database
- Reports are generated

** optional / workaround*

MODEM MEDIA

Documents

- QA Checklist
- Test plan
- Test cases
- Exception reporting
- QA authorization report / Release report

- Project information -- MS Word

MODEM MEDIA

QA Checklist

- Browsers / OS's
- Required plug-ins and versions
- Alt tags
- Print testing
- Links - pop-ups, new windows
- Forms - validation
- Resolution
- Load testing *… date and time stamped*
- Connection speed
- Any other relevant information

MODEM MEDIA

Test Plan

- General overview of what's being tested
- Lists all test cases created
  - Should have a one-to-one relationship to requirements document
- Cannot be done until documentation is received
- High level template

MODEM MEDIA

Test Cases

- Documentation of the actual steps
  - Click the home button
  - Verify you reach the home page
- Literal
- Non-technical user
- Cannot be done until documentation is received

… test case matrix

- WinRunner automated tests

```
#this will check the top nav
set_window ("Asia and the Middle East", 3);
web_image_click("Home", 48, 12);
set_window ("Oceania Air Lines - Welcome to Oceania", 4);
set_window ("Browser Main Window", 0);
toolbar_button_press("MainToolBar", "Back");
```

MODEM MEDIA

Exception Reporting

MS Access database

- **URL**
- **Problem Type**
- **Reproducible**
- **Browser / OS**
- **Tester**
- **Test date**
- **Assignee**

- *TRACKWeb*
- *Test Track*
- *Track Gear*
- *Track Wise*
- *Lotus Notes*
- *Zero Defect*

MODEM MEDIA

QA Authorization Report

- **Exceptions by type, quantity**
- **Sign off from QA and authorizing person**

- **Signatures, signatures, signatures, signatures, signatures, signatures**

MODEM MEDIA

Integration

- Include QA in all schedules and estimates
- Initial QA performed on Asset Inventory
- Test plans are written
- Test cases are generated
- Testing begins
- Exceptions logged
- QA authorization report

… QA Demo Day

MODEM MEDIA

QA Demo Day

- 1/2 day -- full day
- Conference room
- Balloons, streamers, decorations, food
- Handouts
  - Test cases
  - Test plans
  - Exception reports
  - Personalized items
    - *"We ♥ Quality Assurance"*
- "So you wanna be a tester"
- Demonstrations

… It's coming!

MODEM MEDIA

Real World Challenges

QA needs money and an enforcer

- Staff
- Equipment
- Working with developers
- Getting management buy-in
- CYA

MODEM MEDIA

Staff

- Keeping them motivated
  - Growth
  - Ownership
- Showing appreciation
  - Comp time
  - Flexible hours
- Remaining flexible

MODEM MEDIA

Equipment

- QA Lab
- Each tester has 2 computers
- Dial up connection

MODEM MEDIA

Working With Developers

- Avoiding us vs. them
- Be very specific

MODEM MEDIA

Getting Management Buy-in

- More than $
- Constant reporting
  - Let them know before the client does

MODEM MEDIA

CYA

- Adobe Acrobat
- MS Word
- Export all exceptions

MODEM MEDIA

QA Notes			
Producer	Pam James	**Billing Code**	OCAL.0245
Account Manager	Sarah Correro	**Job Desc.**	~3 HTML pages
Estimated Days / Hours	0 / 3	**Actual Days / Hours**	0 / 2.5
Client	Oceania Air Lines	**URL**	http://159.125.14.25/oceania/dev/

1/21/00 11:30am jcf – Met with production team. This will be a small 3 page project to highlight Oceania's new in-seat entertainment. This job was estimated at 3 hours and is scheduled to start QA 1/25 and is due to be delivered 1/27.

1/23/00 bl – Wrote initial test plan (K:\Oceania\Testplans\ocal0245.doc)

1/24 5:30pm bl – Received QA checklist

1/25 jcf – Met with Pam. She says assets have been delayed 2 days. New QA date is 1/27 and new delivery date is 1/29.

1/27 1:45 vl – Tested 3 pages, no defects to report.

MODEM MEDIA

How to test web applications

MODEM MEDIA

What are you testing

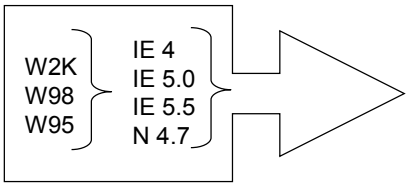
- Copy - proofreader
- Links - link checker
- Load time

MODEM MEDIA

Estimating the testing process

- What are you testing
  - 30 minutes for initial check
    - proofread
    - link check
    - gui check
    - automated script
  - 2 minutes per browser /OS

*(12 browser/OS - 1) * 2 minutes per page = 22 minutes*
22 minutes + 30 minutes for initial check = 52 minutes



52 minutes per page

MODEM MEDIA

Production-related challenges

- Finding adequate time for Test Planning
- Planning with incomplete or no specifications
- Finding time for regression testing
- Finding time for Automated testing

MODEM MEDIA

Documentation required for Test Planning

- Functional Design Document
  - What is this project supposed to do
- Sitemap
  - What pages are in the site
- Copy deck
  - What is the site supposed to say
- iBoards
  - What elements should be on the page

MODEM MEDIA

Test Planning with no Documentation

- QA Checklist
- Sitemap Tools
  - Astra Site Manager
  - Linkbot

MODEM MEDIA

Regression Testing

- Finding Time
- An automated regression test regime
  - Linkbot
  - Adobe Acrobat
  - SilkTest / WinRunner

MODEM MEDIA

Automated Testing

- Requires additional time for planning
- Requires code to be frozen or complete
- Does not work with certain interactive media

MODEM MEDIA

Color palette

This is the title color. Please use the title color for highlight also.

Please use only these colors for text

This is a good color for the top row of charts

MODEM MEDIA

Tips to speed up the process

MODEM MEDIA

Automated Testing

- How to plan your tests
  - Link check
    - needs to be run once
  - Gui check
    - needs to be run in all required browsers/OS's
- Use test scripts as your test cases
  - Use results files for pass/fail

MODEM MEDIA

QA Lab

- Consistency
  - All paths should be identical
  - Need dial-in access
    - to start scripts
    - to check scripts

MODEM MEDIA

Netscape

- Crashes often
  - Dial into your computer
- Do not create profiles

MODEM MEDIA

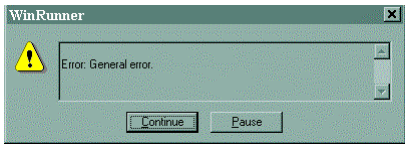
WinRunner Notes

- If an object appears on the border of the screen, WinRunner will not see it

MODEM MEDIA

WinRunner Notes

- If WinRunner gives you a General Error, try renaming the object



MODEM MEDIA

WinRunner Notes

- Report the browser in your script

```
win_get_info("Browser Main Window","owner",value);
report_msg (value);
```

MODEM MEDIA

WinRunner Notes

- **Use the automated script to check alt tags**

```
set_window ("Asia and the Middle East", 3);

web_obj_get_info ("Travel", "ALT", value);
   if (value == "Travel")  ;
   else report_msg("Travel alt tag failed");
```


MODEM MEDIA

WinRunner Notes

- **Create checkpoints for IE and Netscape**

```
#these are the gui checks
set_window ("Asia and the Middle East", 3);
win_get_info("Browser Main Window","owner",value);
# Returns "iexplorer.exe" or "netscape.exe"

if (value == "IEXPLORE.EXE")
          #checkpoint for Explorer
          win_check_gui("Asia and the Middle East", "list7.ckl", "gui11", 1);
else if (value == "iexplore.exe")
          #checkpoint for Explorer
          win_check_gui("Asia and the Middle East", "list7.ckl", "gui11", 1);
else if (value == "NETSCAPE.EXE")
          #checkpoint for Netscape
          win_check_gui("Asia and the Middle East", "list2.ckl", "gui2", 1);
else if (value == "netscape.exe")
          #checkpoint for Netscape
          win_check_gui("Asia and the Middle East", "list2.ckl", "gui2", 1);
else if (value == "Netscape.exe")
          #checkpoint for Netscape
          win_check_gui("Asia and the Middle East", "list2.ckl", "gui2", 1);
else
          report_msg("Browser not found");
```

MODEM MEDIA

WinRunner Notes

- **Always use full paths**

```
win_max ("Browser Main Window");
call_close "K:\\Quality Assurance\\Oceania\\asia_gui_check" ();
call_close "K:\\Quality Assurance\\Oceania\\english_intl_gui_check" ();

call_close "portuguese_gui_check" ();
call_close "german_gui_check" ();
call_close "italian_gui_check" ();
```


MODEM MEDIA

WinRunner Notes

- **Setting your browser**

```
#this will delete Netscape from the registry and set Netscape 4.75 as the default browser
report_msg ("Starting N4.75 tests.");
del_reg_key(HKEY_LOCAL_MACHINE,"Software\Microsoft\Windows\CurrentVersion\App Paths\Netscape.exe")
create_reg_key(HKEY_LOCAL_MACHINE,"Software\Microsoft\Windows\CurrentVersion\App
Paths\Netscape.exe","","C:\Program Files\Netscape\Communicator 4_75\Program\Netscape.exe");
wait(5);
web_browser_invoke (NETSCAPE,"");
wait(5);
```

MODEM MEDIA

Summary

- Define Methodology
- Hire staff
- Integration

… from people to process

MODEM MEDIA

Questions?

QW2001 QuickStart 4Q

Mr. Greg Clower
(SDT Corporation)

Establishing a Wireless Telecommunication Test Automation System

Presentation Abstract

Building an environment to successfully test intelligent network peripherals presents an array of complex problems to resolve. The target environment integrates various SS7 protocols, a proprietary protocol, and voice recognition subsystem -- and requires a controlled and synchronized test environment. Learn how a test automation approach allows the software engineer control over the peripheral interfaces and provides for the testing of the entire call flow sequence, its initiation and consequential message traffic. Discover how this approach provides for function testing as well as scalability for automated performance, load and stress testing.

About the Author

Greg Clower is Lead Automation Engineer for Software Development Technologies and has over 15 years of experience in software quality and test. Before joining SDT, Greg was a Sr. Design Engineer at Lucent Technologies in charge of Telecommunications protocol integration test tools.

Greg has a B.A. degree from Indiana University and has taken core courses in Math, Computer Science and Unix Programming at San Jose State University and University of California, Santa Cruz.

Establishing a Telecommunication Test Automation System

Greg Clower

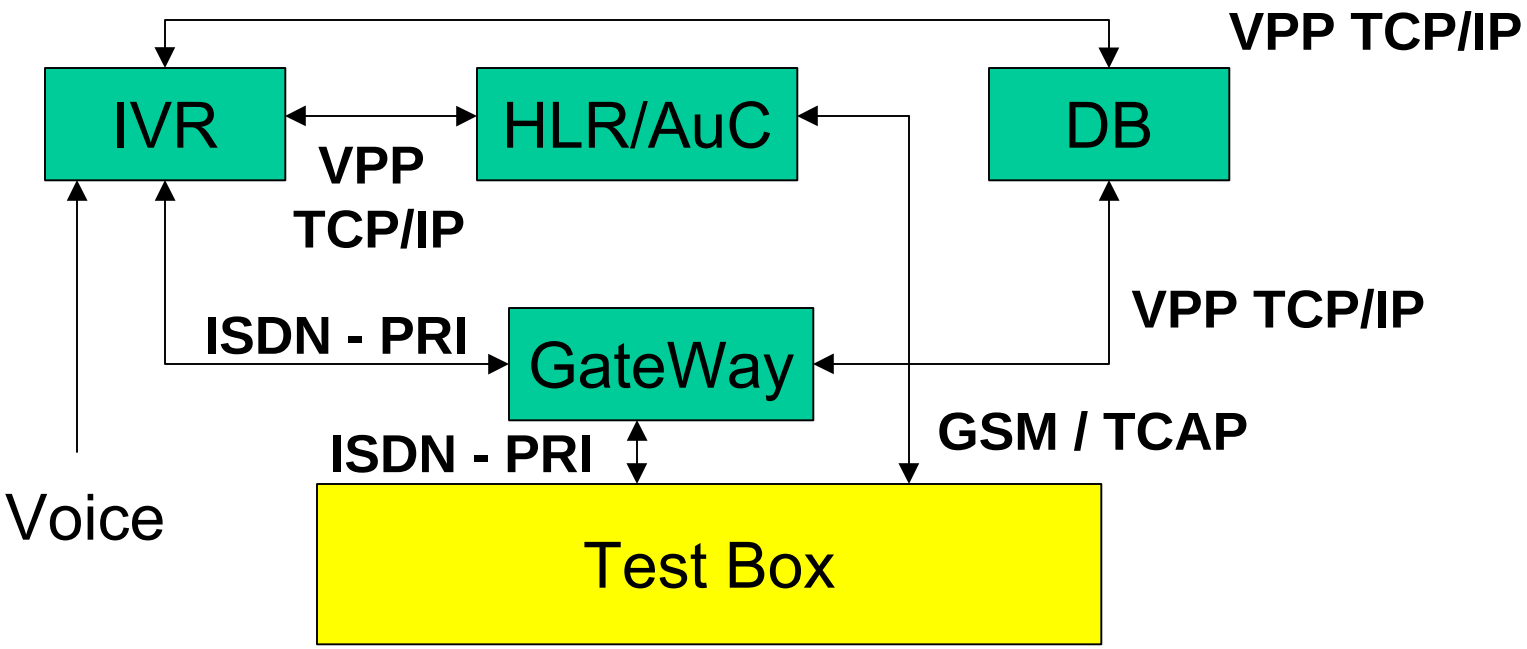
www.sdtcorp.com **sdt@sdtcorp.com**

Agenda

- **Test Environment**
- **Define Requirements**
- **Problem Analysis**
- **Solution**
- **Conclusions**

SOFTWARE DEVELOPMENT TECHNOLOGIES

Test Environment



- **Interactive Voice Response Application – (IVR)**
- **Home Location Register/Authentication Center - (HLR/AuC)**
- **Very Proprietary Protocol over TCP/IP – (VPP)**

Agenda

- Test Environment
- **Define Requirements**
- Problem Analysis
- Solution
- Conclusions

SOFTWARE DEVELOPMENT TECHNOLOGIES

Define Requirements (1 of 2)

- Protocol Support
- IVR Application Stimulation
- Separate Test Design and Test Automation
- Troubleshooting Capability

Define Requirements (2 of 2)

- Absolute Interface Control
- Integrated Test Solution
- Cost Effective Test Solution
- Extensible Environment
- Maintainability

Protocol Support

- **ISDN - PRI for Call Control**
- **GSM / TCAP**
- **VPP - TCP/IP**

IVR Application Stimulation

- IVR Applications Respond to Voice
- Play Voice on Voice Channel
- Vocabulary
  - Phrases
  - Numbers
- Number Representation
  - 2 hundred 37
  - 2-3-7
  - 2-37

Separate Test Design and Automation

- **Maintenance Issue**
- **Role Based Test Development Methodology**
- **Testing Using Action Words**

What is an Action Word ?

- Description of the test steps to be performed
  - Definition
    - Login
    - Dial in
  - Parameters
    - User ID Password
    - Telephone Number
- Enables linkage between Test Design and Test Case Processor
- Action Word Implementation Function
  - Implemented by Automation Engineer
  - Manipulates application to achieve intended function

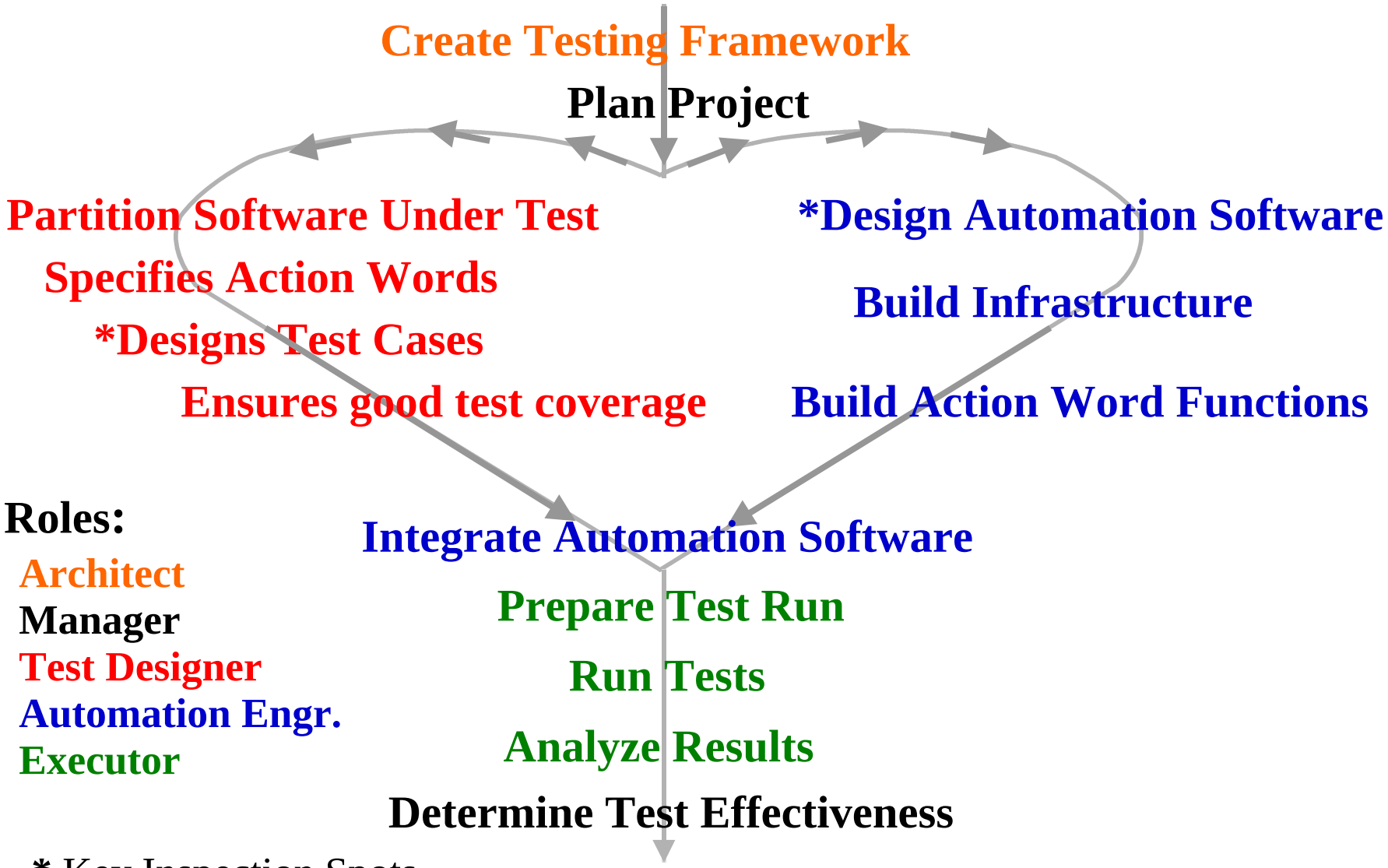
Test Automation – Specialized Roles (1 of 2)

- Application Domain Expert
- Test Manager
  - Plans Project
  - Determines Test Effectiveness
- Test Architect
  - Creates Testing Framework
- Test Designer
  - Partitions System Under Test
  - Specifies Action Words
  - Designs Test Cases
  - Ensures Good Test Coverage

Test Automation – Specialized Roles (2 of 2)

- **Test Automation Engineer**
  - Designs Automation Software
  - Builds Infrastructure
  - Builds Action Word Implementation Functions
  - Integrates Automation Software
- **Test Executor**
  - Prepares Test Run
  - Runs Tests
  - Analyzes Results

Roles-Based Key Activity Overview



Roles:

- Architect
- Manager
- Test Designer
- Automation Engr.
- Executor

***** Key Inspection Spots

Separate Test Design and Automation Benefits

- Parallel Work Paths
- Enhanced Productivity
- Enhanced Efficiency
- Decreased Time to Market

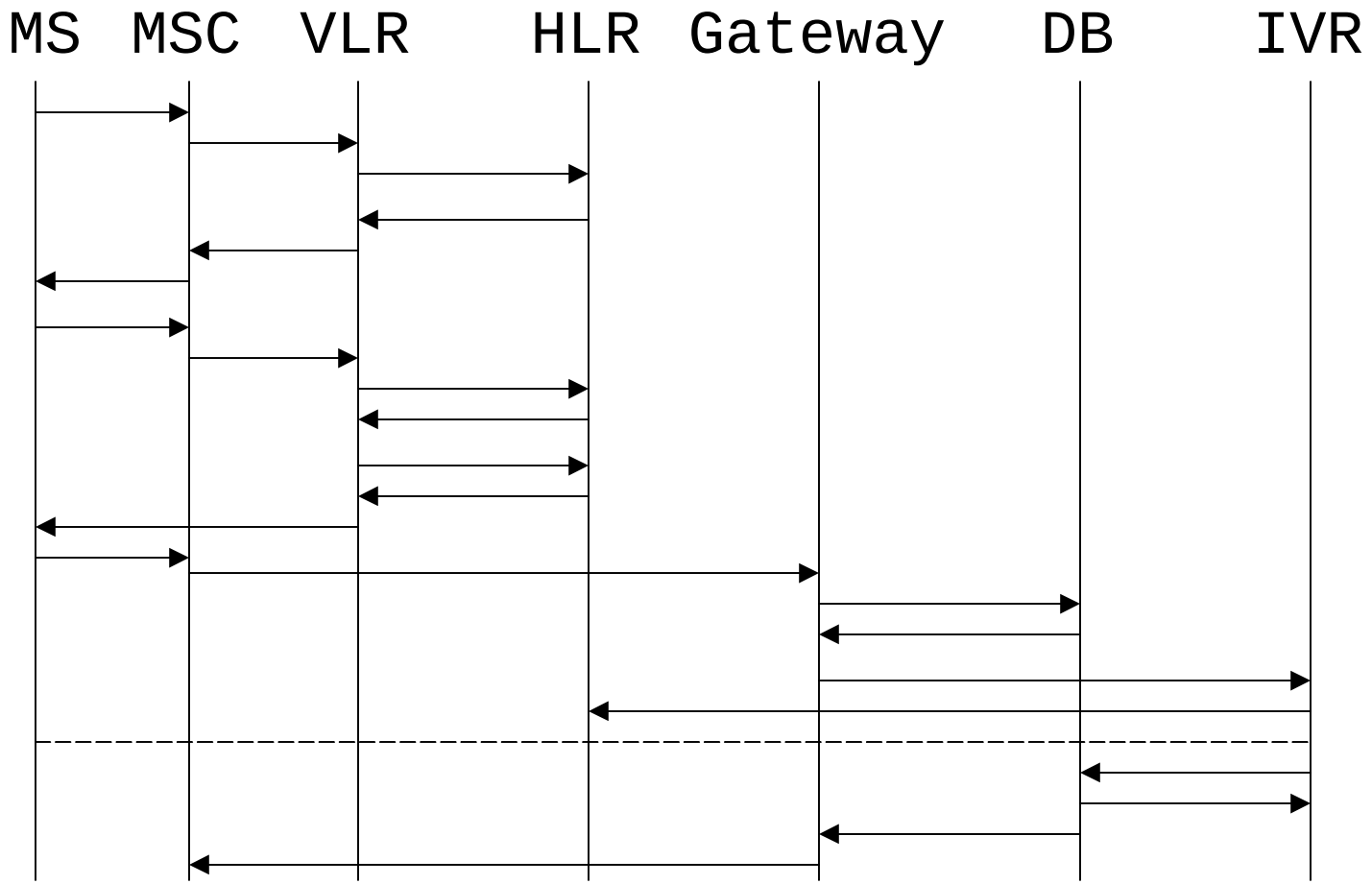
Troubleshooting Capability

- Gather Run-time Test Data
- Examine input Interface
- Examine output Interface
- Choose Right level of detail for Interface
- Results

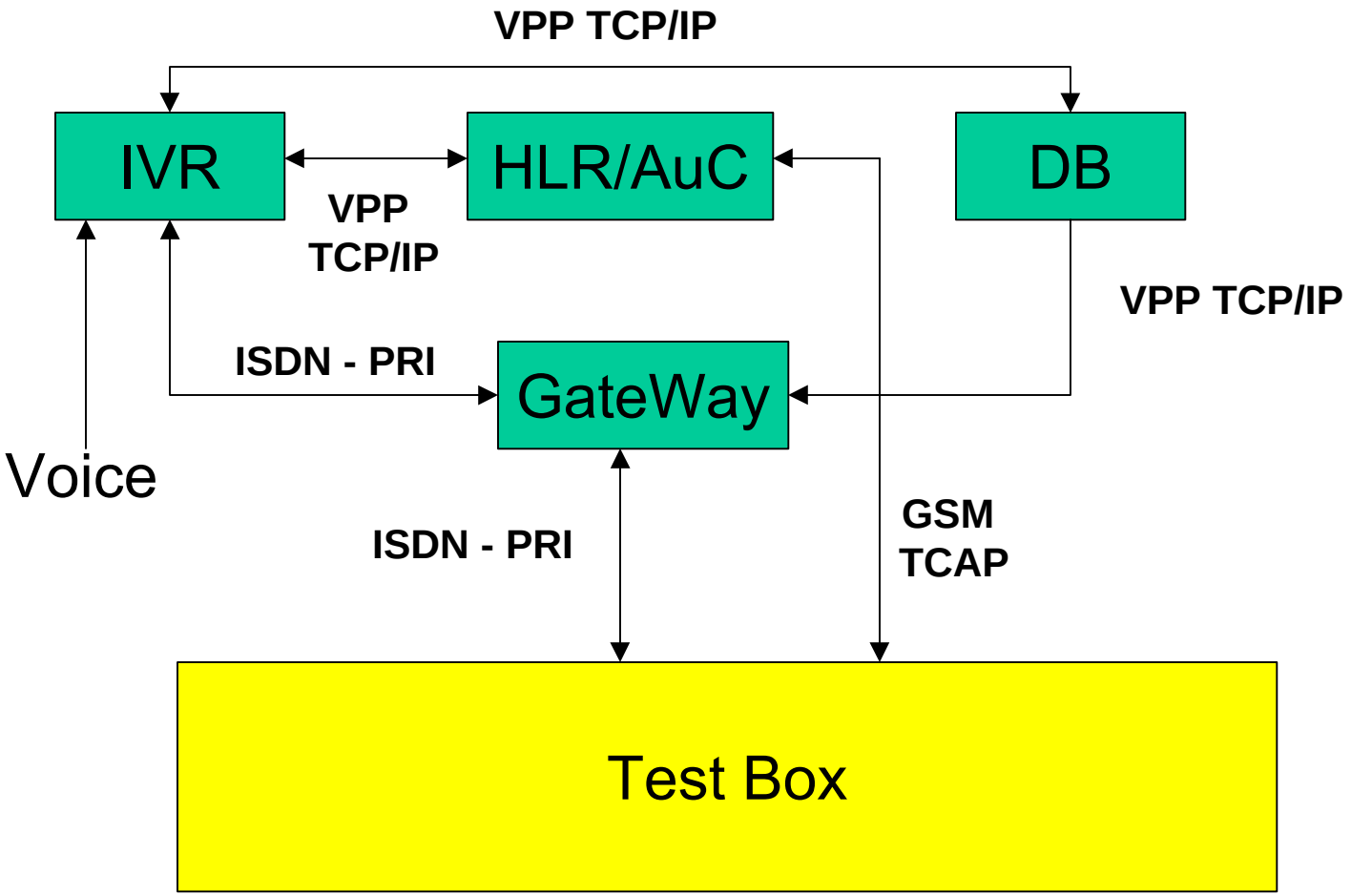
Absolute Interface Control

- **The Message Flow**
- **The Interfaces**

The Message Flow Diagram



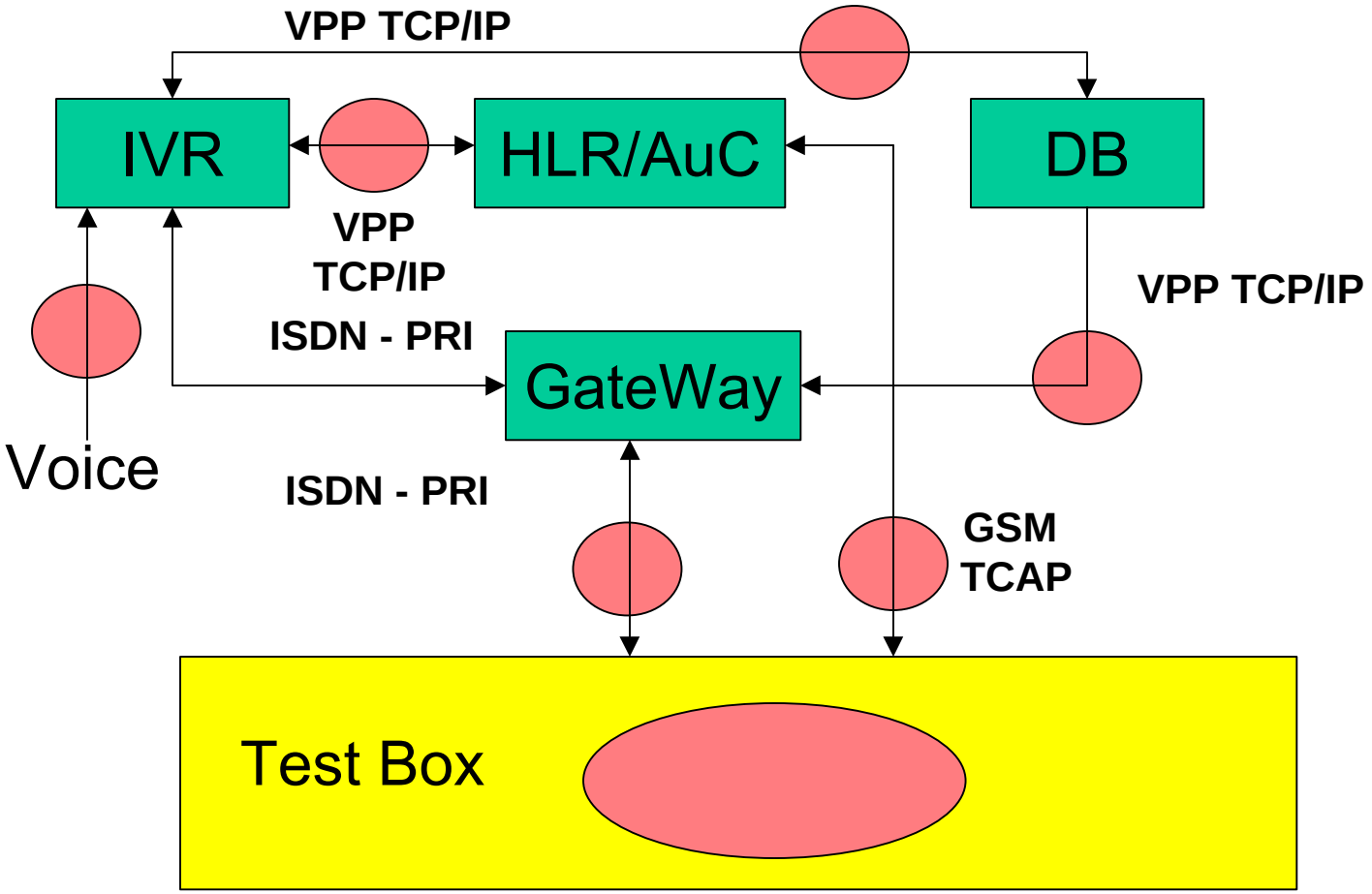
The Interfaces



Integrated Test Solution

- **Automatic Test Control**
- **Highly controlled Interfaces**
- **Synchronized message flow**
- **Test Box Internals**
- **Reports**

Integrated Test Solution



Cost Effective Test Solution

- **Existing Tools**
  - **Protocol Testers**
  - **Bulk Call Generators**
- **Buy/Build**
  - **Generally buy is preferred**

Extensible Environment

- **Call Control**
  - **ISUP**
- **TCAP Applications**
  - **IS-41**
  - **INAP**
  - **SMS**
  - **???**

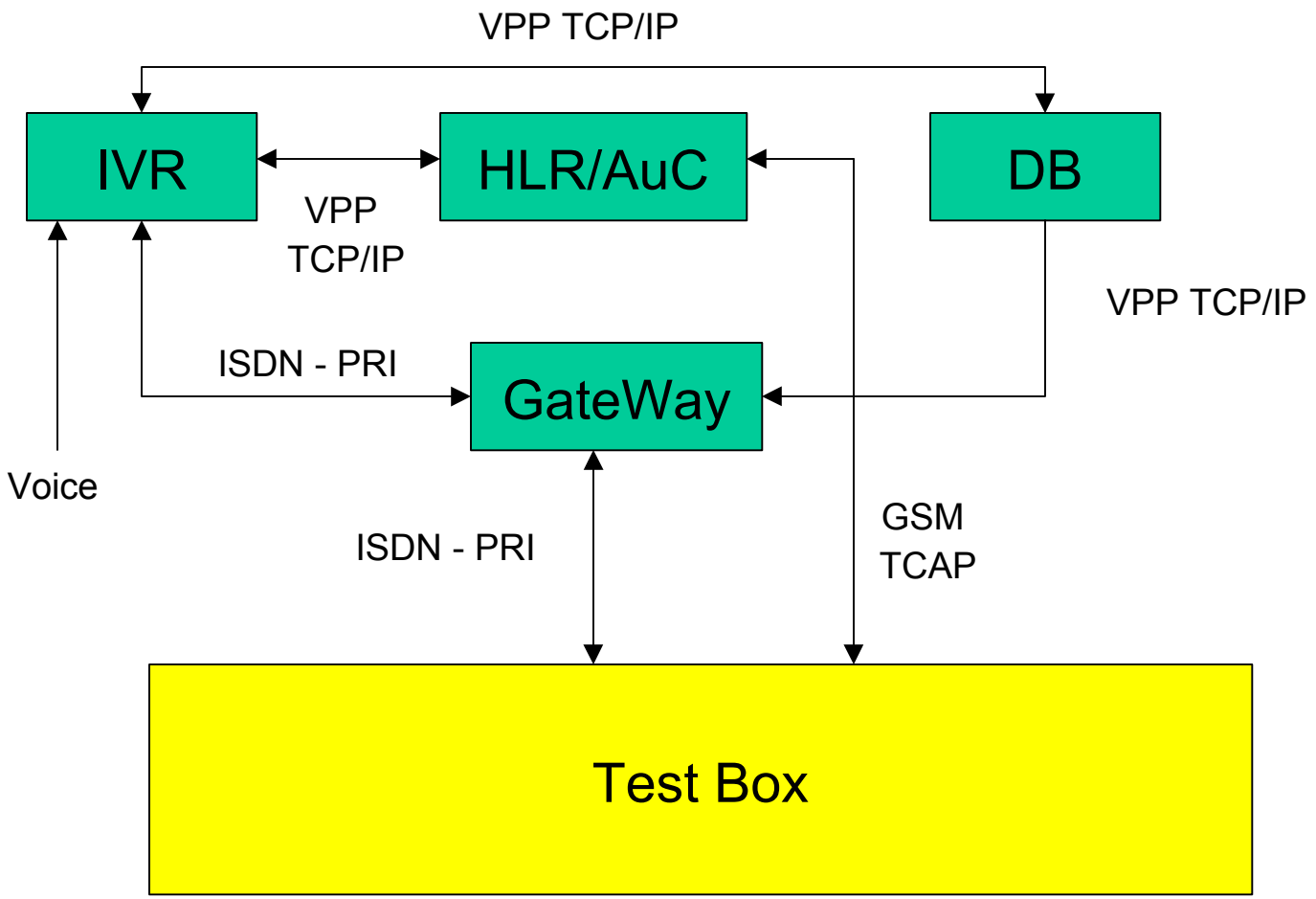
Agenda

- Test Environment
- Define Requirements
- **Problem Analysis**
- Solution
- Conclusions

Problem Analysis

- What Tools are on the Market?
  - Automatic Test Control
  - Call Control
    - Bulk Call Generators
  - Protocol Generation
    - Protocol Testers
- Do they do what we need to do?

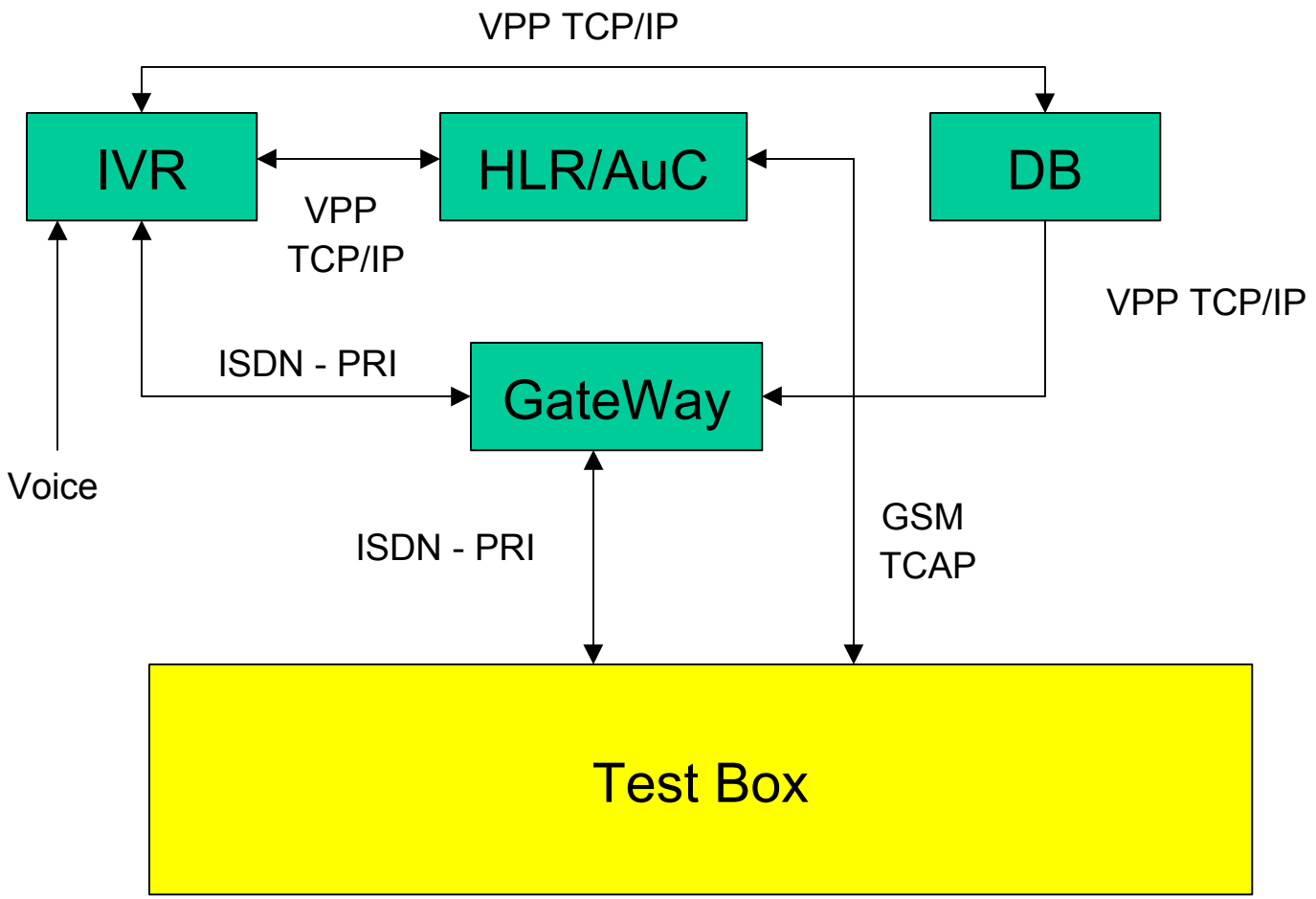
Input Calls



Bulk Call Generators

- Difficult to integrate with Testware and other equipment
- No VPP TCP/IP connection
- Testers want to test not specialize in the tool
- The tool gets in the way
- Expensive – Very Expensive
- Moderately Difficult to use
- Do call control not protocol testing
- System response validation – not easy

Protocol Output and Response



Protocol Testers

- Difficult to integrate with Testware and other equipment
- Proprietary Hardware solution
- Testers want to test not specialize in the tool
- The tool gets in the way
- Very Expensive
- Often Very Difficult to Learn to Use
- Do Not easily allow the use use of bad input

Problem Analysis Conclusions (1 of 2)

- Existing Tools are Expensive – Very Expensive
- Existing Tools are Difficult - Very Difficult to use
- Don’t do what you Need
  - Call Control
  - VPP
  - Protocol Support
  - Voice

Problem Analysis Conclusions (2 of 2)

- Tools are Difficult to integrate
- Tools controlling different test interfaces are difficult to Synchronize in Automatic Test Control Context
- Build your own

Agenda

- Test Environment
- Define Requirements
- Problem Analysis
- **Solution**
- Conclusions

Protocols / Voice

- Off-the-Shelf CO quality Hardware with Protocol based API Support
  - ISDN - PRI/Voice
  - SS7 Protocol Stack
    - MTP
    - SCCP
    - TCAP
    - ISUP
- TCAP State Machine
- GSM
- VPP - TCP/IP

Separate Test Design and Automation

Test Cases
(Action Words in a spreadsheet or db)

A	B	C
. . .		
dial in	4085551212	
check call attempt	Gateway	2971911
send call attempt ack	21	24
hang up		
. . .		

Functional (Designer)

Technical (Automator)

automation system (test tool)

```
case action
   " dial in": ...
   "check call attempt": ...
   "send call attempt ack":
..
   "hang up": …
end
```

Action Word Test Case Spreadsheet

	A	B	C	D	E
1	**cluster**	**TELCOM1**			
2	**sheet**	**Dial-up and Hang up**			
3	**version**	**2.2**			
4	**date**	**2/12/00**			
5	**author**	**DJ**			
6	**testcase**	**TELCOM1**			
7		**Expected Result: Prompts Played**			
9		*Number*			
10	**dial in**	4085551212			
11					
12		*target*	*calling party*		
13	**check call attempt**	**Gateway**	**2971911**		
14		*prompt ID*	*prompt ID*		
15	**Send call attempt ack**	**21**	**24**		
16					
17	**hang up**				
18					
19					
20					

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

Test Automation Control

- Test Engine
  - Heart of the Automated Test Process
  - Interprets Action Word Spread Sheets
    - Sequential Action Word Execution
    - Invokes Action Word Function
  - Built in Action Words
  - GUI Message Center
  - API support
    - Action Word Registration
    - State Control
    - Reporting/Report Generation
    - Test Flow
    - Checking
  - Reporting Capability

Troubleshooting Capability

- Engine Run-time Logs
- High level Test Results Report
- Function Trace Log
- Debug Log
- Engine Error Log

Run Time Log

0	20010322	152827	[ENGINEVERSION]	5.00		
1	20010322	152827	[CLUSTERNAME]	Example cluster		
2	20010322	152827	[CLUSTERVERSION]	1.2		
3	20010322	152827	[CLUSTERDATE]	June 28, 2000		
4	20010322	152827	[CLUSTERAUTHOR]	CMG Engine Team		
6	20010322	152827	[SCENARIO]	Try some basic report functionality		
•	20010322 3	152827	[ACTIONWORD]	print some numbers	1	2
9	20010322	152828	[COMMENT]	1		
9	20010322	152828	[COMMENT]	2		
9	20010322	152828	[COMMENT]	3		
12	20010322	152828	[ACTIONWORD]	perform a check	correct	wrong
12	20010322 [CHK_FAILED]	152829 [VERTICAL]	[CHECK]	expected message	correct	wrong
15	20010322	152829	[ACTIONWORD]	print an error	Just an error message	
15	20010322	152830	[ERROR]	[ERR_ERROR]	Just an error message	
18	20010322	152830	[SCENARIO]	Try report functionality combined with argument commands		

Run Time Log (cont)

21	20010322 -2.3	152830	[ACTIONWORD]	print some numbers	200	208	
21	20010322	152831	[COMMENT]	200			
21	20010322	152831	[COMMENT]	208			
21	20010322	152831	[COMMENT]	-2.3			
22	20010322	152831	[ACTIONWORD]	print some numbers	50	6.28e-005	
22	20010322	152832	[COMMENT]	50			
22	20010322	152832	[COMMENT]	6.28e-005			
22	20010322	152832	[COMMENT]	25			
25	20010322	152832	[COMMENT]	Cluster variable "Beta" with value "25" exported to keep file.			
25	20010322	152832	[COMMENT]	Cluster variable "Alpha" with value "200" exported to keep file.			
28	20010322	152832	[ACTIONWORD]	perform a check	correct	correct	
28	20010322	152833	[CHECK]	expected message	correct	correct	[CHK_PASSED] [VERTICAL]

Test Results Report

==========

SDT TestFrame-Test Execution Report

Licensed to : Greg Clower
Company : SDT
Serial number : 500-000057-1-01

Engine version : 5.00
Cluster name : Example cluster
Cluster version : 1.2
Cluster date : June 28, 2000

==========

Test Results Report (cont)

31 :	perform a check	&NotEmpty
	Check of	expected message
	Expected :	&NotEmpty
	Recorded :	
	Result :	failed
32 :	perform a check	&NotEmpty Some text
	Check of	expected message
	Expected :	&NotEmpty
	Recorded :	Some text
	Result :	passed

Test Results Report (cont)

```
==========================================================
Number of test lines          : 25

Succeeded test lines          : 64%

Number of errors
        Error                 : 4
Errors were at lines          : 15, 37, 65.8, 66.8

Number of checks              : 10
Passed checks                 : 5                50%
Failed checks                 : 5                50%
Fails were at lines           : 12, 30, 31, 65.11, 66.11

Start time                    : Fri Mar 23 16:09:45 2001
Stop time                     : Fri Mar 23 16:10:07 2001
Time used                     : 00:00:22
==========================================================
```

Debug Log Philosophy

- Document Function Entry
- Document Argument Values at Entry Time
- Delimit Argument Values with [ ]
- Make use of Default case in Switch Stmt
- When something goes wrong
  - Document everything

The Debug Log

```
E:\SDTEngine5\ReviewPro\Navigation\WinRunner\Debug\Greg_Clower.log :
                                                        Date Stamp : Tue Aug 29 15:00:13 2000
tf_OS_Setup        : ********** Start Up ********** :
tf_OS_Setup        : rOSFlag                    : Windows
tf_OS_Setup        : Running Mode is            : [debug]
tf_OS_Setup        : TestDirector Test Name     : [null]
tf_OS_Setup        : TfIni Path                 : E:\SDTEngine5\ReviewPro\Execution
                                                        \engine5.ini]
tf_OS_Setup        : Analysis Path              : [E:\SDTEngine5\SDTEngine\Analysis]
tf_OS_Setup        : Execution Path             : [E:\SDTEngine5\SDTEngine\Execution]
tf_OS_Setup        : Navigation Path            : [E:\SDTEngine5\SDTEngine\Navigation]
tf_OS_Setup        : ProjectAnalysis Path       : [E:\SDTEngine5\ReviewPro\Analysis]
tf_OS_Setup        : ProjectExecution Path      : [E:\SDTEngine5\ReviewPro\Execution]
tf_OS_Setup        : ProjectNavigation Path     : [E:\SDTEngine5\ReviewPro\Navigation]
tf_OS_Setup        : GuiPath Path               : [E:\SDTEngine5\SDTEngine\Navigation\
                                                        WinRunner\GUIs\]
tf_OS_Setup        : ProjectGuiPath Path        : [E:\SDTEngine5\ReviewPro\Navigation\
                                                        WinRunner\GUIs\]
tf_OS_Setup        : EnginePath Path            : [E:\SDTEngine5\SDTEngine\Navigation\
                                                        WinRunner\]
tf_OS_Setup        : BinPath Path               : [E:\SDTEngine5\SDTEngine\Execution\
                                                        bin\]
tf_OS_Setup        : LowLevelPath Path          : [E:\SDTEngine5\SDTEngine\Navigation\
                                                        WinRunner\LowLevels\]
```

The Debug Log (cont)

```
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\LowLevels\Declaration.low)]
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\LowLevels\GridHandling.low)]
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\LowLevels\ObjHandling.low)]
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\LowLevels\Standard.low)]
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\LowLevels\StringHandling.low)]
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\LowLevels\WebObjHandling.low)]
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\LowLevels\WinHandling.low)]
tf_OS_Call      : OS function [Reload]      : [reload (E:\SDTEngine5\SDTEngine\Navigation\
                                                WinRunner\HighLevels\Global.lib)]
```

The Debug Log (cont)

```
engine5          : ************ New Action Word ************** :
engine5          : Starting new action word          : Tue Aug 29 15:00:20 2000
engine5          : New action word name is           : Reviewpro Webshare Login
engine5          : ************************************** :
engine5          : Argument 2 description is         : [User Name]
engine5          : Argument 2 value is               : [iris rose]
engine5          : Argument 3 description is         : [Password]
engine5          : Argument 3 value is               : [CBK2468]
engine5          : Argument 4 description is         : [Push Button]
engine5          : Argument 4 value is               : [Login]
engine5          :                                   :
```


SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

The Debug Log (cont)

```
tf_CriticalChecks      : Window argument received        : [Webshare Login]
tf_ExpGuiWin           : Input Parameters                : [Webshare Login||]
tf_ExpGuiWin           : Assigning rSeconds              : [300]
tf_ExpGuiWin           : Window exists                   : Webshare Login
tf_ExpGuiWin           : Window activated                : Webshare Login
tf_GetBitmap           : Input Parameters                : [Webshare Login|E:\SDTEngine5\
                                                           ReviewPro\Analysis\Bitmaps\
                                                           ReviewPro|Webshare Login]
tf_CriticalChecks      : rWindow <= 0                    : [14]
tf_CalculateGrid       : Input Parameters                : []
tf_CheckParms          : Input Parameters                : [2|4|0|0]
tf_SetRoutine          : Input Parameters                : [2|4]
tf_SetRoutine          : FieldType[2]                    : [edit]
tf_SetRoutine          : FieldObject[2]                  : [User Name]
tf_SetRoutine          : ParameterValue[2]               : [iris rose]
tf_SetEdit             : Input Parameters                : [User Name|iris rose]
```


SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

Maintainability

- Highly Modular Architecture
- Layers of abstraction pushing details down
- Code Fragmentation & Isolation
- Minimize Change Propagation
  - Fix once, in one place
  - Fix available to all

Test Case 1

Functions	Occurrances	Present Maintenance Cost	After Function Maintenance Cost	Savings
TestCaseStartUp	1	$16.67	$16.67	$0.00
TestCaseLogin	1	$16.67	$16.67	$0.00
WriteStepResults	19	$316.67	$16.67	$300.00
SetObjectData	23	$383.33	$16.67	$366.67
TestCaseShutDown	1	$16.67	$16.67	$0.00
		$750.00	$83.33	$666.67

Test Case 1 (cont)

Functions	Occurrances	LOC Reduction	Gross Code Reduction	Starting LOC	Ending LOC	% Reduction
				699	358	48.78%
TestCaseStartUp	1	11				
TestCaseLogin	1	7				
WriteStepResults	19	266				
SetObjectData	23	46				
TestCaseShutDown	1	11				
			341			

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

Test Case 2

Functions	Occurrances	Present Maintenance Cost	After Function Maintenance Cost	Savings
TestCaseStartUp	1	$16.67	$0.00	$16.67
TestCaseLogin	1	$16.67	$0.00	$16.67
WriteStepResults	20	$333.33	$0.00	$333.33
SetObjectData	43	$716.67	$0.00	$716.67
PressBrowserOKButton	10	$166.67	$0.00	$166.67
TestCaseShutDown	1	$16.67	$0.00	$16.67
		$1,266.67	$0.00	$1,266.67

Test Case 2 (cont)

Functions	Occurrances	LOC Reduction	Gross Code Reduction	Starting LOC	Ending LOC	% Reduction
				1392	627	54.96%
TestCaseStartUp	1	11				
TestCaseLogin	1	7				
WriteStepResults	20	280				
SetObjectData	43	86				
PressBrowserOKButton	10	370				
TestCaseShutDown	1	11				
			765			

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

Test Case 3

Functions	Occurrances	Present Maintenance Cost	After Function Maintenance Cost	Savings
TestCaseStartUp	1	$16.67	$0.00	$16.67
TestCaseLogin	1	$16.67	$0.00	$16.67
WriteStepResults	29	$483.33	$0.00	$483.33
SetObjectData	45	$750.00	$0.00	$750.00
PressBrowserOKButton	5	$83.33	$0.00	$83.33
TestCaseShutDown	1	$16.67	$0.00	$16.67
		$1,366.67	$0.00	$1,366.67

Test Case 3 (cont)

Functions	Occurrances	LOC Reduction	Gross Code Reduction	Starting LOC	Ending LOC	% Reduction
				1546	836	45.92%
TestCaseStartUp	1	11				
TestCaseLogin	1	7				
WriteStepResults	29	406				
SetObjectData	45	90				
PressBrowserOKButton	5	185				
TestCaseShutDown	1	11				
			710			

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

The Bottom Line

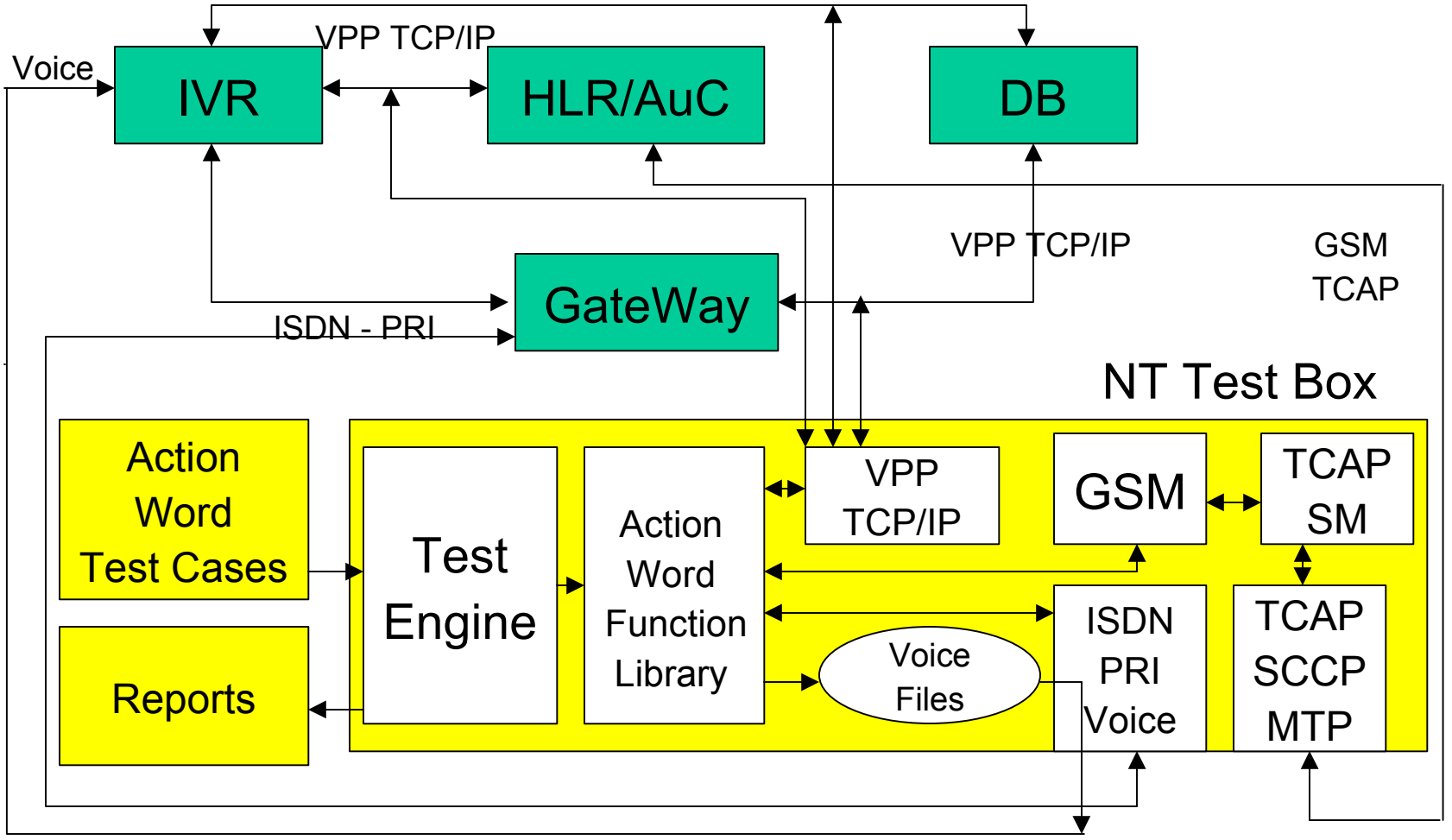
		Present Maintenance Cost	After Function Maintenance Cost	Savings
Cost for above 3 Cases		$3,383.33	$83.33	$3,300.00
Average per file		$1,127.78		$1,100.00
Total Cost		$16,916.67	$83.33	$16,833.33
Hours		169.17	0.83	99.51%
Days		21.15	0.10	99.51%
Weeks		4.23	0.03	99.38%
Months		1.06		

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

The Bottom Line (cont)

Gross Code Reduction	Starting LOC	Ending LOC	% Reduction
1816	3637	1821	49.93%

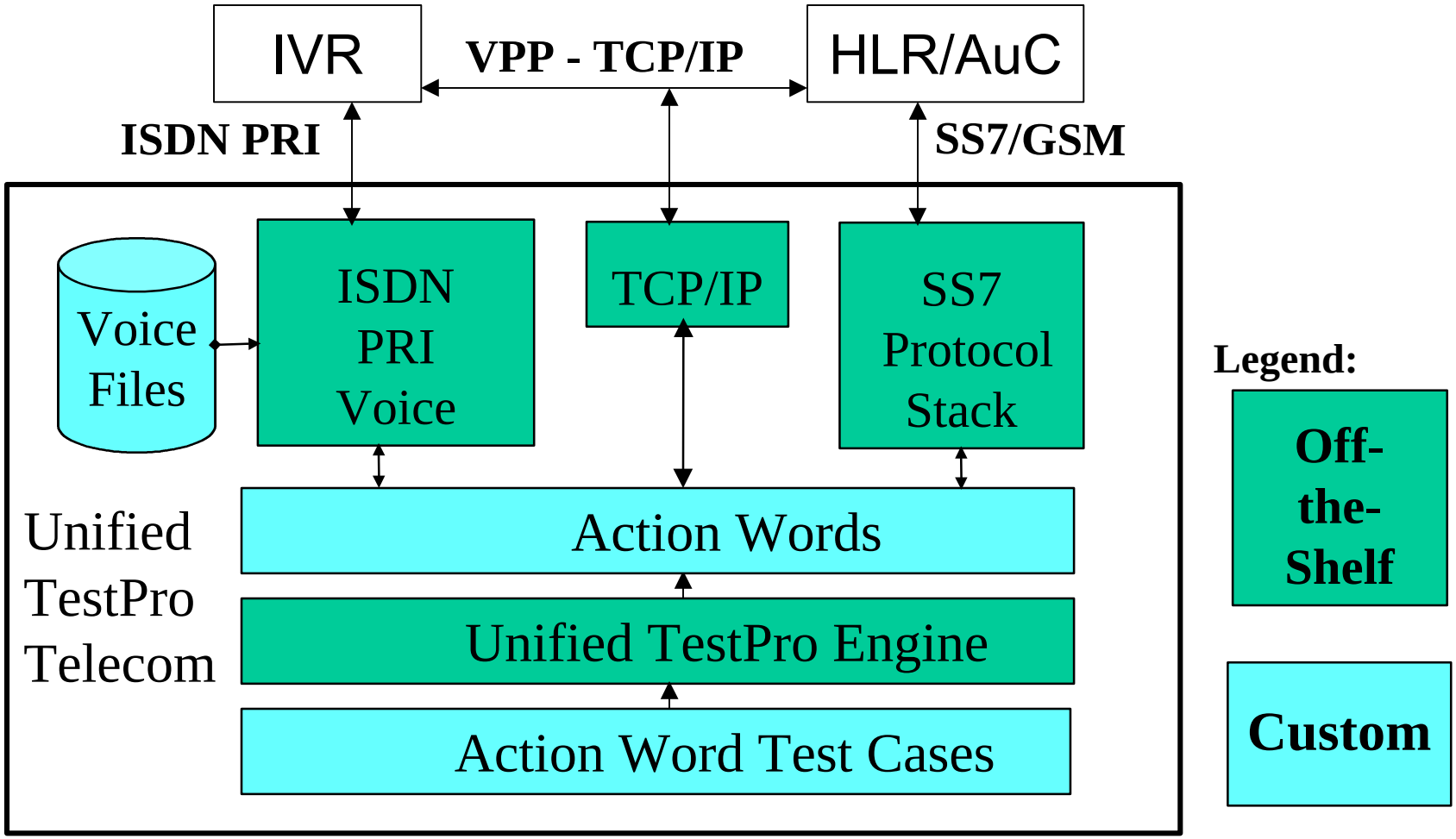
Building the Test Box



Agenda

- Test Environment
- Define Requirements
- Problem Analysis
- Solution
- Conclusions

Telecommunication
Unified TestPro Solution



Project Benefits

- Cost effective
- Software solution
- Off-the-shelf hardware
- Separation of Test Design and Execution
  - Enhanced Test Productivity
  - Schedule Reduction
- Absolute Interface Control
- Integrated Test solution
- Extensible

Summary

- This model works when you:
  - Do up-front Automation Architecture Design
  - Have correct Test Automation Control Technology
  - Treat Tool Development as an Engineering Project
    - Architected
    - Scheduled
  - Get the right level of Senior management support
  - Get the programming resources made available at the right time

Acronyms

- **ISUP** — **ISDN User Part**
- **MTP** — **Message Transfer Part**
- **SCCP** — **Signaling Connection and Control Part**
- **TCAP** — **Transaction Capabilities Application Part**
- **GSM** — **Global System for Mobile Communications**
- **INAP** — **Intelligent Network Application Part**
- **SMS** — **Short Message Service**
- **IS-41** — **Interim Specification 41 (aka ANSI-41)**
- **HLR** — **Home Location Register**
- **VLR** — **Visitor Location Register**
- **AuC** — **Authentication Center**
- **IVR** — **Interactive Voice Response**
- **ASN1** — **Abstract Syntax Notation 1**

References

- **Buwalda, Hans, *Testing with Action Words*, STAR May 1998**
- **Heine, Gunnar, *GSM Networks: Protocols, Terminology, and Implementation*, Artech House, IBSN 0-89006-471-7**
- **Kit, Edward, *Software Testing in the Real World*, Addison Wesley Longman, 1996**
- **Kit, Edward, *Integrated, Effective Test Design and Automation*, Software Development Magazine, February 1999**
- **Rose, Marshall T. , *The Open Book A Practical Perspective on OSI*, Prentice Hall, IBSN 0-13-643016-3**
- **SDT – Test Design and Automation: see Unified TestPro at http://www.sdtcorp.com/unifiedtestpro.pdf**
- **Steedman, Douglas, *Abstract Syntax Notation One (ASN1): The Tutorial & Reference*, Technology Appraisals, IBSN 1 871802 06 7**

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

The End

QW2001 QuickStart 7Q

Mr. James Bach
(Satisfice Inc)

High Accountability Exploratory Testing

Key Points

- How to have an orderly and defensible test process without pre-scripted test cases.
- How a session protocol can create the basis for a measurable process.
- Using metrics to track exploratory testing.
- How to explain a test strategy, after the fact.

Presentation Abstract

Exploratory testing means concurrent test design and test execution. Instead of writing tests down in advance, you make them up as you go. It's an ad hoc process. Experience shows that testers who use this approach find a lot of bugs quickly.

A problem with exploratory testing is that it's normally not as reviewable and, therefore, not as accountable as pre-planned testing. In this talk we will examine a way to get the benefits of exploratory testing while also providing high accountability for the test process. It's called session-based test management. Basically, testing activity is focused and packaged into time boxes called sessions. The sessions have a charter, reviewable output, and each session is debriefed by a test lead. Although what happens in each session is not determined in advance, we are able to record and measure testing productivity and test coverage in retrospect. We will talk about these metrics in some detail.

In our experience, we've found that the key to this approach is the debriefing, which is a coaching opportunity for the test lead while providing and opportunity for the tester practice explaining his test strategies.

About the Author

James Bach (http://www.satisfice.com) is founder and principal consultant of Satisfice, Inc. James cut his teeth as a programmer, tester, and SQA manager in Silicon Valley and the world of market-driven software development. He has worked at Apple, Borland, a couple of startups, and a couple of consulting companies.

Through his models of Good Enough quality, exploratory testing, and heuristic test

design, he focuses on helping individual software testers cope with the pressures of life in the trenches and answer the questions "What am I doing here? What should I do now?"

High Accountability Exploratory Testing

James Bach
Satisfice, Inc.
james@satisfice.com
http://www.satisfice.com

Exploratory testing relies on tester intuition. It is unscripted and improvisational.

But...

How do I, as test manager, understand what's happening, so I can direct the work and defend it to my clients?

SKILL
there's no shortcut

This is a black box…
Just like your mind.

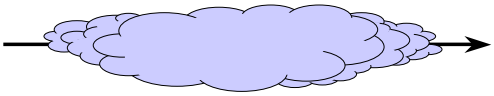
No one can read your mind.

You must gain the skill to explain your testing…
so that you can be accountable for it.

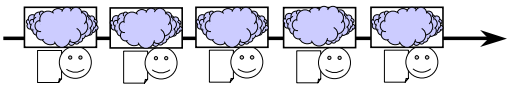
That requires a lot of practice.
In our experience: *several months* of daily practice.

Introducing the Test Session

1) Charter
2) Time Box
3) Reviewable Result
4) Debriefing



VS.



Charter:
A clear mission for the session

- A charter may suggest what should be tested, how it should be tested, and what problems to look for.
- A charter is not meant to be a detailed plan.
- General charters may be necessary at first:
  - *"Analyze the Insert Picture function"*
- Specific charters provide better focus, but take more effort to design:
  - *"Test clip art insertion. Focus on stress and flow techniques, and make sure to insert into a variety of documents. We're concerned about resource leaks or anything else that might degrade performance over time."*

Time Box:
Focused test effort of fixed duration

Short: 60 minutes (+-15)
Normal: 90 minutes (+-15)
Long: 120 minutes (+-15)

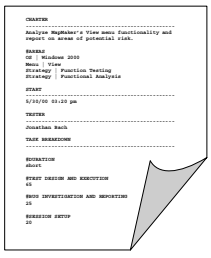
- *Brief enough for accurate reporting.*
- *Brief enough to allow flexible scheduling.*
- *Brief enough to allow course correction.*
- *Long enough to get solid testing done.*
- *Long enough for efficient debriefings.*
- *Beware of overly precise timing.*

Debriefing:
Measurement begins with observation

- The manager reviews ***session sheet*** to assure that he understands it and that it follows the protocol.
- The tester answers any questions.
- Session metrics are checked.
- Charter may be adjusted.
- Session may be extended.
- New sessions may be chartered.
- Coaching happens.

Reviewable Result:
A scannable session sheet

- Charter
  - *#AREAS*
- Start Time
- Tester Name(s)
- Breakdown
  - *#DURATION*
  - *#TEST DESIGN AND EXECUTION*
  - *#BUG INVESTIGATION AND REPORTING*
  - *#SESSION SETUP*
  - *#CHARTER/OPPORTUNITY*
- Data Files
- Test Notes
- Bugs
  - *#BUG*
- Issues
  - *#ISSUE*


CHARTER

Analyze MapMaker's View menu functionality and
report on areas of potential risk.

AREAS
OS | Windows 2000
Menu | View
Strategy | Function Testing
Strategy | Functional Analysis

START

5/30/00 03:20 pm

TESTER

Jonathan Bach

TASK BREAKDOWN

SESSION
about

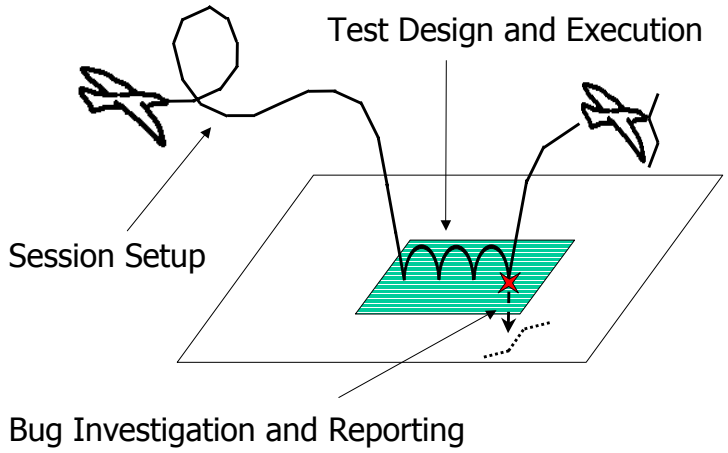
#TEST DESIGN AND EXECUTION
45

#BUG INVESTIGATION AND REPORTING
25

#SESSION SETUP
25

The Breakdown Metrics

Testing is like looking for worms



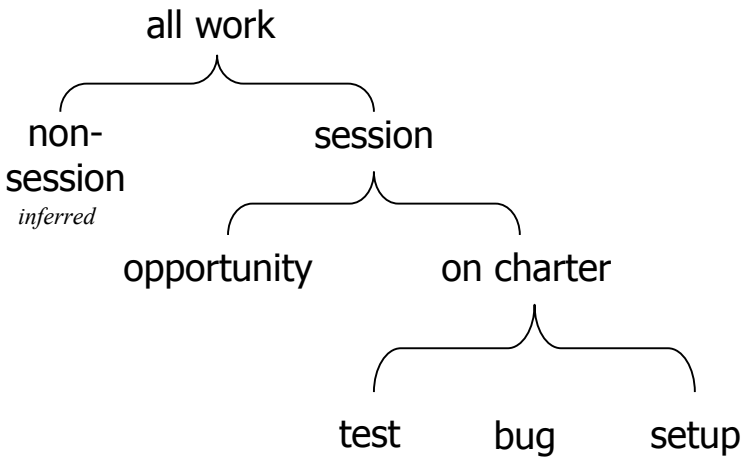
Reporting the TBS Breakdown

A guess is okay, but follow the protocol

- Test, Bug, and Setup are orthogonal categories.
- Estimate the percentage of charter work that fell into each category.
- Nearest 5% or 10% is good enough.
- If activities are done simultaneously, report the highest precedence activity.
- Precedence goes in order: T, B, then S.
- All we really want is to track interruptions to testing.
- Don't include Opportunity Testing in the estimate.

Activity Hierarchy

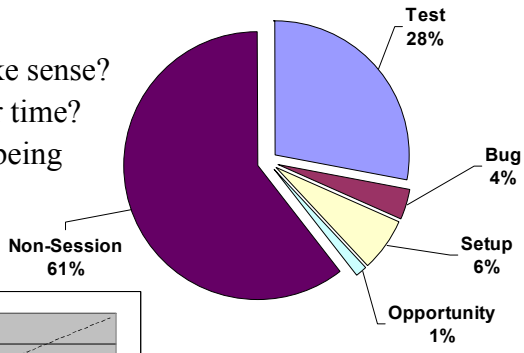
All test work fits here, somewhere

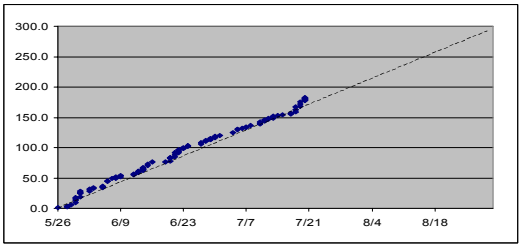


Work Breakdown:

Diagnosing the productivity

- Do these proportions make sense?
- How do they change over time?
- Is the reporting protocol being followed?





Coverage:
Specifying coverage areas

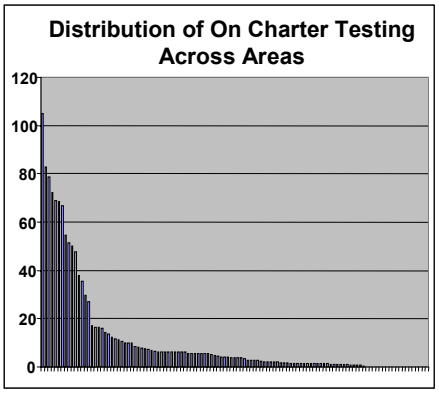
- These are text labels listed in the Charter section of the session sheet. (e.g. “insert picture”)
- Coverage areas can include anything
  - *areas of the product*
  - *test configuration*
  - *test strategies*
  - *system configuration parameters*
- Use the debriefings to check the validity of the specified coverage areas.

Coverage:
Are we testing the right stuff?

- Is this a risk-based test strategy?

or

- Is it a lop-sided set of coverage areas?
- Is it distorted reporting?



Using the Data to Estimate a Test Cycle

1. How many perfect sessions (100% on-charter testing) does it take to do a cycle? *(let's say 40)*
2. How many sessions can the team (of 4 testers) do per day? *(let's say 3 per day, per tester = 12)*
3. How productive are the sessions? *(let's say 66% is on-charter test design and execution)*
4. Estimate: 40 / (12 * .66) = **5 days**
5. We base the estimate on the data we've collected. When any conditions or assumptions behind this estimate change, we will update the estimate.

Challenges of High Accountability Exploratory Testing

- Architecting the system of charters (test planning)
- Making time for debriefings
- Getting the metrics right
- Creating good test notes
- Keeping the technique from dominating the testing
- Maintaining commitment to the approach

For example session sheets and metrics see
http://www.satisfice.com/sbtm

QW2001 QuickStart 8Q

Mr. Robert A. Sabourin
(AmiBug.Com)

The Effective SQA Manager: Getting Things Done

Key Points

- SQA Management
- Effective Process
- Process Improvement

Presentation Abstract

This interactive tutorial walks you through several "down to earth" practical aspects of running an SQA team.

The tutorial is presented in parable form. In this tutorial the audience will experience the real life problems encountered by a NOGO.COMs neophyte SQA Manager "Fred". "Fred" must turn around an enthusiastic but severely under staffed and under budget team of SQA professionals working in a chaotic development environment into a productive effective team! "Fred" is under the gun - he has to get things done!

About the Author

Robert Sabourin has been involved in all aspects of development, testing and management of software engineering projects. Robert graduated from McGill University in 1982. Since writing his first program in 1972, Robert has become an accomplished software engineering management expert. He is presently the President of AmiBug.Com, Inc.; a Montreal-based international firm specializing in software engineering and and software quality assurance training, management consulting and professional development. AmiBug helps companies set up software engineering and quality assurance teams and process through a combination of training and management consulting. Robert was the Director of Research and Development at Purkinje Inc where he was charged with developing world class critical medical software used by clinicians at the point of care. Previously, Robert managed Software Development at Alis Technologies for over ten years. He has built several successful software development teams and champions the implementation of "light effective process" to achieve excellence in delivering on-time, on-quality, on-budget commercial software solutions.

Robert has championed many complex international multilingual software

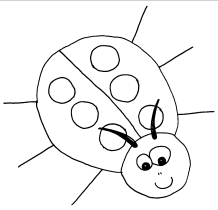
development and globalization efforts involving several intricate business partnerships and relationships including international government (Czech, Egypt, France, Morocco, Algeria...) and commercial entities (Microsoft, IBM, AT&T, HP, Thompson CSF, Olivetti...). Systems included concurrent coordinated multilingual multiplatform product releases.

Robert's pioneering work with Infolytica Corporation led to the development of the first commercially available platform independent graphics standard GKS and several toolkits which allowed for cross platform development and porting of complex CAD, Graphics, Analysis and Non-Destructive Simulation systems.

Robert is a frequent guest lecturer at McGill University where he relates theoretical aspects of Software Engineering to real world examples with practical hands-on demonstrations.

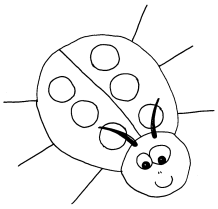
In 1999, Robert completed a short book illustrated by his daughter Catherine entitled"I Am a Bug" (ISBN 0-9685774-0-7).

Robert has received professional recognition for many accomplishments over the years. At TEPR 2000 - award for best electronic patient record product to EHS using the Purkinje CNC component. Byte Middle-East's 1992 Product of the Year for the AVT-710 product family achieving a ZERO FIELD REPORTED software defect rate with over 15,000 units installed. (Project involved over 27-man month's effort!); Quebec Order of Engineers' recognition for creating and managing the Alis R&D Policy Guide - Development Framework and process.



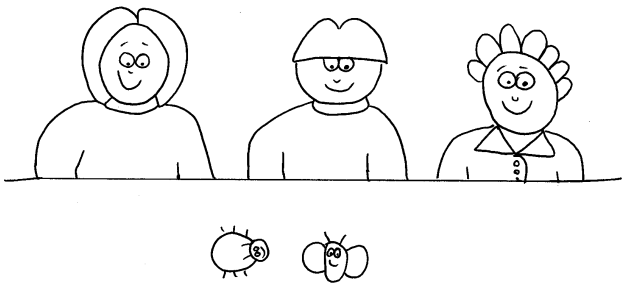
Becoming an Effective SQA Manager

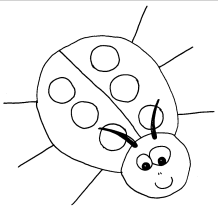
Robert Sabourin
President
AmiBug.Com, Inc.
Montreal, Canada
rsabourin@amibug.com



Becoming an Effective SQA Manager

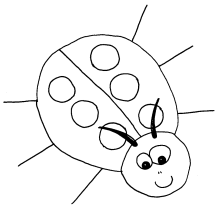
- It's all about people! (and the occasional bug too)





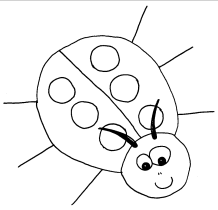
Becoming an Effective SQA Manager

- Overview
  - Introductions
  - Fundamental Question in Software Engineering!
  - Summary of key points from the *"Parable of the Effective SQA Manager"*



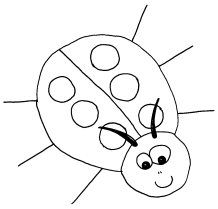
Becoming an Effective SQA Manager

- Robert Sabourin , *Software Evangelist*
- President
- AmiBug.Com Inc.
- Montreal, Quebec, Canada
- rsabourin@amibug.com



AmiBug.Com, Inc.

- Software Development & SQA Consulting
- Services
  - Training, Coaching and Professional Development
  - Light Effective Process
  - Team Building and Organization
  - We help people to get things done!



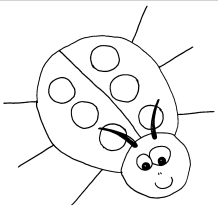
I am a Bug

Robert & Catherine Sabourin

ISBN: 0-9685774-0-7

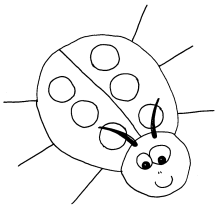
www.amazon.com

In the style of a children's book. Explains elements of software development process in a fun easy to read format.



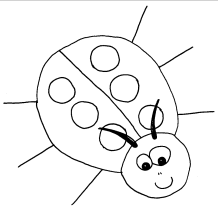
Fundamental Question

- How do you know when you are finished?

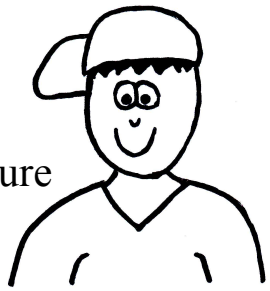


Definition of a Bug

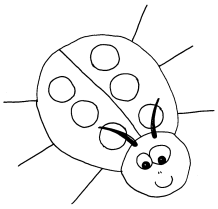
- To make our job more fun, whenever we have a *concern with software*, we call it a "bug".



Fred and NoGo.com

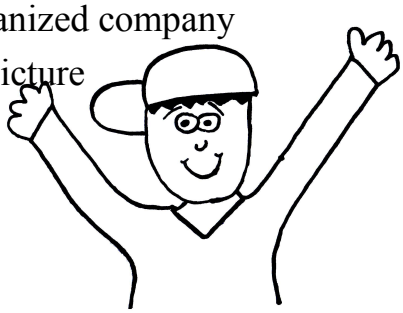


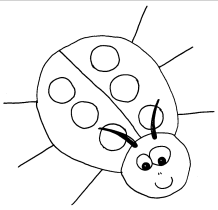
- Story about Fred
- Fred will have a simple adventure
- Learn many things



Fred

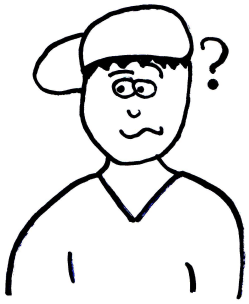
- Composite of many I have worked with
- Worked as a guru in software testing
  - Worked for a well organized company
  - Isolated from the big picture
  - Worked well with developers

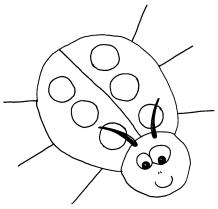




Fred

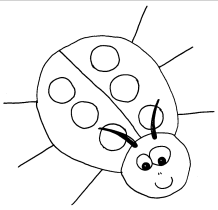
- But his company for some reason just didn't slice it
  - Ran out of funds
  - Could not sustain the pressure
  - Even with great testers like Fred





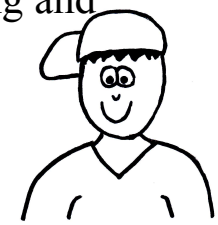
Fred's New Job

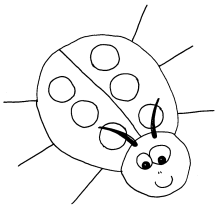
- Head hunters found Fred a new job in no time flat
  - Fred was hired as an SQA director at NoGo.Com
  - Fred was brought in to make things happen and *get things done*



NoGo.Com Evaluation Criteria

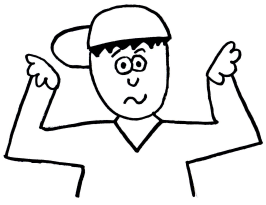
- Does he breathe?
- Does he get along with developers?
- Can he find serious, damaging and dangerous bugs?
- Available Now?

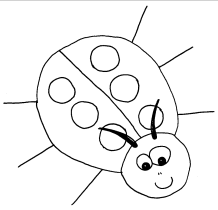




NoGo.Com Chaos

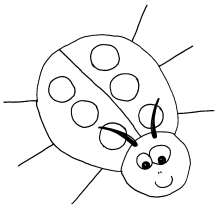
- Fred needed help
  - *Politics*
  - *Developers vs SQA*
  - *Prod Man vs Developers*
  - *Prod Man vs SQA*
  - *Test bottlenecks projects*





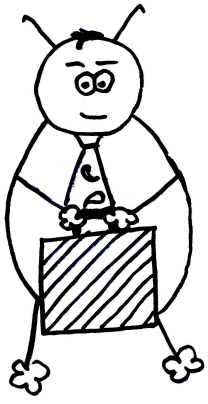
NoGo.Com Blame

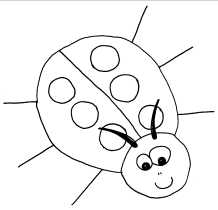
- Fred is squeezed!
  - *Scapegoat*
  - *Responsible*
  - *Must get things done*
  - *Needs Wisdom*
  - *Needs Sage Council*
  - *Needs a Mentor*



The E-SQA Manager

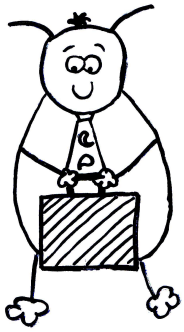
- Warm and welcoming
- Available
- Door is open
- People around office look busy
- Seems to have time

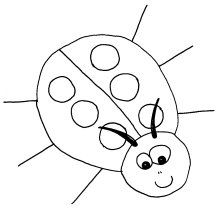




Welcome to Q - II

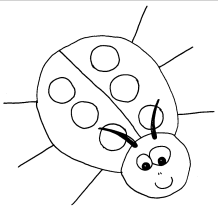
- E-SQA Manager said something that caught Fred off guard
- He said "Welcome to Quadrant II"
- And he shook my hand …
- what is going on?





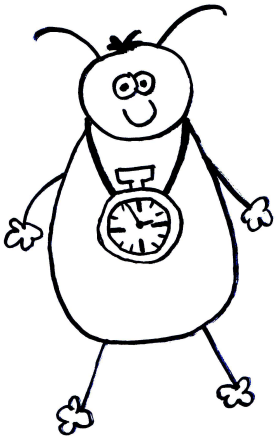
Q - II

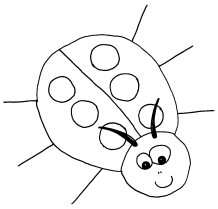
- E-SQA Manager explained Q - II
- Steve Covey "7 Habits of Highly Effective People"
- A new paradigm of time management



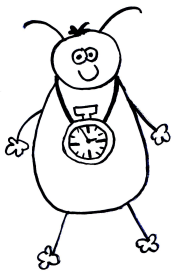
Quadrants

- What we do with our time?
- How do we use our time

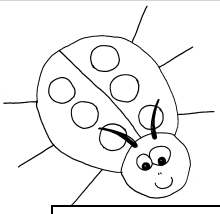




Quadrants

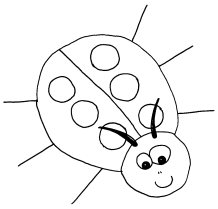


- Urgency
  - Things that require and demand our attention now
- Importance
  - Things that have significance, meaning and value



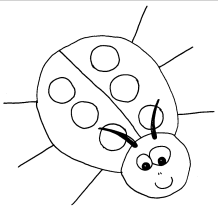
Time Management Matrix

Urgent Important	Not Urgent Important
Urgent Not Important	Not Urgent Not Important



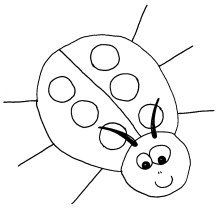
Four Quadrants

- QI
  - Urgent / Important
  - The pressing issue of the day that if it is not dealt with all other things become irrelevant!



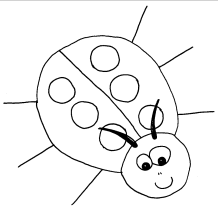
Four Quadrants

- QII
  - Not Urgent / Important
  - Long term issues which have significance and which improve things
  - Not pressing



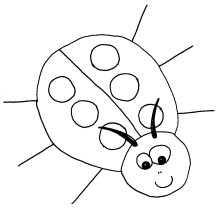
Four Quadrants

- QIII
  - Urgent / Not Important
  - Those unimportant activities which take your immediate attentions
  - Time stealers
  - Some phone calls or unimportant interruptions



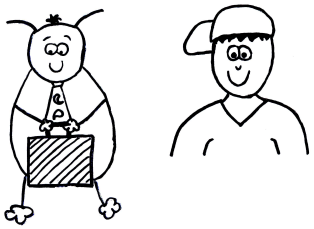
Four Quadrants

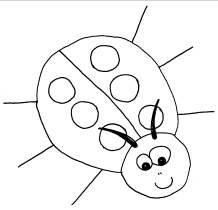
- QIV
  - Not Urgent / Not Important
  - Some wasteful mindless activities
  - Watching a mindless TV show
  - Reading a romance novel
  - Some unreasonably popular web sites



What about Vacations?

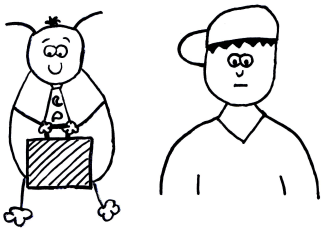
- Should be in QII
- Not QIV

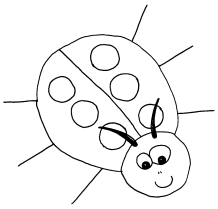




What about learning and teaching?

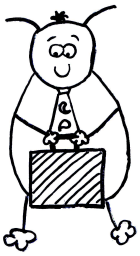
- Should be in QII

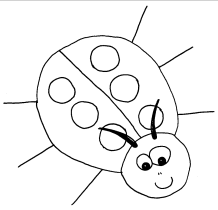




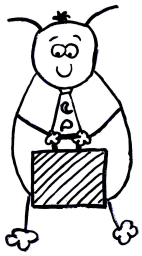
QII

- This is where you have to be in order to make a significant impact of your environment
- To make things better and have a lasting influence
- To be effective
- To really get things done!

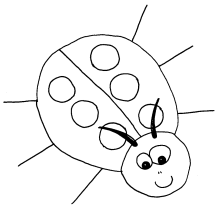




QII

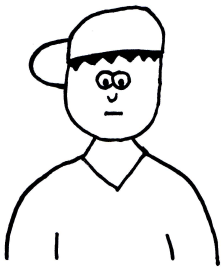


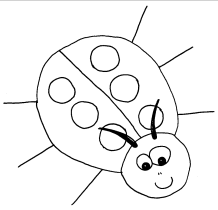
- By seeing the E-SQA manager to learn about how to become a more effective SQA manager Fred is essentially in QII
- This is not an urgent activity but obviously important



QII

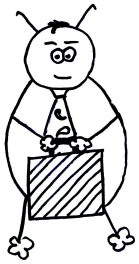
- You can force yourself into QII by doing activities such as
  - Retreats
  - Writing a diary
  - Taking time to get advice from other
  - Sharpen the saw

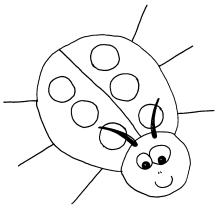




Inverted pyramid

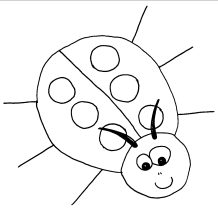
- The effective SQA manager facilitates
  - Makes it possible for the team to succeed
  - Makes it possible for individuals to succeed



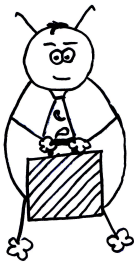


The 4 Ps

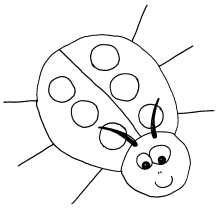
- E-SQA Manager explains that to get things done you and all of your team must understand the 4 Ps
  - Purpose
  - People
  - Practical Process



Purpose

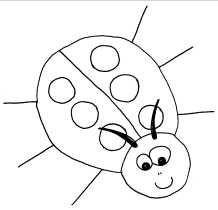


- The E-SQA Manager explains
- "At our organization we have a special focus on helping increase the value of our organization, we look at things from a business prospective always keeping in mind the key stakeholders, *our customers, employees and shareholders*"

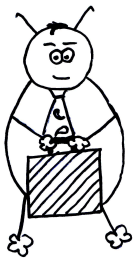


Purpose

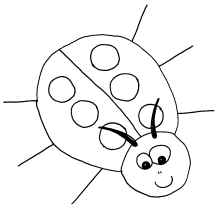
- "The role of SQA in our organization is to provide objective input to facilitate business decisions (wise smart and good decisions)"
- "SQA keeps internal stakeholders aware of all the issues that relate to shipping a product"
- Some friends of mine in Washington State have a similar purpose for the testing role!



Purpose

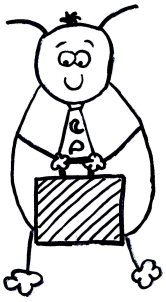


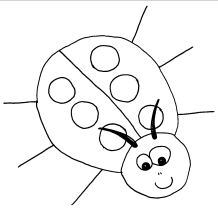
- To be an effective SQA manager you must be an "*on purpose*" SQA manager
  - *On Time*
  - *On Quality*
  - *On Budget*
  - Are meaningless unless you are *On Purpose*



Purpose of SQA Team

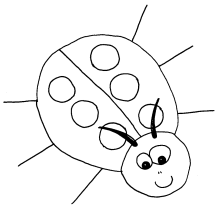
- The E-SQA Manager on Service Model
  - We have a service model for SQA generalized to Software Engineering
    - Metrics collection tracking as a service
    - Analysis as a service
    - Configuration management and construction as a service
    - Integration and System testing services
    - Formal inspection services



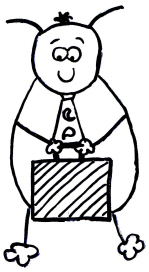


Purpose of SQA Team

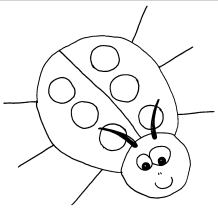
- On the E-SQA Service Model
  - We like to ensure that the customers of our service are Raving Fans!
  - A Raving Fan customer is a customer who is not just satisfied, but is so excited that they are like a walking, talking sales promotions department!
  - If you really want a booming business you need raving fans!



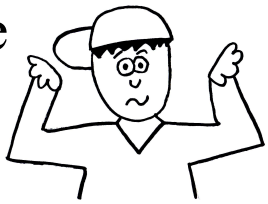
Getting to Raving Fan Service



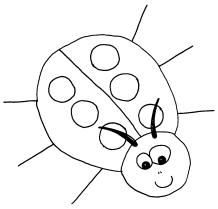
- Three Secrets to Raving Fan SQA
  - DECIDE WHAT YOU WANT
  - DISCOVER WHAT THE CUSTOMER WANTS
  - DELIVER PLUS ONE PERCENT



Getting to Raving Fan Service

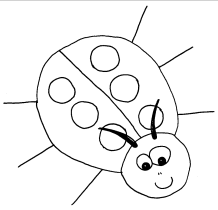


- DECIDE WHAT YOU WANT
  - Create a clear vision of what you want the SQA department to be like at some time in the ideal future
  - The vision should be of the internal customer using the services offered by the team
  - Have a good idea of what is excellence!



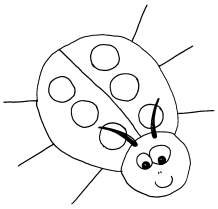
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - Identify who in the organization are your customers
    - Not just the leads and managers but all those touched by your service



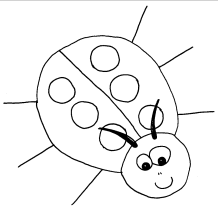
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - On an individual basis find out what they expect from your team, what type of products, services, information, data etc. (be polite and try to get specific not vague input)



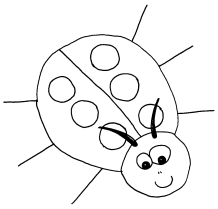
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - Developers want clear bug descriptions which help them find and correct the associated defects
  - Management wants the status of the product in terms of what works, what does not work, how close is the product to something which can he shipped!
  - Help Desk support people want good descriptions of a work around for all of the bugs we decide to leave in the product so that they can help end users



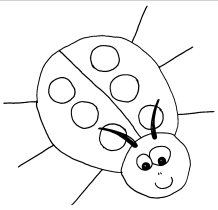
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - As required adapt your vision to mesh with that of the internal customers
    - If there are wide gaps and gaping holes consider redirecting the customer to some other department or organization (do not try to be all things for all people)



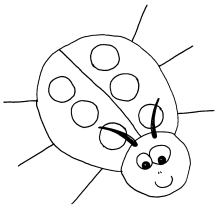
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - Sales may expect SQA to provide platform recommendations
    - In this case, you should redirect the customer to Product Management and make sure that it is clear that you do not provide that service



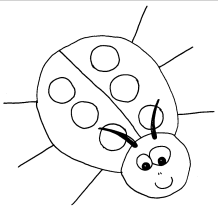
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - End-user support may expect SQA to provide a work around for bugs left in the product
    - If this is a service you intend to offer make sure it is part of your teams work on any project and take care to document in a clear straight forward manner any work around



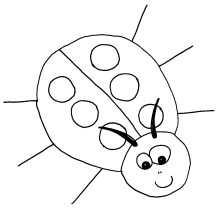
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - End-user support may expect SQA to provide a work around for bugs left in the product
    - Do we provide this in a language our end users should understand or in a language our customer service representatives understand!



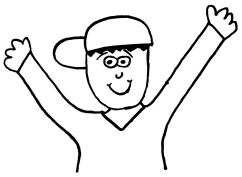
Getting to Raving Fan Service

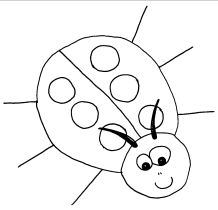
- DELIVER THE VISION PLUS ONE PERCENT
  - With a vision in hand establish a strategy which will allow you to deliver



Getting to Raving Fan Service

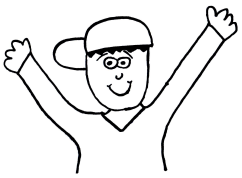
- DELIVER THE VISION PLUS ONE PERCENT
  - Baby steps are the order of the day!
    - We do not jump from the current state to the ideal vision in one step

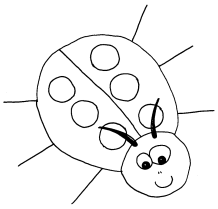




Getting to Raving Fan Service

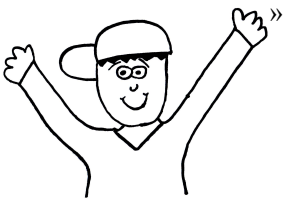
- DELIVER THE VISION PLUS ONE PERCENT
  - On each project implement some process change which brings you closer to the ideal
    - Consistency, consistency, consistency
    - Consistency creates credibility!

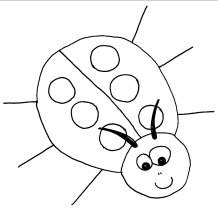




Getting to Raving Fan Service

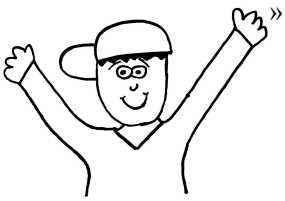
- DELIVER THE VISION PLUS ONE PERCENT
  - Inconsistency can destroy a lot of built up good will and productivity
    - » Bug reports descriptions varying depending on who wrote the report can make the whole team look incompetent even if it is only due to the fact that one junior tester was taking a great initiative to help out in an area he was not familiar with

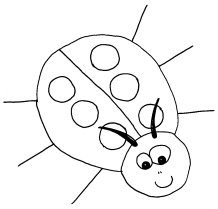




Getting to Raving Fan Service

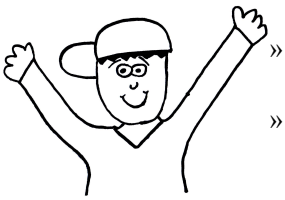
- DELIVER THE VISION PLUS ONE PERCENT
  - The way you treat one project should be the same way you treat all projects!
    - » Do not try to add all sorts of new process steps or deliverables until you can consistently deliver what is presently required

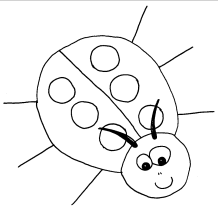




Getting to Raving Fan Service

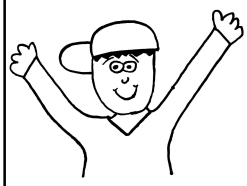
- DELIVER THE VISION PLUS ONE PERCENT
  - » Example BABY STEPS
  - » Bug Graph - update it once a week, consistently, accurate, punctual, available
  - » Bug Graph - update daily ONLY after weekly is working perfectly
  - » Bug Graph - on demand in real time ONLY after daily is working perfectly

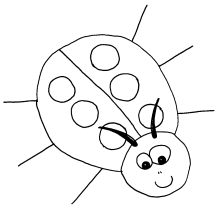




Getting to Raving Fan Service

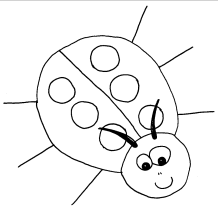
- DELIVER THE VISION PLUS ONE PERCENT
  - Meet first and then exceed customers expectation
    - If the customer expects a great test plan outline with coverage of all features, ensure this is consistently met before adding requirement tracing or usage scenarios





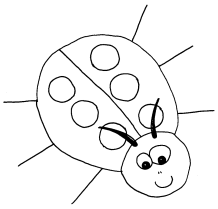
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - Figure out how to measure whether you are generating Raving Fan Internal Customers
    - Do they come back for more help, advise, guidance
    - Do they use the deliverables
    - Are they excited?
    - Are they having fun?



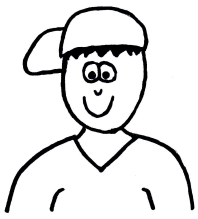
Getting to Raving Fan Service

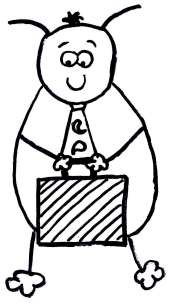
- Of course getting to Raving Fan Service is not a one man job
- The leader has to have the vision but the vision must be consistent with the purpose
- And then you have to get the people involved!

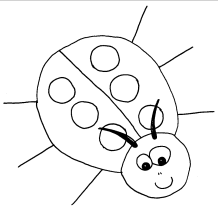


People

- The second P is “People”
- It is all about people!

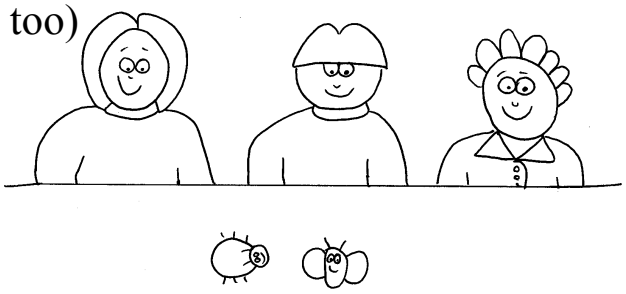


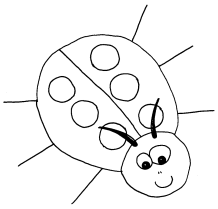




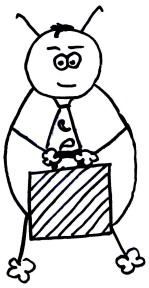
People

- It's all about people! (and the occasional bug too)

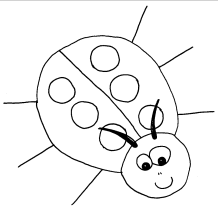




People

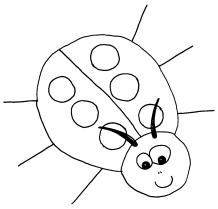


- E-SQA Manager:
  - It certainly makes a big difference if people are in SQA because they want to be in SQA rather than otherwise
  - With a smaller team of people who liked to work in SQA you can be more productive that with a larger team including staff who did not want to work in SQA

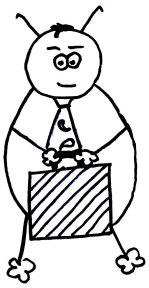


People

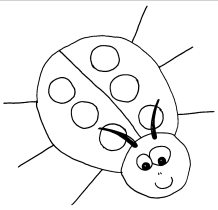
- If you want to get "Raving Fan Customers" for SQA you will need to have "Gung Ho" staff to deliver the service!
- You should read "Gung Ho!" also by Ken Blanchard and Sheldon Bowles which talks about how to increase Productivity, Profits and Prosperity by having a "Gung Ho!" team!
- People who are excited about going to work and being productive!



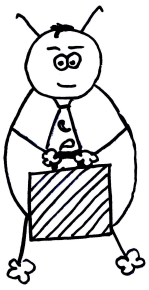
People



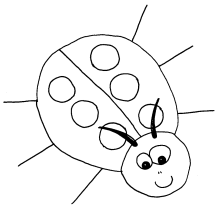
- The basics about "Gung Ho" staff
  - Worthwhile work
    - Important
    - Leading to shared goals
    - Value driven



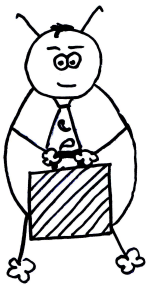
People



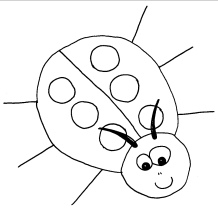
- The basics about "Gung Ho" staff
  - In control of achieving the goals
    - Well marked territory
    - Listen to and respect
    - Able but challenged



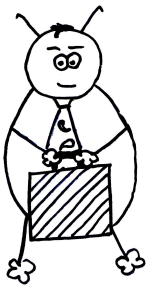
People



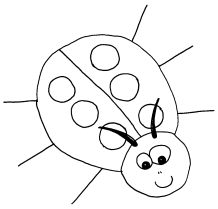
- The basics about "Gung Ho" staff
  - Cheering others on
    - Feedback timely and true
    - Keep score and cheer progress
    - Enthusiasm



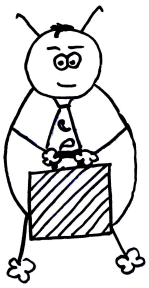
People



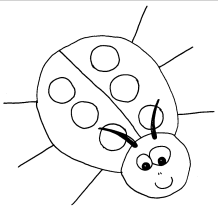
- E-SQA Manager
  - When I was starting out in SQA management I believed that "Happy people are productive"
  - I used to take the gang out for a beer or to the ball game
  - We partied and had a great team spirit



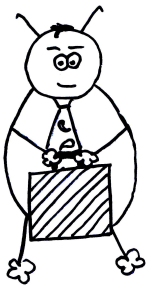
People



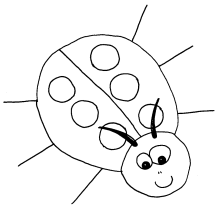
- E-SQA Manager
  - But it didn't make things work better at the office
  - In fact in some ways it was worse because people were more focused on the social extra curricular activities than on the job at hand!
  - Something was missing



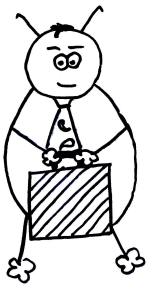
People

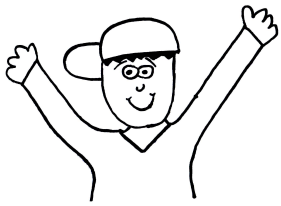


- E-SQA Manager
  - Then I learned that the more appropriate model was that "Productive people are happy"
  - If people have a clear important role, are allowed to succeed, and are given solid timely feedback
  - So now I focus my management efforts on my people and ensuring that they are and want to be productive!

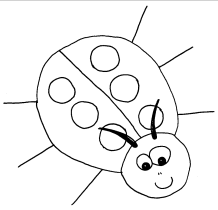


People

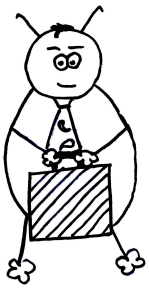




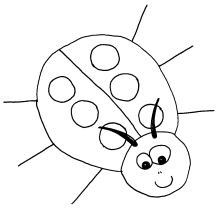
- E-SQA Manager
  - We still have parties to celebrate important achievements and project milestones and even sometimes to highlight individual success, but we are celebrating the productivity of people not celebrating to make people productive!



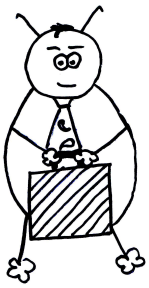
People



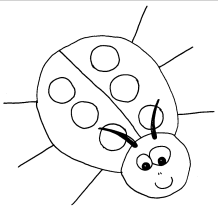
- E-SQA Manager
  - Feedback is the "Breakfast of Champions"
  - Remember feedback is about the behavior not the person
    - Next time we can do better by trying this instead of that



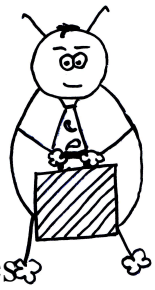
People



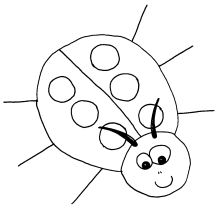
- E-SQA Manager offered some tips a managing people
  - Different Strokes for different folks at different times!
  - Adapt leadership style to the situation
  - Choose leadership styles deliberately!



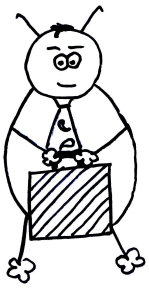
People



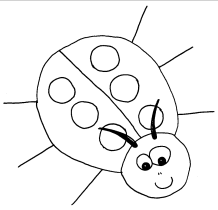
- 4 Basic Situational Leadership Styles
  - Directing
  - Coaching
  - Supporting
  - Delegating



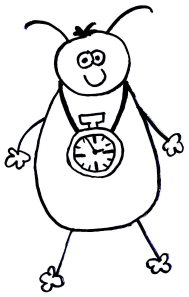
People



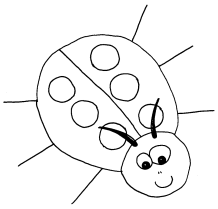
- Directing Leadership
  - Tell people specifically what to do
  - Provide constant feedback, praising and redirection
  - Used when someone is new to a task and uncertain as to how to successfully achieve the task
  - Used sometimes in an emergency situation



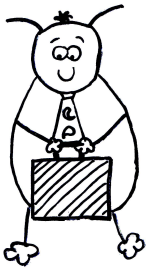
People



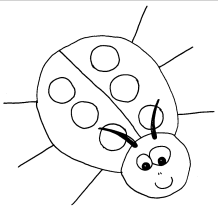
- Coaching Leadership
  - Provide guidance and advice on how to achieve goals based on input from staff
  - Does not need close direction but needs to learn how to achieve success
  - Used when someone has a proven track record but is new to this specific task
  - Team member is mature enough to ask for assistance



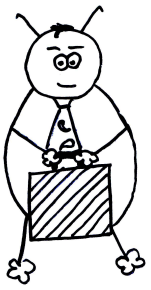
People



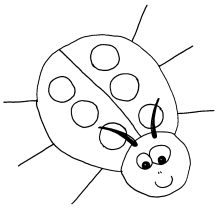
- Supporting Leadership
  - Staff participate in decision making with leader
  - Staff works with leader to establish goals and milestones
  - Person can work quite autonomously but needs leaders help
  - Team member is mature



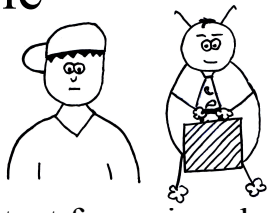
People



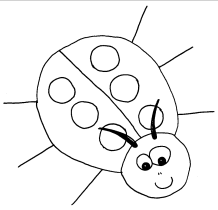
- Delegating Leadership
  - Staff is given broad goal and parameters and then takes full ownership of task
  - Constant feedback is not required and the need for it is driven by team member mostly to confirm that big picture business drivers have not changed
  - Team member is capable of successfully achieving assignment
  - Team member is autonomous



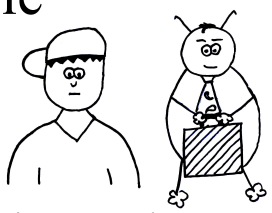
People



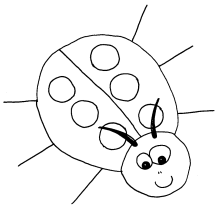
- SQA Bill Of Rights
  - Right to know the business context for assigned activities. Staff must be able to answer the question: "What is the business reason for doing this assigned activity?"
  - Right to know what it means to finish assigned activity



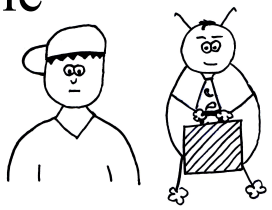
People



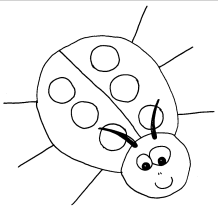
- SQA Bill Of Rights
  - Right to know what software being tested is supposed to do and if assumptions are to be made the right to double check with product or development management before testing activity starts
  - Right to get software which the development team honestly believes works



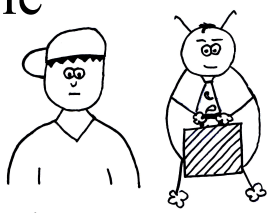
People



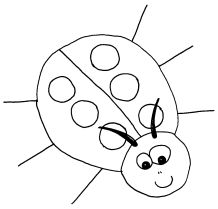
- SQA Bill Of Rights
  - Right to have fun at work
  - Right to learn new work methods, techniques and technologies
  - Right to try out innovations which may fail



People

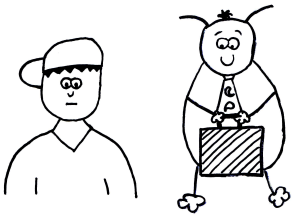


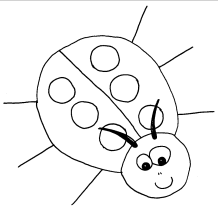
- SQA Bill Of Rights
  - Right to speak directly with developer responsible for code being tested
  - Right to report a bug discovered even if it may already be in the bug list (never loose a bug)
  - Right to know how much effort to spread across a testing assignment



People Earn Respect

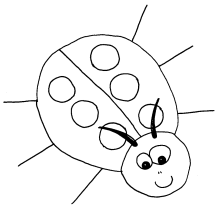
- You must earn respect
- From peers
- From customers
- From stakeholders





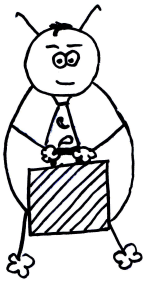
People Human Side

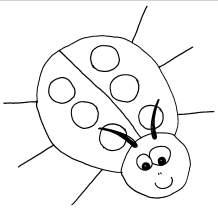
- The human side of the equation is the most unpredictable
  - There is always something you do not know
  - Problems at home
  - Peer personality conflicts
  - Poor self-esteem of team members
  - Be sensitive, clear, firm and honest



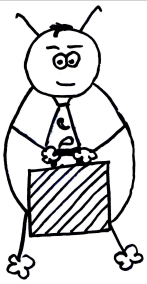
Practical Process

- Baby Step Innovation
  - On each project implement two innovations
    - Technical or technology innovation
    - Process or management innovation

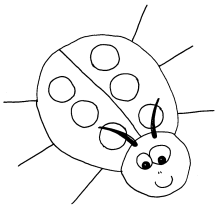




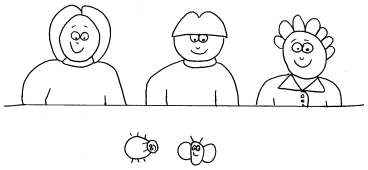
Practical Process



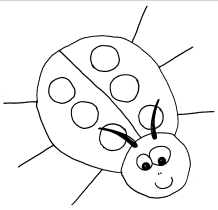
- Baby Step Innovation
  - Ensure that all projects operate within a couple of innovations from each other
  - Every project is a pilot project for some innovation
  - Pull innovation if it does not look promising after being given a fair chance



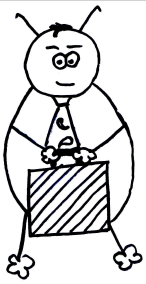
Practical Process



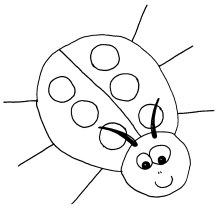
- Have effective meetings
  - As few people as possible
  - Efficient use of time
  - Separate project meetings from team meetings
  - Team meetings invited guests, info from exec



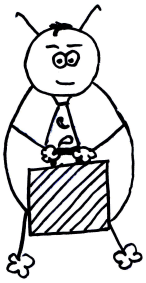
Practical Process



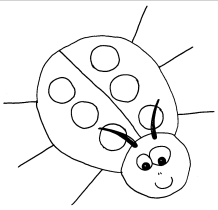
- Be punctual at all times
  - An SQA Manager must set an example
  - Be on time in all matters at all times
  - If you expect your people to deliver on time you must deliver on time
  - Make sure administrative issues, pay issues and all people issues are dealt on time



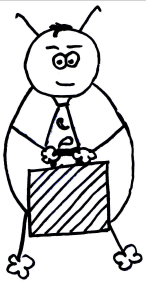
Practical Process



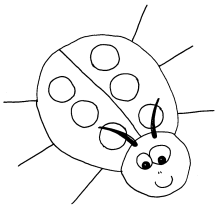
- It is really quite simple
  - All you have to do is always MAKE AND KEEP COMMITMENTS!



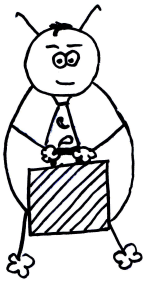
Practical Process



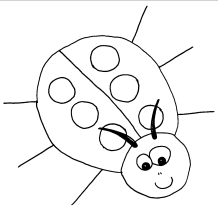
- Off Site SQA Team Retreats
  - Focus on what can be changed
  - Look at past recent experience
  - All team members come prepared
  - Capture results and recommendations



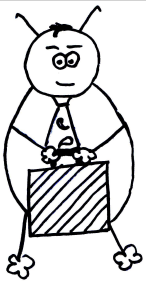
Practical Process



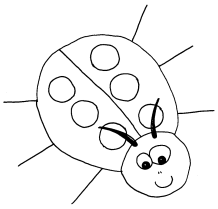
- Project Post Mortem Review
  - Key team members bring lists
    - 5 excellent things to be encouraged in future projects
    - 5 things that could have been done better and should be improved in future projects
    - A couple of specific recommendations or personal comments



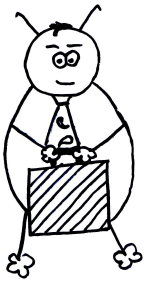
Practical Process



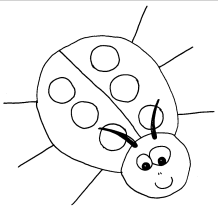
- Project Budgets
  - Need to know how many resources to commit to project
  - Effort based and spread across project in rational way
  - Reviewed and revised frequently with project stake holders



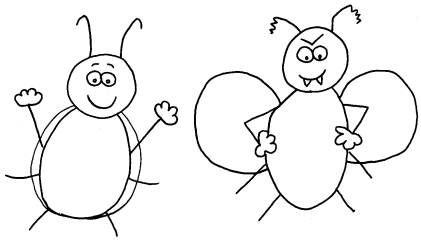
Practical Process



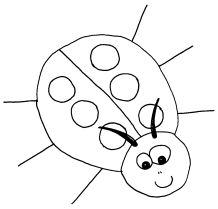
- Staffing
  - Have access to contract resources to increase capacity for short bursts during crunch periods
  - Good test scripts and plans help
  - Have permanent staff coach contract staff for leverage



About Bugs

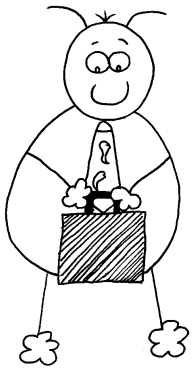


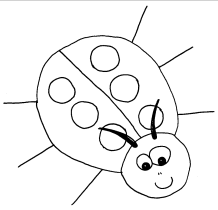
Bugs are not Good or Bad



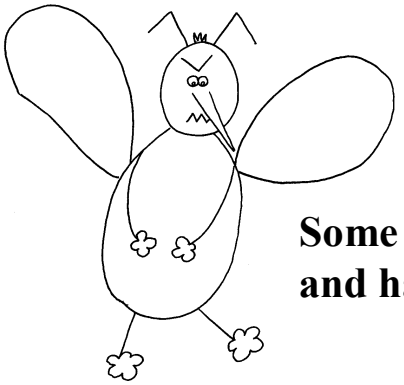
About Bugs

Some bugs are important and have a high priority!

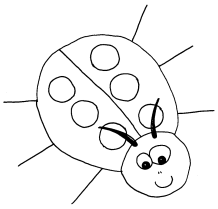




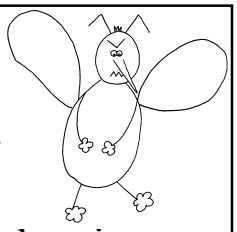
About Bugs



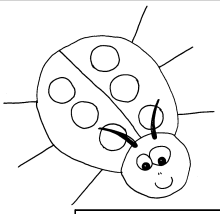
Some bugs are dangerous and have a high severity!



About Bugs

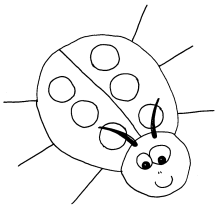


- Setting the priority and severity of a bug is a business decision
- Changing business conditions impact the priority and severity of a bug!
  - Always review previous decisions in light of changing business context
  - Ensure staff assigning priority and severity are aware of all relevant business drivers

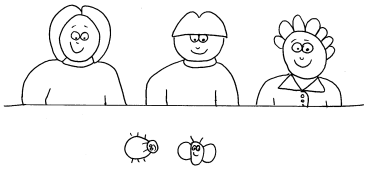


Bug Quadrants

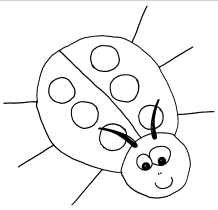
Urgent Severe	Urgent Not Severe
Not Urgent Severe	Not Urgent Not Severe



Business Decisions

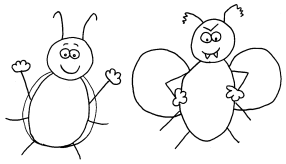


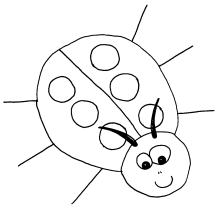
- SQA:
  - Objective input
- Development:
  - Technical implementation
- Product Management:
  - Customer driven requirements



Quadrant Changing

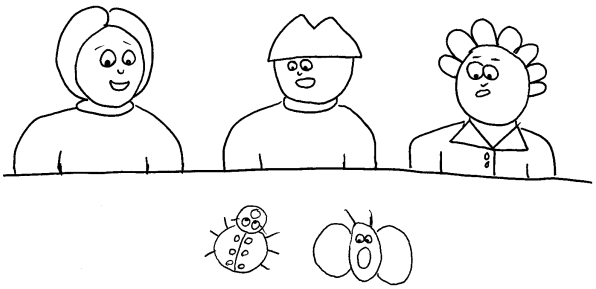
- Same technical bug can be in a different quadrant depending on the business context
- Monitor business drivers!
- Focus find and fix quadrant -1- bugs high priority/high severity

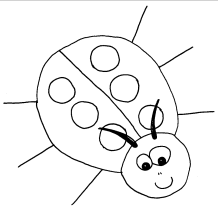




Finished?

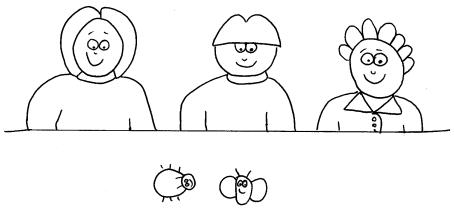
- How do you know you are finished?

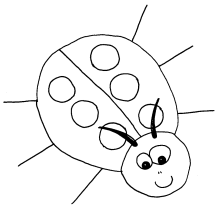




You know you are finished when …

- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) ...

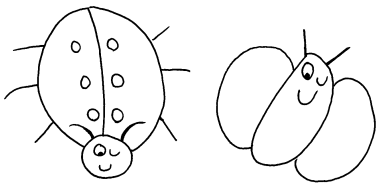


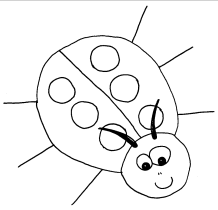


You know you are finished when …

- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) …

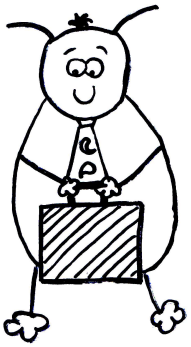
At least for now!

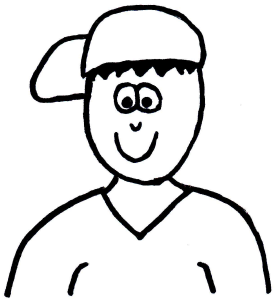




Thank You

- Questions?





QW2001 Special Panel Session 9Q

Dr. John D. Musa - Panel Chair
(Consultant)

How Will The Internet Affect Software Quality Practice? (Panel Discussion)

Presentation Abstract

The Internet is having dramatic effects in all areas of software development. This panel will focus on how the Internet will affect the practice of software quality. The panelists were selected to represent different backgrounds. They will outline some of their visions on the problems that will challenge us and offer some possible solutions to those problems. The panel will invite the audience to enrich the session by contributing their questions, comments, and views, to which the panelists will respond as appropriate.

About the Panel Moderator

John D. Musa is one of the creators of software reliability engineering , with more than 30 years varied and extensive experience as a software development practitioner and manager. Principal author of the highly-acclaimed pioneering book Software Reliability and author of the practical Software Reliability Engineering, Musa has published more than 100 papers on SRE. Elected IEEE Fellow in 1986 for many seminal contributions, he was recognized in 1992 as the leading contributor to testing technology. His leadership has been noted by every recent edition of Who's Who in America and American Men and Women of Science. Musa, widely recognized as a leader in SRE practice, initiated and led the effort that convinced AT&T to make SRE a "Best Current Practice." Musa has helped a wide variety of companies with a great diversity of software-based products deploy SRE. He is an experienced international speaker and teacher (over 200 major presentations) A founder of the IEEE Technical Committee on SRE, he is closely networked with SRE leaders, providing a broad perspective.

Panel Members

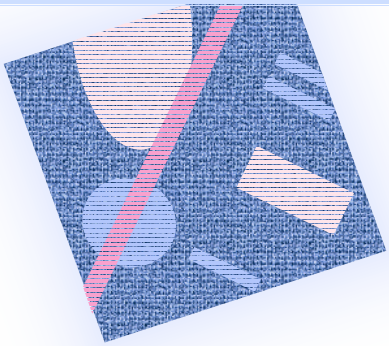
James Bach heads up Satisfice, a software testing consulting firm with a world class test lab located in rural Northern Virginia. James has extensive experience in a variety of testing situations, including for Silicon Valley startups, and larger organizations such as Microsoft, Borland, and Apple Computers.

Dr. Edward Miller is President of Software Research, Inc., San Francisco, California, where he has been involved with software test tools development and software engineering quality questions. Dr. Miller has worked in the software quality management field for 25 years in a variety of capacities, and has been involved in the development of families of automated software and analysis

support tools. He was chairman of the 1985 1st International Conference on Computer Workstations, and has participated in IEEE conference organizing activities for many years. He is the author of Software Testing and Validation Techniques, an IEEE Computer Society Press tutorial text. Dr. Miller received his Ph.D. (Electrical Engineering) degree from the University of Maryland, an M.S. (Applied Mathematics) degree from the University of Colorado, and a BSEE from Iowa State University.

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people. Johanna publishes "Reflections", an acclaimed quarterly newsletter about managing product development. Johanna's handbook, "Hiring Technical People: A Guide to Hiring the Right People for the Job," has proved a boon to perplexed managers, as have her articles in Software Development, Cutter IT, IEEE Computer, Software Testing and Quality Engineering, and IEEE Software. Johanna is the founder and principal of Rothman Consulting Group, Inc., and is a member of the clinical faculty of The Gordon Institute at Tufts University, a practical management degree program for engineers.

How Will the Internet Affect Software Quality Practice?



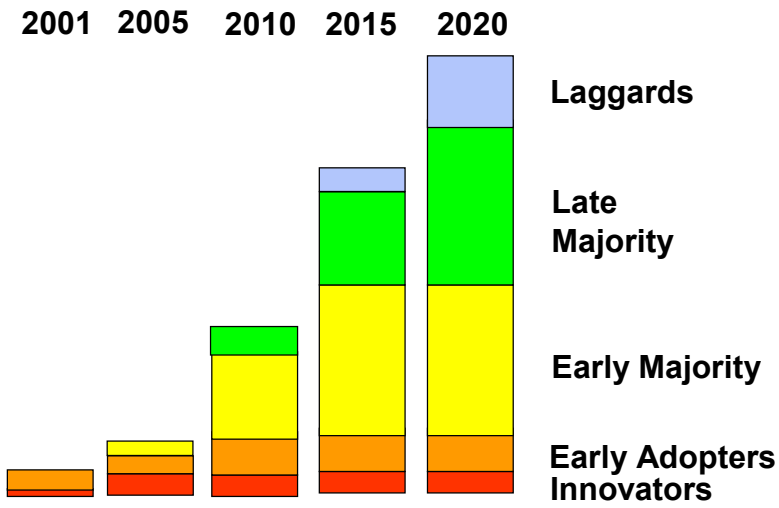
John D. Musa
j.musa@ieee.org

Classes of Internet Users

- Innovators: Risk-taking style leaders who will filter out products that early adopters will look at
- Early Adopters: Progressive solid-citizen leaders who will take some risks with products demonstrated promising
- Early Majority: Followers who will use a product all their cohorts are using
- Late Majority: Reluctant followers who will use a product when alternatives are clearly undesirable
- Laggards: Resistors who will use a product only when alternatives are no longer available

2

Time Profile of Internet Users

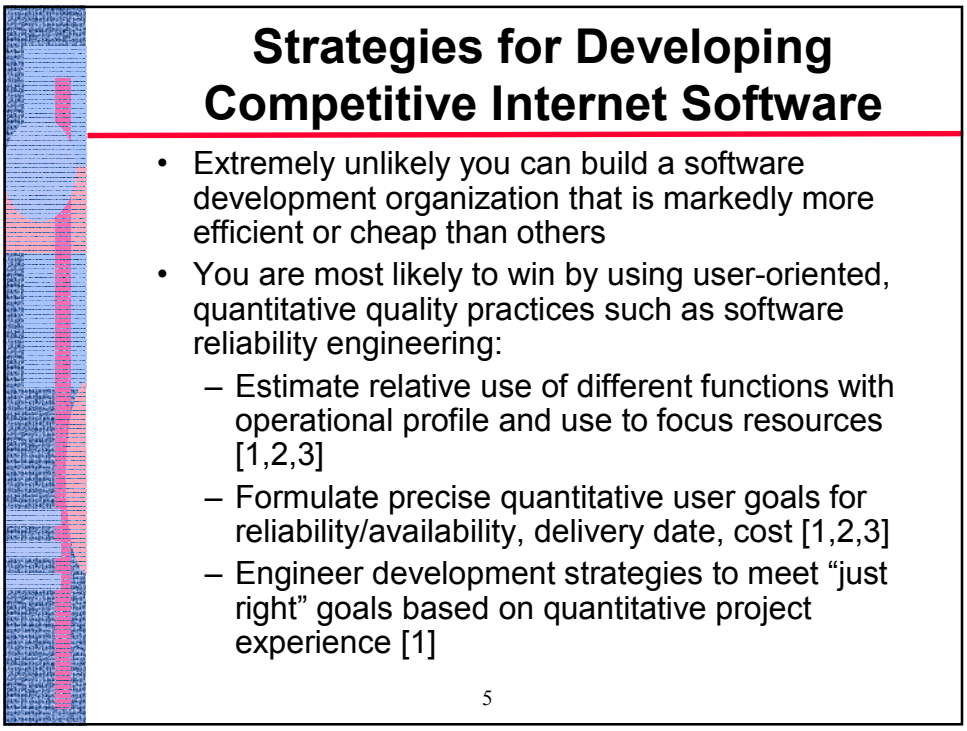


3

Relative Quality Expectations of Internet Users

User Class	*Rel/Avail*	*Cost*	*Time to Market*
Innovators	Sub. Lower	Sub. Higher	First
Early Adopters	Lower	Higher	Earlier
Early Majority	Competitive	Competitive	Same
Late Majority	Higher	Lower	Later
Laggards	Much Higher	Much Lower	Much Later

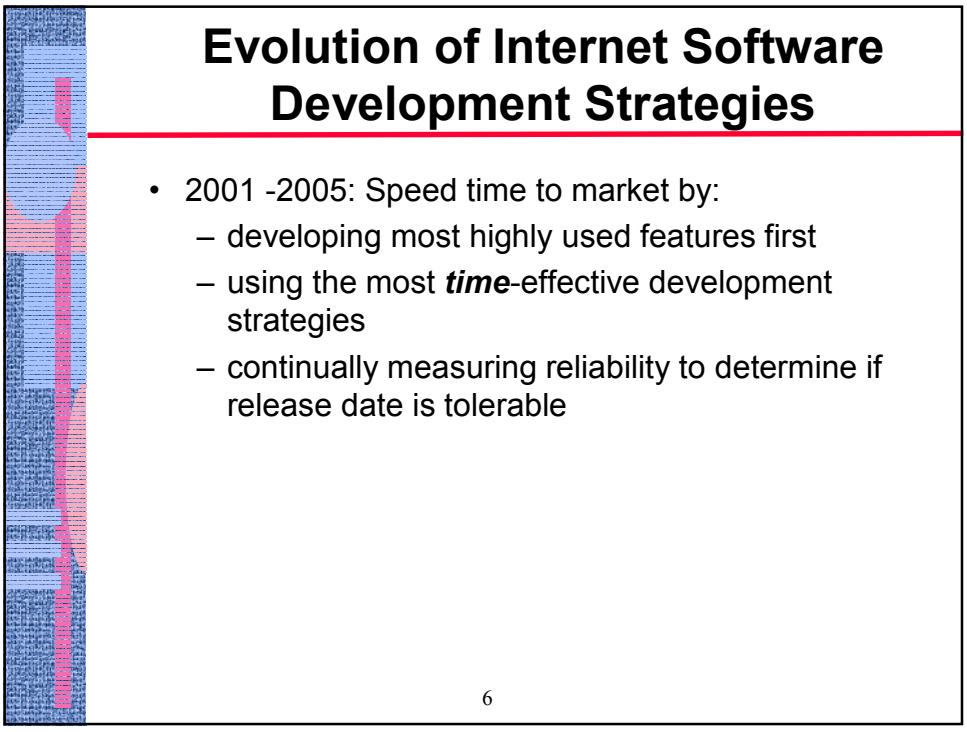
4



Strategies for Developing Competitive Internet Software

- Extremely unlikely you can build a software development organization that is markedly more efficient or cheap than others
- You are most likely to win by using user-oriented, quantitative quality practices such as software reliability engineering:
  - Estimate relative use of different functions with operational profile and use to focus resources [1,2,3]
  - Formulate precise quantitative user goals for reliability/availability, delivery date, cost [1,2,3]
  - Engineer development strategies to meet "just right" goals based on quantitative project experience [1]

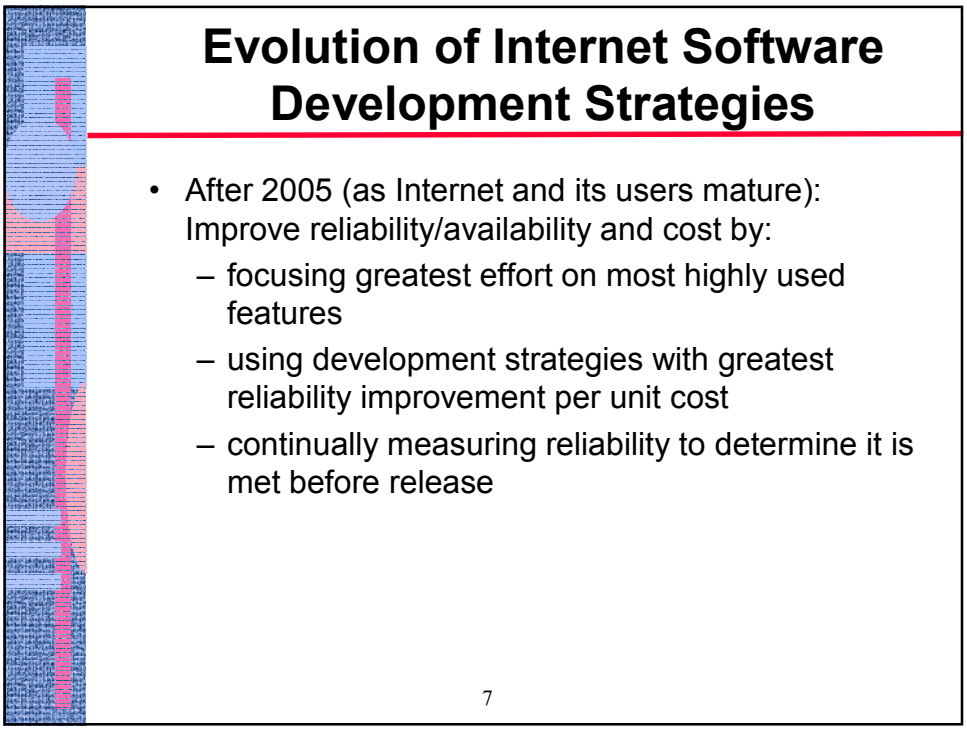
5

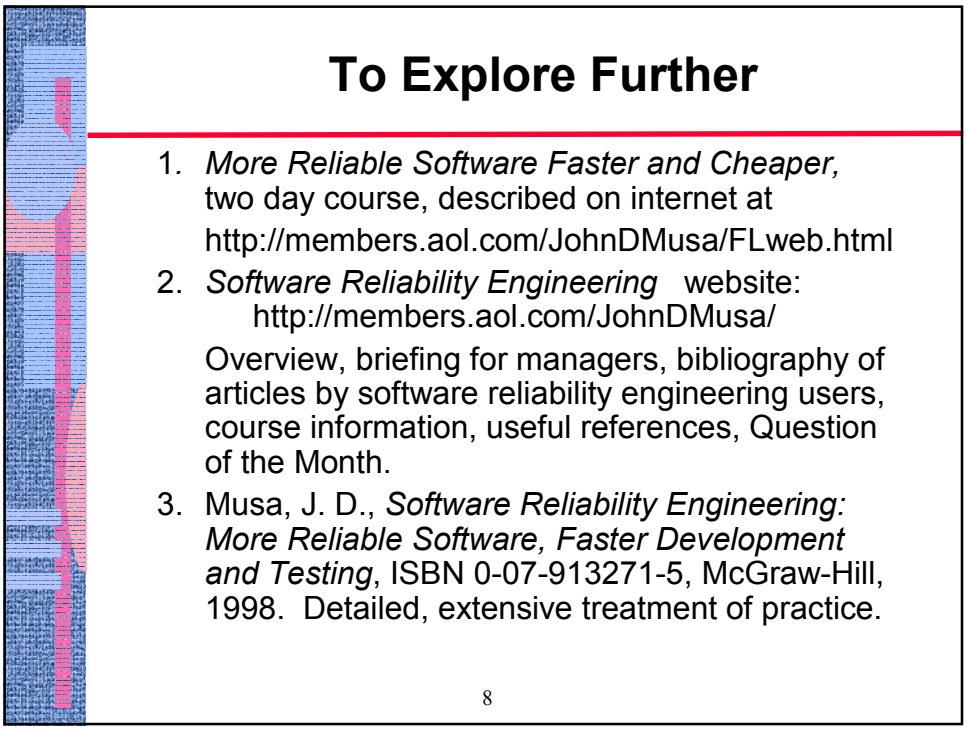


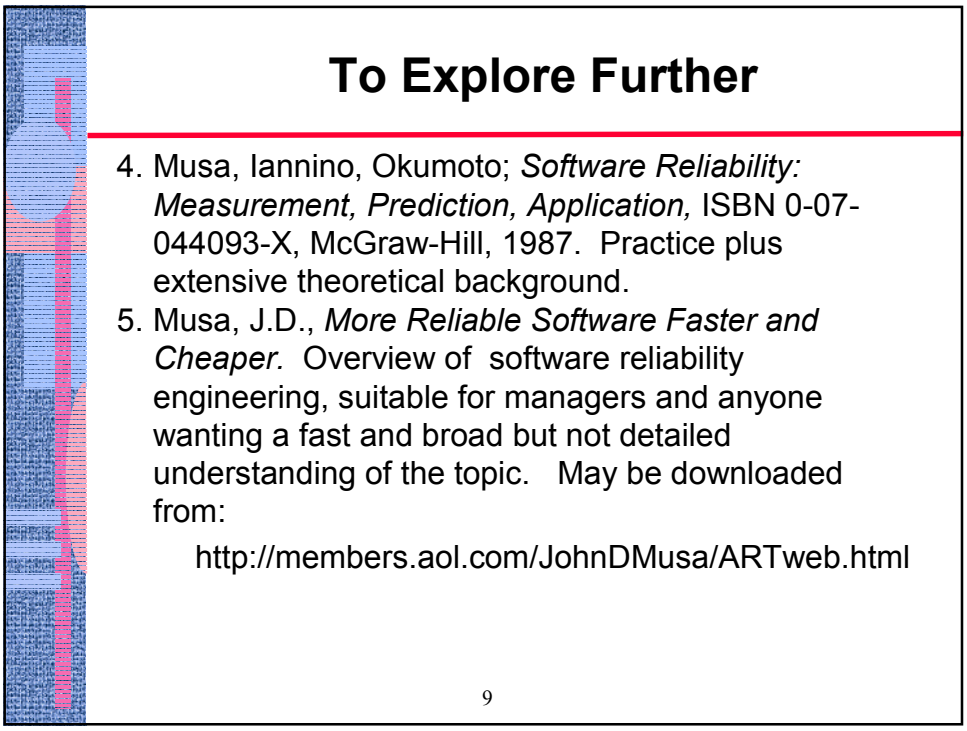
Evolution of Internet Software Development Strategies

- 2001 -2005: Speed time to market by:
  - developing most highly used features first
  - using the most ***time***-effective development strategies
  - continually measuring reliability to determine if release date is tolerable

6







To Explore Further

4. Musa, Iannino, Okumoto; *Software Reliability: Measurement, Prediction, Application,* ISBN 0-07-044093-X, McGraw-Hill, 1987. Practice plus extensive theoretical background.
5. Musa, J.D., *More Reliable Software Faster and Cheaper.* Overview of software reliability engineering, suitable for managers and anyone wanting a fast and broad but not detailed understanding of the topic. May be downloaded from:

   http://members.aol.com/JohnDMusa/ARTweb.html

9

How Will The Internet Affect Software Quality Practices?

Johanna Rothman
Rothman Consulting Group, Inc.
781-641-4046
jr@jrothman.com
www.jrothman.com

Two Ways to Look at the Context

It's Different

- Everything has to be done faster
- Managing content is as critical as managing source code
- We're not all in one place or one time

It's Not Different

- We've always had to be fast
- We've know how to manage source code, how different could content be?
- We've done geographically distributed projects before

However

- The context is our set of weaknesses (as an industry)
- The Internet *is* different, not because it's innately different, but because the way we work stresses us at our most vulnerable points

Possible Trends

- Iterative and incremental development leads to incremental and iterative testing
- Quality is not just about defects
- Opportunities
  - Learn about different kinds of testing and ways to test
  - Effect change in product development
  - Project Management techniques are even more important than before
  - Working with people is more important

QW2001 Workshop W1

Dr. Cem Kaner
(Florida Institute of Technology)

Developing The Right Test Documentation

Key Points

- The best approach to project documentation depends on the project context, and not necessarily on published standards.
- There are useful questions for learning your project's requirements for test documentation.
- Context-free questions and lists can guide you in developing good tests and test strategies when you don't have time to develop a full set of documentation.

Presentation Abstract

The best approach to test documentation depends on the project context. For example, a paper-intensive test documentation strategy like IEEE 829 is useful for some projects but can get in the way of development of a high-volume automated testing strategy. In the course of writing the third edition of Testing Computer Software, we are looking at test planning/documentation from a different viewpoint. It seems to us that: * The set of documentation is a deliverable, that can be significantly expensive and that can have a significant impact on the project or the company. * To decide what documentation is appropriate, we should do a requirements analysis, asking

- who are the favored, disfavored, and ignored users/recipients of this documentation and why
- what they need or want, and why
- what it costs to fully or partially satisfy these requirements

We also consider the content of the test plan. We think that we have some guidance to offer in terms of evaluating the coverage of the test documentation (how well different aspects of the product are covered and how well different risks are covered).

The associated paper will present some specific test documentation techniques that we use (various types of charts), that we have taught before.

This presentation pulls together work from other talks and from the Los Altos Workshops on Software Testing. There is a lot of material. It can easily fill a day's

tutorial and it can be scaled back to a shorter session (45-90 minutes) that is supported by a long paper.

About the Author

Cem Kaner is Professor of Computer Sciences at the Florida Institute of Technology.

Prior to joining Florida Tech, Kaner worked in Silicon Valley for 17 years, doing and managing programming, user interface design, testing, and user documentation. He is the senior author (with Jack Falk and Hung Quoc Nguyen) of TESTING COMPUTER SOFTWARE (2nd Edition) and (with David Pels) of BAD SOFTWARE: WHAT TO DO WHEN SOFTWARE FAILS.

Through his consulting firm, KANER.COM, he teaches courses on black box software testing and consults to software publishers on software testing, documentation, and development management.

Kaner is also the co-founder and co-host of the Los Altos Workshop on Software Testing, the Software Test Managers' RoundTable, the Workshop on Heuristic & Exploratory Techniques, and the Florida Workshops on Model-Based Testing.

Kaner is also attorney whose practice is focused on the law of software quality. He is active (as an advocate for customers, authors, and small development shops) in several legislative drafting efforts involving software licensing, software quality regulation, and electronic commerce.

Kaner holds a B.A. in Arts & Sciences (Math, Philosophy), a Ph.D. in Experimental Psychology (Human Perception & Performance: Psychophysics), and a J.D. (law degree). He is Certified in Quality Engineering by the American Society for Quality.

Developing the Right Test Documentation

Cem Kaner, J.D., Ph.D.
Department of Computer Sciences
Florida Institute of Technology

James Bach
Satisfice, Inc.

May, 2001
Software Quality Week

Acknowledgments

- These notes outline the test planning chapter in prep for Testing Computer Software, 3rd Ed., by Cem Kaner, James Bach, Hung Quoc Nguyen, Jack Falk, Brian Lawrence & Bob Johnson. They incorporate and adapt materials by these authors.
- Many of the ideas in these notes were reviewed and refined at the Third Los Altos Workshop on Software Testing (LAWST), February 7-8, 1998, and at the Eleventh LAWST, October 28-29, 2000.
  - The participants at LAWST 3 were: Chris Agruss, James Bach, Karla Fisher, David Gelperin, Kenneth Groder, Elisabeth Hendrickson, Doug Hoffman, III (recorder), Bob Johnson, Cem Kaner (host), Brian Lawrence (facilitator), Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Johanna Rothman, Jane Stepak, Melora Svoboda, Jeremy White, and Rodney Wilson.
  - The participants at LAWST 11 were: Chris Agruss, James Bach, Hans Buwalda, Marge Farrell. Sam Guckenheimer, Elisabeth Hendrickson, Doug Hoffman, III (recorder), Bob Johnson, Karen Johnson, Cem Kaner (host), Brian Lawrence (facilitator), Alan Myrvold, Hung Quoc Nguyen, Noel Nyman, Neal Reizer, Amit Singh, and Melora Svoboda.

Abstract

This workshop has grown out of our dissatisfaction with paper-intensive approaches that attempt to provide a seemingly reproducible, somewhat mechanical process for planning and managing testing and test documentation. Over the past 17 years, we have criticized IEEE standard 829 (on software test documentation) and related approaches as being often inappropriate.

Colleagues have asked what we would put in IEEE 829's place. To date, our responses have been piecemeal. This seminar's notes are a draft of our attempt to write a more comprehensive response.

- They start from the premise that the best approach to test documentation depends on the project context. For example, creating detailed test documentation can be useful for some projects but can get in the way of the development of a high-volume automated testing strategy. *What are the relevant differences between these projects?* Before adopting an implementation guideline (like IEEE 829), we should analyze our requirements. There is no point spending a fortune on creating a deliverable (here, the test documentation set) that will not be used or that will interfere with the efficient running of the project. Instead, we should build a documentation set that will actually satisfy the real needs of the project.
- The notes also reflect our view that testing is an exercise in critical thinking and careful questioning. A test case is a question that you ask of the program *(Are you broken in this way?).* The point of a test case is to reduce uncertainty associated with the product. (A test is good if it will reduce uncertainty, whether it finds a bug or not.) A test plan is a structure for asking questions of the project and the product. These notes suggest strategies for asking better questions, and they provide useful clusters of questions.
- The notes also provide samples of some common test planning documents, such as tables and matrices. These will probably be among the building blocks of any testing program that you set up.

Overview

- Problems with the (allegedly) standard approach
- Defining your documentation requirements
- A model for testing and test documentation
- Test documentation elements

Problems with the (allegedly) standard approach

- IEEE Standard 829 for Software Test Documentation
  - Test plan
  - Test-design specification
  - ***Test-case specification***
    - ***Test-case specification identifier***
    - ***Test items***
    - ***Input specifications***
    - ***Output specifications***
    - ***Environmental needs***
    - ***Special procedural requirements***
    - ***Intercase dependencies***
  - Test-procedure specification
  - Test-item transmittal report
  - Test-log

We often see one or more pages per test case.

Problems with the (allegedly) standard approach

- What is the documentation cost per test case?
- What is the maintenance cost of the documentation, per test case?
- If software design changes create documentation maintenance costs, how much inertia do we build into our system? How much does extensive test documentation add to the cost of late improvement of the software? How much should we add?
- What inertia is created in favor of invariant regression testing?
- Is this incompatible with exploratory testing? Do we always want to discourage exploration?

Problems with the (allegedly) standard approach

- What is the impact on high-volume test automation?
- How often do project teams start to follow 829 but then give it up mid-project? What does this do to the net quality of the test documentation and test planning effort?
- WHAT REQUIREMENTS DOES A STANDARD LIKE THIS FULFILL?
- WHICH STAKEHOLDERS GAIN A NET BENEFIT FROM IEEE STANDARD DOCUMENTATION?
- WHAT BENEFITS DO THEY GAIN, AND WHY ARE THOSE BENEFITS IMPORTANT TO THEM?

Problems with the (allegedly) standard approach

It is essential to understand your requirements for test documentation.

Unless following a "standard" helps you meet your requirements, it is empty at best, anti-productive at worst.

Defining documentation requirements

- Stakeholders, interests, actions, objects
- Asking questions
- Context-free questions
- Context-free questions specific to test planning
- Evaluating a plan

Discovering Requirements

- Requirements
  - Anything that drives or constrains design
- Stakeholders
  - Favored, disfavored, and neutral stakeholders
- Stakeholders' interests
  - Favored, disfavored, and neutral interests
- Actions
  - Actions support or interfere with interests
- Objects

Questioning

- Requirements analysis requires information gathering
  - Read books on consulting
  - Gause & Weinberg, *Exploring Requirements* is an essential source on context-free questioning
- There are many types of questions:
  - Open vs. closed
  - Hypothetical vs. behavioral
  - Opinion vs. factual
  - Historical vs. predictive
  - Context-dependent and context-free

The classic context-free questions

- The traditional newspaper reporters' questions are:
  - Who
  - What
  - When
  - Where
  - How
  - Why
- For example, *Who will use this feature? What does this user want to do with it? Who else will use it? Why? Who will choose not to use it? What do they lose? What else does this user want to do in conjunction with this feature? Who is not allowed to use this product or feature, why, and what security is in place to prevent them?*
- We use these in conjunction with questions that come out of the testing model (see below). The model gives us a starting place. We expand it by asking each of these questions as a follow-up to the initial question.

Context-Free Questions: Defining the Problem

Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140) and Bach's Evaluation Strategies (Rapid Testing Course notes)*

- Why is it necessary to solve the problem?
- What benefits will you receive by solving the problem?
- What is the unknown?
- What is it that you don't yet understand?
- What is the information that you have?
- What is the source of this problem? (Specs? Field experience? An individual stakeholder's preference?)
- Who are the stakeholders?
- How does it relate to which stakeholders?
- What isn't the problem?
- Is the information sufficient? Or is it insufficient? Or redundant? Or contradictory?
- Should you draw a diagram of the problem? A figure?

Context-Free Questions: Defining the Problem

- Where are the boundaries of the problem?
- What product elements does it apply to?
- How does this problem relate to the quality criteria?
- Can you separate the various parts of the problem? Can you write them down? What are the relationships of the parts of the problem?
- What are the constants (things that can't be changed) of the problem?
- What are your critical assumptions about this problem?
- Have you seen this problem before?
- Have you seen this problem in a slightly different form?
- Do you know a related problem?
- Try to think of a familiar problem having the same or a similar unknown.
- Suppose you find a problem related to yours that has already been solved. Can you use it? Can you use its method?
- Can you restate your problem? How many different ways can you restate it? More general? More specific? Can the rules be changed?
- What are the best, worst, and most probable cases you can imagine?

Context-Free Questions

Context-free process questions
- Who is the client?
- What is a successful solution worth to this client?
- What is the real (underlying) reason for wanting to solve this problem?
- Who can help solve the problem?
- How much time is available to solve the problem?

Context-free product questions
- What problems could this product create?
- What kind of precision is required / desired for this product?

Metaquestions (when interviewing someone for info)
- Am I asking too many questions?
- Do my questions seem relevant?
- Are you the right person to answer these questions?
- Is there anyone else who can provide additional information?
- Is there anything else I should be asking?
- Is there anything you want to ask me?
- May I return to you with more questions later?

***A sample of additional questions based on Gause & Weinberg's* Exploring Requirements p. 59-64**

What is your group's mission?

- Find important problems
- Assess quality
- Certify to standard
- Fulfill process mandates
- Satisfy stakeholders
- Assure accountability
- Advise about QA
- Advise about testing
- Advise about quality
- Maximize efficiency
- Minimize time
- Minimize cost

The quality of testing depends on which of these possible missions matter and how they relate.

Many debates about the goodness of testing are really debates over missions and givens.

Test Docs Requirements Questions

- **Is test documentation a product or tool?**
- **Is software quality driven by legal issues or by market forces?**
- **How quickly is the design changing?**
- **How quickly does the specification change to reflect design change?**
- **Is testing approach oriented toward proving conformance to specs or nonconformance with customer expectations?**
- **Does your testing style rely more on already-defined tests or on exploration?**
- **Should test docs focus on what to test (objectives) or on how to test for it (procedures)?**
- **Should the docs ever control the testing project?**

Test Docs Requirements Questions

- **If the docs control parts of the testing project, should that control come early or late in the project?**
- **Who are the primary readers of these test documents and how important are they?**
- **How much traceability do you need? What docs are you tracing back to and who controls them?**
- **To what extent should test docs support tracking and reporting of project status and testing progress?**
- **How well should docs support delegation of work to new testers?**
- **What are your assumptions about the skills and knowledge of new testers?**
- **Is test doc set a process model, a product model, or a defect finder?**

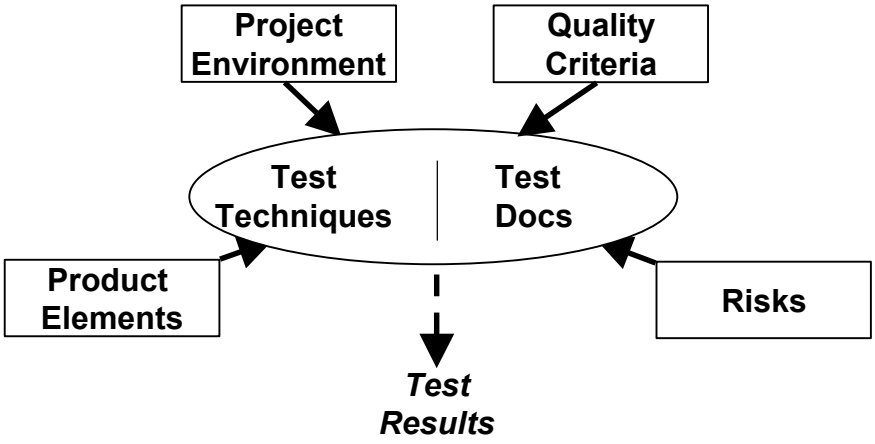
Test Docs Requirements Questions

- **A test suite should provide prevention, detection, and prediction. Which is the most important for this project?**
- **How maintainable are the test docs (and their test cases)? And, how well do they ensure that test changes will follow code changes?**
- **Will the test docs help us identify (and revise/restructure in face of) a permanent shift in the risk profile of the program?**
- **Are (should) docs (be) automatically created as a byproduct of the test automation code?**

Ultimately, write a mission statement

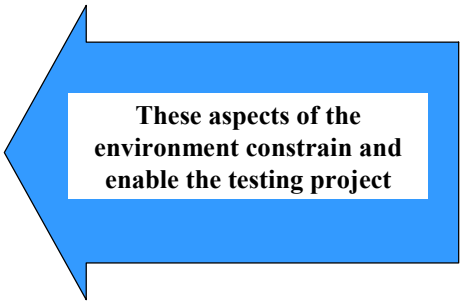
- Try to describe your core documentation requirements in one sentence that doesn't have more than three components.
- Examples:
  - The test documentation set will primarily support our efforts to find bugs in this version, to delegate work, and to track status.
  - The test documentation set will support ongoing product and test maintenance over at least 10 years, will provide training material for new group members, and will create archives suitable for regulatory or litigation use.

A Model of Software Testing



Project Environment Factors:

- Stakeholders
- Processes
- Staff
- Schedules
- Equipment
- Tools & Test Materials
- Information
- Items Under Test
- Logistics
- Budget
- Deliverables



Project Factors

- **Stakeholders:**
  - ***Anyone who is a client of the main project***
  - ***Anyone who is a client of the testing project***
    - Includes customers (purchasers), end users, tech support, programmers, project mgr, doc group, etc.
- **Processes:**
  - ***The tasks and events that comprise the main project***
    - How the overall project is run
  - ***The tasks and events that comprise the test project***
    - How the testing project is run
- **Staff:**
  - ***Everyone who helps develop the product***
    - *Sources of information and assistance*
  - ***Everyone who will perform or support testing***
    - *Special talents or experiences of team members*
    - *Size of the group*
    - *Extent to which they are focused or are multi-tasking*
    - *Organization: collaboration & coordination of the staff*
    - *Is there an independent test lab?*

Project Factors

- **Schedules:** ***The sequence, duration and synchronization of events***
  - When will testing start and how long is it expected to take?
  - When will specific product elements be available to test?
  - When will devices or tools be available to support testing?
- **Equipment:** ***Hardware required for testing***
  - What devices do we need to test the product with? Do we have them?
- **Tools & Test Materials:** ***Software required or desired for testing.***
  - Automation: Are such tools available? Do we want to use them? Do we have them? Do we understand them?
  - Probes or diagnostics to help observe the product under test?
  - Matrices, checklists, other testing documentation?
- **Information:** ***(As needed for testing) about the project or product.***
  - Specifications, requirements documents, other reference materials to help us determine pass/fail or to credibly challenge odd behaviour.
    - What is the availability of these documents?
    - What is the volatility of these documents?

Project Factors

- **Items Under Test:** ***Anything that will be tested***
  - For each product element:
    - Is it available (or when will it be)?
    - Is it volatile (and what is the change process)?
    - Is it testable?
- **Logistics:** ***Facilities and support needed for organizing and conducting the testing***
  - Do we have the supplies / physical space, power, light / security systems (if needed) / procedures for getting more?
- **Budget:** ***Money and other resources for testing***
  - Can we afford the staff, space, training, tools, supplies, etc.?
- **Deliverables:** ***The observable products of the test project***
  - Such as bug reports, summary reports, test documentation, master disk.
    - What are you supposed to create and can you do it?
  - Will we archive the items under test and other products of testing?

Product Elements: **A product is…**

An experience or solution provided to a customer.

Everything that comes in the box, plus the box!

Functions and data, executed on a platform, that serve a purpose for a user.

1 A software product is much more than code.
2 It involves a purpose, platform, and user.
3 It consists of many interdependent *elements*.

Product Elements:

- **Structures: *Everything that comprises the physical product***
  - Code: the code structures that comprise the product, from executables to individual routines
  - Interfaces: points of connection and communication between subsystems
  - Hardware: hardware components integral to the product
  - Non-executable files: any files other than programs, such as text files, sample data, help files, etc.
  - Alternate Media: anything beyond software and hardware, such as paper documents, web links and content, packaging, license agreements, etc.

Product Elements:

- **Functions: *Everything that the product does*.**
  - User Interface: functions that mediate the exchange of data with the user
  - System Interface: functions that exchange data with something other than the user, such as with other programs, hard disk, network, printer, etc.
  - Application: functions that define or distinguish the product or fulfill core requirements
  - Error Handling: functions that detect and recover from errors, including error messages
  - Testability: functions provided to help test the product, such as diagnostics, log files, asserts, test menus, etc.
- **Temporal relationships: *How the program functions over time***
  - Sequential operation: state-to-state transitions
  - Data: changes in variables over time
  - System interactions: such as synchronization or ordering of events in distributed systems

Product Elements:

- **Data:** ***Everything that the product processes***
  - Input: data that is processed by the product
  - Output: data that results from processing by the product
  - Preset: data supplied as part of the product or otherwise built into it, such as prefab databases, default values, etc.
  - Persistent: data stored internally and expected to persist over multiple operations. This includes modes or states of the product, such as options settings, view modes, contents of documents, etc.
  - Temporal: data based on time, such as date stamps or number of events recorded in a unit of time

Product Elements:

- **Platform:** ***Everything on which the product depends***
  - External Hardware: components and configurations that are not part of the shipping product, but are required (or optional) in order for the product to work. Includes CPU's, memory, keyboards, peripheral boards, etc.
  - External Software: software components and configurations that are not a part of the shipping product, but are required (or optional) in order for the product to work. Includes operating systems, concurrently executing applications, drivers, fonts, etc.
- **Operations:** ***How the product will be used***
  - Usage Profile: the pattern of usage, over time, including patterns of data that the product will typically process in the field. This varies by user and type of user.
  - Environment: the physical environment in which the product will be operated, including such elements as light, noise, and distractions.

Product Elements: Coverage

Product coverage is the proportion of the product that has been tested.

- **There are as many kinds of coverage as there are ways to model the product.**
  - Structural
  - Functional
  - Temporal
  - Data
  - Platform
  - Operations

See Software Negligence & Testing Coverage at www.kaner.com for 101 examples of coverage "measures."

Quality Criteria

- Capability
- Reliability
- Usability
- Performance
- Installability
- Compatibility
- Supportability
- Testability
- Maintainability
- Portability
- Localizability
- Efficiency

Quality is value to some person
-- Jerry Weinberg

Risk

Hazard:

A dangerous condition (something that could trigger an accident)

Risk:

Possibility of suffering loss or harm.

Accident:

A hazard is encountered, resulting in loss or harm.

- Useful material available free at http://seir.sei.cmu.edu
- http://www.coyotevalley.com (Brian Lawrence)
- Good paper by Stale Amland, *Risk Based Testing and Metrics,* 16th International Conference on Testing Computer Software, 1999.

Risk

- Project risk management involves
  - Identification of the different risks to the project (issues that might cause the project to fail or to fall behind schedule or to cost too much or to dissatisfy customers or other stakeholders)
  - Analysis of the potential costs associated with each risk
  - Development of plans and actions to reduce the likelihood of the risk or the magnitude of the harm
  - Continuous assessment or monitoring of the risks (or the actions taken to manage them)

Risk-Based Testing

- Two key dimensions:
  - Find errors (risk-based approach to technical tasks of testing)
  - Manage the process of finding errors (risk-based test management)
- Our focus today is on methods for finding errors efficiently.

Risks: Where to look for errors

- **Qualities**: Failure to conform to a quality criterion (risk of unreliability, risk of unmaintainability, etc.)
- **New things**: newer features may fail.
- **New technology:** new concepts lead to new mistakes.
- **Learning Curve:** mistakes due to ignorance.
- **Changed things**: changes may break old code.
- **Late change:** rushed decisions, rushed or demoralized staff lead to mistakes.
- **Rushed work:** some tasks or projects are chronically underfunded and all aspects of work quality suffer.

Risks: Where to look for errors

- **Tired programmers:** long overtime over several weeks or months yields inefficiencies and errors
- **Other staff issues:** alcoholic, mother died, two programmers who won't talk to each other (neither will their code)…
- **Just slipping it in:** pet feature not on plan may interact badly with other code.
- **N.I.H.:** external components can cause problems.
- **N.I.B.:** (not in budget) Unbudgeted tasks may be done shoddily.

Risks: Where to look for errors

- **Ambiguity**: ambiguous descriptions (in specs or other docs) can lead to incorrect or conflicting implementations.
- **Conflicting requirements:** ambiguity often hides conflict, result is loss of value for some person.
- **Unknown requirements**: requirements surface throughout development. Failure to meet a legitimate requirement is a failure of quality for that stakeholder.
- **Evolving requirements**: people realize what they want as the product develops. Adhering to a start-of-the-project requirements list may meet contract but fail product. (check out **http//www.agilealliance.org/)**

Risks: Where to look for errors

- **Complexity**: complex code may be buggy.
- **Bugginess:** features with many known bugs may also have many unknown bugs.
- **Dependencies:** failures may trigger other failures.
- **Untestability:** risk of slow, inefficient testing.
- **Little unit testing:** programmers find and fix most of their own bugs. Shortcutting here is a risk.
- **Little system testing so far**: untested software may fail.
- **Previous reliance on narrow testing strategies:** (e.g. regression, function tests), can yield a backlog of errors surviving across versions.

Risks: Where to look for errors

- **Weak testing tools:** if tools don't exist to help identify / isolate a class of error (e.g. wild pointers), the error is more likely to survive to testing and beyond.
- **Unfixability:** risk of not being able to fix a bug.
- **Language-typical errors**: such as wild pointers in C. See
  - Bruce Webster, *Pitfalls of Object-Oriented Development*
  - Michael Daconta et al. *Java Pitfalls*

Risks: Where to look for errors

- **Criticality**: severity of failure of very important features.
- **Popularity**: likelihood or consequence if much used features fail.
- **Market:** severity of failure of key differentiating features.
- **Bad publicity:** a bug may appear in PC Week.
- **Liability:** being sued.

Bug Patterns as a Source of Risk

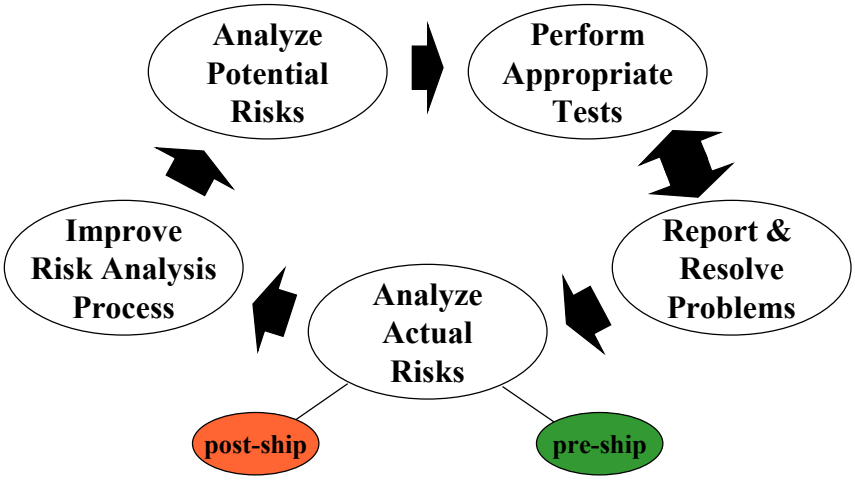
- *Testing Computer Software* lays out a set of 480 common defects. You can use these or develop your own list.
  - *Find a defect in the list*
  - *Ask whether the software under test could have this defect*
  - *If it is theoretically possible that the program could have the defect, ask how you could find the bug if it was there.*
  - *Ask how plausible it is that this bug could be in the program and how serious the failure would be if it was there.*
  - *If appropriate, design a test or series of tests for bugs of this type.*

Build Your Own Model of Bug Patterns

Too many people start and end with the TCS bug list. It is outdated. It was outdated the day it was published. And it doesn't cover the issues in *your* system. Building a bug list is an ongoing process that constantly pays for itself. Here's an example from Hung Nguyen:

- This problem came up in a client/server system. The system sends the client a list of names, to allow verification that a name the client enters is not new.
- Client 1 and 2 both want to enter a name and client 1 and 2 both use the same new name. Both instances of the name are new relative to their local compare list and therefore, they are accepted, and we now have two instances of the same name.
- As we see these, we develop a library of issues. The discovery method is exploratory, requires sophistication with the underlying technology.
- Capture winning themes for testing in charts or in scripts-on-their-way to being automated.

Risk-Driven Testing Cycle



Test Techniques

- Analyze the situation.
- Model the test space.
- Select what to cover.
- Determine test oracles.
- Configure the test system.
- Operate the test system.
- Observe the test system.
- Evaluate the test results.

A *test technique* is a recipe for performing these tasks that will reveal something worth reporting

General Test Techniques

- Function
- Regression
- Domain driven
- Stress driven
- Specification driven
- Risk driven
- Scenario / use case / transaction flow
- User testing
- Exploratory
- Random / statistical

Function Testing

- Tag line
  - "Black box unit testing."
- Fundamental question or goal
  - Test each function thoroughly, one at a time.
- Paradigmatic case(s)
  - Spreadsheet, test each item in isolation.
  - Database, test each report in isolation
- Strengths
  - Thorough analysis of each item tested
- Blind spots
  - Misses interactions, misses exploration of the benefits offered by the program.

Regression Testing

- Tag line
  - "Repeat testing after changes."
- Fundamental question or goal
  - Manage the risks that (a) a bug fix didn't fix the bug or (b) the fix (or other change) had a side effect.
- Paradigmatic case(s)
  - Bug regression (Show that a bug was not fixed)
  - Old fix regression (Show that an old bug fix was broken)
  - General functional regression (Show that a change caused a working area to break.)
  - Automated GUI regression suites
- Strengths
  - Reassuring, confidence building, regulator-friendly

Regression Testing

- Blind spots / weaknesses
  - Anything not covered in the regression series.
  - Repeating the same tests means not looking for the bugs that can be found by other tests.
  - Pesticide paradox
  - Low yield from automated regression tests
  - Maintenance of this standard list can be costly and distracting from the search for defects.

Automating Regression Testing

- This is the most commonly discussed automation approach:
  - create a test case
  - run it and inspect the output
  - if the program fails, report a bug and try again later
  - if the program passes the test, save the resulting outputs
  - in future tests, run the program and compare the output to the saved results. Report an exception whenever the current output and the saved output don't match.

Potential Regression Advantages

- Dominant paradigm for automated testing.
- Straightforward
- Same approach for all tests
- Relatively fast implementation
- Variations may be easy
- Repeatable tests

GUI Regression: Interesting Papers

- Chris Agruss, Automating Software Installation Testing
- James Bach, Test Automation Snake Oil
- Hans Buwalda, Testing Using Action Words
- Hans Buwalda, Automated testing with Action Words: Abandoning Record & Playback
- Elisabeth Hendrickson, The Difference between Test Automation Failure and Success
- Cem Kaner, Avoiding Shelfware: A Manager's View of Automated GUI Testing
- John Kent, Advanced Automated Testing Architectures
- Bret Pettichord, Success with Test Automation
- Bret Pettichord, Seven Steps to Test Automation Success
- Keith Zambelich, Totally Data-Driven Automated Testing

Domain Testing

- Tag lines
  - "Try ranges and options."
  - "Subdivide the world into classes."
- Fundamental question or goal
  - A stratified sampling strategy.
  - Think of this as a sampling strategy. It provides you with a rationale for selecting a few test cases from a huge population. Divide large space of possible tests into subsets. Pick best representatives from each set.
- Paradigmatic case(s)
  - Equivalence analysis of a simple numeric field
  - Printer compatibility testing

Domain Testing

- In classical domain testing
  - Two values (single points or n-tuples) are equivalent if the program would take the same path in response to each.
- The classical domain strategies all assume
  - that the predicate interpretations are simple, linear inequalities.
  - the input space is continuous and
  - coincidental correctness is disallowed.
- It is possible to move away from these assumptions, but the cost can be high, and the emphasis on paths is troublesome because of the high number of possible paths through the program.
  - Clarke, Hassell, & Richardson, p. 388

Equivalence and Risk

Our working definition of equivalence:

> *Two test cases are equivalent if you expect the same result from each.*

This is fundamentally subjective. It depends on what you expect. And what you expect depends on what errors you can anticipate:

> *Two test cases can only be equivalent by reference to a specifiable risk.*

Two different testers will have different theories about how programs can fail, and therefore they will come up with different classes.

A boundary case in this system is a "best representative."

> *A best representative of an equivalence class is a test that is at least as likely to expose a fault as every other member of the class.*

Domain Testing

- Strengths
  - Find highest probability errors with a relatively small set of tests.
  - Intuitively clear approach, generalizes well
- Blind spots
  - Errors that are not at boundaries or in obvious special cases.
  - Also, the actual domains are often unknowable.

Domain Testing: Interesting Papers

- Thomas Ostrand & Mark Balcer*, The Category-partition Method For Specifying And Generating Functional Tests,* Communications of the ACM, Vol. 31, No. 6, 1988.
- Debra Richardson, et al., *A Close Look at Domain Testing*, IEEE Transactions On Software Engineering, Vol. SE-8, NO. 4, July 1982
- Michael Deck and James Whittaker**, *Lessons learned from fifteen years of cleanroom testing.* STAR '97 Proceedings** (in this paper, the authors adopt boundary testing as an adjunct to random sampling.)
- Richard Hamlet & Ross Taylor, Partition Testing Does Not Inspire Confidence, Proceedings of the Second Workshop on Software Testing, Verification, and Analysis, IEEE Computer Society Press, 206-215, July 1988

Stress Testing

- Tag line
  - "Overwhelm the product."
- Fundamental question or goal
  - Learn about the capabilities and weaknesses of the product by driving it through failure and beyond. What does failure at extremes tell us about changes needed in the program's handling of normal cases?
- Paradigmatic case(s)
  - Buffer overflow bugs
  - High volumes of data, device connections, long transaction chains
  - Low memory conditions, device failures, viruses, other crises.
- Strengths
  - Expose weaknesses that will arise in the field.
  - Expose security risks.
- Blind spots
  - Weaknesses that are not made more visible by stress.

Stress Testing: Interesting Papers

- Astroman66, Finding and Exploiting Bugs 2600
- Bruce Schneier, Crypto-Gram, May 15, 2000
- James A. Whittaker and Alan Jorgensen, Why Software Fails
- Whittaker & Jorgenson, How to Break Software.

Specification-Driven Testing

- Tag line:
  - "Verify every claim."
- Fundamental question or goal
  - Check the product's conformance with every statement in every spec, requirements document, etc.
- Paradigmatic case(s)
  - Traceability matrix, tracks test cases associated with each specification item.
  - User documentation testing

Specification-Driven Testing

- Strengths
  - Critical defense against warranty claims, fraud charges, loss of credibility with customers.
  - Effective for managing scope / expectations of regulatory-driven testing
  - Reduces support costs / customer complaints by ensuring that no false or misleading representations are made to customers.
- Blind spots
  - Any issues not in the specs or treated badly in the specs /documentation.

Scenario Testing

Tag lines

- "Do something useful and interesting"
- "Do one thing after another."

Fundamental question or goal

- Challenging cases that reflect real use.

Paradigmatic case(s)

- Appraise product against business rules, customer data, competitors' output
- Life history testing (Hans Buwalda's "soap opera testing.")
- Use cases are a simpler form, often derived from product capabilities and user model rather than from naturalistic observation of systems of this kind.

Scenario Testing

- The ideal scenario has several characteristics:
  - It is realistic (e.g. it comes from actual customer or competitor situations).
  - There is no ambiguity about whether a test passed or failed.
  - The test is complex, that is, it uses several features and functions.
  - There is a stakeholder who will make a fuss if the program doesn't pass this scenario.
- Strengths
  - Complex, realistic events. Can handle (help with) situations that are too complex to model.
  - Exposes failures that occur (develop) over time
- Blind spots
  - Single function failures can make this test inefficient.
  - Must think carefully to achieve good coverage.

Scenario Testing: Interesting Papers

- Hans Buwalda on Soap Operas (in the conference proceedings of STAR East 2000)
- Kaner, A pattern for scenario testing, at www.testing.com
- Lots of literature on use cases

Risk-Based Testing

- Tag line
  - "Find big bugs first."
- Fundamental question or goal
  - Define and refine tests in terms of the kind of problem (or risk) that you are trying to manage
  - OR prioritize the testing effort in terms of the relative risk of different areas or issues we could test for.
- Paradigmatic case(s)
  - Failure Mode and Effects Analysis (FMEA)
  - Equivalence class analysis, reformulated.
  - Test in order of frequency of use (Musa).
  - Stress tests, error handling tests, security tests, tests looking for predicted or feared errors, sample from predicted-bugs list.

Risk-Based Testing

- Strengths
  - Optimal prioritization (assuming we correctly identify and prioritize the risks)
  - High power tests
- Blind spots
  - Risks that were not identified or that are surprisingly more likely.
  - Some "risk-driven" testers seem to operate too subjectively. How will I know what level of coverage that I've reached? How do I know that I haven't missed something critical?

Evaluating Risk

- Several approaches that call themselves "risk-based testing" ask which tests we should run and which we should skip if we run out of time.
- We think this is only half of the risk story. The other half is focuses on test design.
  - It seems to us that a key purpose of testing is to find defects. So, a key strategy for testing should be defect-based. Every test should be questioned:
    - How will this test find a defect?
    - What kind of defect do you have in mind?
    - What power does this test have against that kind of defect? Is there a more powerful test? A more powerful suite of tests?

Evaluating Risk

- Many of us who think about testing in terms of risk, analogize testing of software to the testing of theories:
  - Karl Popper, in his famous essay *Conjectures and Refutations*, lays out the proposition that a scientific theory gains credibility by being subjected to (and passing) harsh tests that are intended to refute the theory.
  - We can gain confidence in a program by testing it harshly (if it passes the tests). Subjecting it to easy tests doesn't tell us much about what will happen to the program in the field.

Risk-Based Testing: Interesting Papers

- Stale Amland, Risk Based Testing
- James Bach, Reframing Requirements Analysis
- James Bach, Risk and Requirements- Based Testing
- James Bach, James Bach on Risk-Based Testing
- Stale Amland & Hans Schaefer, Risk based testing, a response
- Carl Popper, Conjectures & Refutations

User Testing

- Tag line
  - Strive for realism
  - Let's try this with real humans (for a change).
- Fundamental question or goal
  - Identify failures that will arise in the hands of a person, i.e. breakdowns in the overall human/machine/software system.
- Paradigmatic case(s)
  - Beta testing
  - In-house experiments using a stratified sample of target market
  - Usability testing

User Testing

•Strengths
- Design issues are more credibly exposed.
- Can demonstrate that some aspects of product are incomprehensible or lead to high error rates in use.
- In-house tests can be monitored with flight recorders (capture/replay, video), debuggers, other tools.
- In-house tests can focus on areas / tasks that you think are (or should be) controversial.

•Blind spots
- Coverage is not assured (serious misses from beta test, other user tests)
- Test cases can be poorly designed, trivial, unlikely to detect subtle errors.
- Beta testing is not free, beta testers are not skilled, the technical results are mixed. Distinguish marketing betas from technical betas.

Exploratory Testing

Simultaneously:
- Learn about the product
- Learn about the market
- Learn about the ways the product could fail
- Learn about the weaknesses of the product
- Learn about how to test the product
- Test the product
- Report the problems
- Advocate for repairs
- **Develop new tests based on what you have learned so far.**

Exploratory Testing

- Tag line
  - "Simultaneous learning, planning, and testing."
- Fundamental question or goal
  - Software comes to tester under-documented and/or late. Tester must simultaneously learn about the product and about the test cases / strategies that will reveal the product and its defects.
- Paradigmatic case(s)
  - Skilled exploratory testing of the full product
  - Rapid testing
  - Emergency testing (including thrown-over-the-wall test-it-today testing.)
  - Third party components.
  - Troubleshooting / follow-up testing of defects.

Exploratory Testing

- Strengths
  - Customer-focused, risk-focused
  - Takes advantage of each tester's strengths
  - Responsive to changing circumstances
  - Well managed, it avoids duplicative analysis and testing
  - High bug find rates
- Blind spots
  - The less we know, the more we risk missing.
  - Limited by each tester's weaknesses (can mitigate this with careful management)
  - This is skilled work, juniors aren't very good at it.

Paired Exploratory Testing--Acknowledgment

The following, paired testing, slides developed out of several projects.

We particularly acknowledge the help and data from participants in the First and Second Workshops on Heuristic and Exploratory Techniques (Front Royal, VA, November 2000 and March 2001, hosted by James Bach and facilitated by Cem Kaner), those being Jon Bach, Stephen Bell, Rex Black, Robyn Brilliant, Scott Chase, Sam Guckenheimer, Elisabeth Hendrickson, Alan A. Jorgensen, Brian Lawrence, Brian Marick, Mike Marduke, Brian McGrath, Erik Petersen, Brett Pettichord, Shari Lawrence Pfleeger, Becky Winant, and Ron Wilson.

Additionally, we thank Noel Nyman and Ross Collard for insights and James Whittaker for co-hosting one of the two paired testing trials at Florida Tech.

A testing pattern on paired testing was drafted by Brian Marick, based on discussions at the Workshop on Patterns of Software Testing (POST 1) in Boston, January 2001 (hosted primarily by Sam Guckenheimer / Rational and Brian Marick, facilitated by Marick). The latest draft is at "Pair Testing" pattern) (<http://www.testing.com/test-patterns/patterns/pair-testing.pdf>).

Paired Exploratory Testing

- Based on our (and others') observations of effective testing workgroups at several companies. We noticed several instances of high productivity, high creativity work that involved testers grouping together to analyze a product or to scheme through a test or to run a series of tests. We also saw/used it as an effective training technique.
- In 2000, we started trying this out, at WHET, at Satisfice, and at one of Satisfice's clients. The results were spectacular. We obtained impressive results in quick runs at Florida Tech as well, and have since received good reports from several other testers.

Paired Programming

- Developed independently of paired testing, but many of the same problems and benefits apply.
- The eXtreme Programming community has a great deal of experience with paired work, much more than we do, offers many lessons:
  - Kent Beck, *Extreme Programming Explained*
  - Ron Jeffries, Ann Anderson & Chet Hendrickson, *Extreme Programming Installed*
- Laurie Williams of NCSU does research in pair programming. For her publications, see <http://collaboration.csc.ncsu.edu/laurie/>

What is Paired Testing

- Two testers and (typically) one machine.
- Typically (as in XP)
  - Pairs work together voluntarily. One person might pair with several others during a day.
  - A given testing task is the responsibility of one person, who recruits one or more partners (one at a time) to help out.
- We've seen stable pairs who've worked together for years.
- One tester strokes the keys (but the keyboard may pass back and forth in a session) while the other suggests ideas or tests, pays attention and takes notes, listens, asks questions, grabs reference material, etc.

A Paired Testing Session

- **Start with a charter**
  - Testers might operate from a detailed project outline, pick a task that will take a day or less
  - Might (instead or also) create a flipchart page that outlines this session's work or the work for the next few sessions.
    - An exploratory testing session lasts about 60-90 minutes.
  - The charter for a session might include what to test, what tools to use, what testing tactics to use, what risks are involved, what bugs to look for, what documents to examine, what outputs are desired, etc.

Benefits of Paired Testing

- Pair testing is different from many other kinds of pair work because testing is an *idea generation activity* rather than a plan implementation activity. Testing is a heuristic search of an open-ended and multi-dimensional space.
- Pairing has the effect of forcing each tester to explain ideas and react to ideas. When one tester must phrase his thoughts to another tester, that simple process of phrasing seems to bring the ideas into better focus and naturally triggers more ideas.
- If faithfully performed, we believe this will result in more and better ideas that inform the tests.

Benefits of Paired Testing

- Generate more ideas
  - Naturally encourages creativity
  - More information and insight available to apply to analysis of a design or to any aspect of the testing problem
  - Supports the ability of one tester to stay focused and keep testing. This has a major impact on creativity.
- More fun

Benefits of Paired Testing

- Helps the tester stay on task. Especially helps the tester pursue a streak of insight (an exploratory vector).
  - A flash of insight need not be interrupted by breaks for note-taking, bug reporting, and follow-up replicating. The non-keyboard tester can:
    - Keep key notes while the other follows the train of thought
    - Try to replicate something on a second machine
    - Grab a manual, other documentation, a tool, make a phone call, grab a programmer--get support material that the other tester needs.
    - Record interesting candidates for digression
- Also, the fact that two are working together limits the willingness of others to interrupt them, especially with administrivia.

Benefits of Paired Testing

- Better Bug Reporting
  - Better reproducibility
  - Everything reported is reviewed by a second person.
  - Sanity/reasonability check for every design issue
    - (example from Kaner/Black on Star Office tests)
- Great training
  - Good training for novices
  - Keep learning by testing with others
  - Useful for experienced testers when they are in a new domain

Benefits of Paired Testing

- Additional technical benefits
  - Concurrency testing is facilitated by pairs working with two (or more) machines.
  - Manual load testing is easier with several people.
  - When there is a difficult technical issue with part of the project, bring in a more knowledgeable person as a pair

Risks and Suggestions

- Paired testing is *not* a vehicle for fobbing off errand-running on a junior tester. The pairs are partners, the junior tester is often the one at the keyboard, and she is always allowed to try out her own ideas.
- Accountability must belong to one person. Beck and Jeffries, et al. discuss this in useful detail. One member of the pair owns the responsibility for getting the task done.
- Some people are introverts. They need time to work alone and recharge themselves for group interaction.
- Some people have strong opinions and don't work well with others. Coaching may be essential.

Risks and Suggestions

- Have a coach available.
  - Generally helpful for training in paired testing and in the conduct of any type of testing
  - When there are strong personalities at work, a coach can help them understand their shared and separate responsibilities and how to work effectively together.

Exploratory Testing: Interesting Papers

- Chris Agruss & Bob Johnson, Ad Hoc Software Testing Exploring the Controversy of Unstructured Testing
- Whittaker & Jorgenson, How to Break Software

Random / Statistical Testing

- Tag line
  - "High-volume testing with new cases all the time."
- Fundamental question or goal
  - Have the computer create, execute, and evaluate huge numbers of tests.
    - The individual tests are not all that powerful, nor all that compelling.
    - The power of the approach lies in the large number of tests.
    - These broaden the sample, and they may test the program over a long period of time, giving us insight into longer term issues.

Random / Statistical Testing

- Paradigmatic case(s)
  - Some of us are still wrapping our heads around the richness of work in this field. This is a tentative classification
    - NON-STOCHASTIC RANDOM TESTS
    - STATISTICAL RELIABILITY ESTIMATION
    - STOCHASTIC TESTS (NO MODEL)
    - STOCHASTIC TESTS USING ON A MODEL OF THE SOFTWARE UNDER TEST
    - STOCHASTIC TESTS USING OTHER ATTRIBUTES OF SOFTWARE UNDER TEST

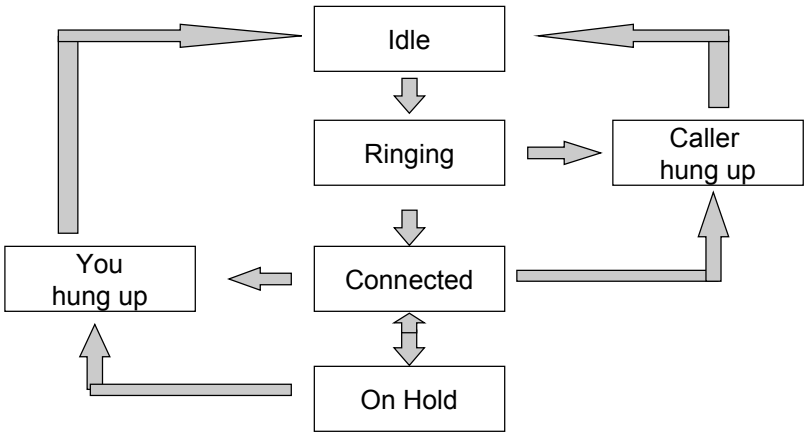
Random / Statistical Testing: Non-Stochastic

- Fundamental question or goal
  - The computer runs a large set of essentially independent tests. The focus is on the results of each test. Tests are often designed to minimize sequential interaction among tests.
- Paradigmatic case(s)
  - Function equivalence testing: Compare two functions (e.g. math functions), using the second as an oracle for the first. Attempt to demonstrate that they are not equivalent, i.e. that the achieve different results from the same set of inputs.
  - Other test using fully deterministic oracles (see discussion of oracles, below)
  - Other tests using heuristic oracles (see discussion of oracles, below)

Statistical Reliability Estimation

- Fundamental question or goal
  - Use random testing (possibly stochastic, possibly oracle-based) to estimate the stability or reliability of the software. Testing is being used primarily to qualify the software, rather than to find defects.
- Paradigmatic case(s)
  - Clean-room based approaches

The Need for Stochastic Testing: An Example



Stochastic Tests--No Model: "Dumb Monkeys"

- Fundamental question or goal
  - High volume testing, involving a long sequence of tests.
  - A typical objective is to evaluate program performance over time.
  - The distinguishing characteristic of this approach is that the testing software does not have a detailed model of the software under test.
  - The testing software might be able to detect failures based on crash, performance lags, diagnostics, or improper interaction with other, better understood parts of the system, but it cannot detect a failure simply based on the question, "Is the program doing what it is supposed to or not?"

Stochastic Tests-- No Model: "Dumb Monkeys")

- Paradigmatic case(s)
  - Executive monkeys: Know nothing about the system. Push buttons randomly until the system crashes.
  - Clever monkeys: More careful rules of conduct, more knowledge about the system or the environment. See Freddy.
  - O/S compatibility testing: No model of the software under test, but diagnostics might be available based on the environment (the NT example)
  - Early qualification testing
  - Life testing
  - Load testing
- Notes
  - Can be done at the API or command line, just as well as via UI

Stochastic, assert or diagnostics-based random tests

- Fundamental question or goal
  - High volume random testing using random sequence of fresh or pre-defined tests that may or may not self-check for pass/fail. The primary method for detecting pass/fail uses assertions (diagnostics built into the program) or other (e.g. system) diagnostics.
- Paradigmatic case(s)
  - Telephone example (asserts)
  - Embedded software example (diagnostics)

Random Testing: Stochastic, Regression-Based

- Fundamental question or goal
  - High volume random testing using random sequence of pre-defined tests that can self-check for pass/fail.
- Paradigmatic case(s)
  - Life testing
  - Search for specific types of long-sequence defects.

Random Testing: Stochastic, Regression-Based

- Notes
  - Create a series of regression tests. Design them so that they don't reinitialize the system or force it to a standard starting state that would erase history. The tests are designed so that the automation can identify failures. Run the tests in random order over a long sequence.
  - This is a low-mental-overhead alternative to model-based testing. You get pass/fail info for every test, but without having to achieve the same depth of understanding of the software. Of course, you probably have worse coverage, less awareness of your actual coverage, and less opportunity to stumble over bugs.
  - Unless this is very carefully managed, there is a serious risk of non-reproduceability of failures.

Random Testing: Sandboxing the Regression Tests

- In a random sequence of standalone tests, we might want to qualify each test, T1, T2, etc, as able to run on its own. Then, when we test a sequence of these tests, we know that errors are due to interactions among them rather than merely to cumulative effects of repetition of a single test.
- Therefore, for each Ti, we run the test on its own many times in one long series, randomly switching as many other environmental or systematic variables during this random sequence as our tools allow.
- We call this the "sandbox" series—Ti is forced to play in its own sandbox until it "proves" that it can behave properly on its own. (This is an 80/20 rule operation. We do want to avoid creating a big random test series that crashes only because one test doesn't like being run or that fails after a few runs under low memory. We want to weed out these simple causes of failure. But we don't want to spend a fortune trying to control this risk.)

Random Testing: Sandboxing the Regression Tests

Suppose that you create a random sequence of standalone tests (that were not sandbox-tested), and these tests generate a hard-to-reproduce failure.

You can run a sandbox on each of the tests in the series, to determine whether the failure is merely due to repeated use of one of them.

Random Testing: Model-based Stochastic Tests

- Fundamental Question or Goal
  - Build a state model of the software. (The analysis will reveal several defects in itself.) Generate random events / inputs to the program. The program responds by moving to a new state. Test whether the program has reached the expected state.
- Paradigmatic case(s)
  - I haven't done this kind of work. Here's what I understand:
    - Works poorly for a complex product like Word
    - Likely to work well for embedded software and simple menus (think of the brakes of your car or walking a control panel on a printer)
    - In general, well suited to a limited-functionality client that will not be powered down or rebooted very often.
    - Maintenance is a critical issue because design changes add or subtract nodes, forcing a regeneration of the model.

Random Testing: Model-based Stochastic Tests

Alan Jorgensen, Software Design Based on Operational Modes, Ph.D. thesis, Florida Institute of Technology:

> The applicability of state machine modeling to mechanical computation dates back to the work of Mealy [Mealy, 1955] and Moore [Moore, 1956] and persists to modern software analysis techniques [Mills, et al., 1990, Rumbaugh, et al., 1999]. Introducing state design into software development process began in earnest in the late 1980's with the advent of the cleanroom software engineering methodology [Mills, et al., 1987] and the introduction of the State Transition Diagram by Yourdon [Yourdon, 1989].
>
> A deterministic finite automata (DFA) is a state machine that may be used to model many characteristics of a software program. Mathematically, a DFA is the quintuple, M = (Q, Σ, δ, q0, F) where M is the machine, Q is a finite set of states, Σ is a finite set of inputs commonly called the "alphabet," δ is the transition function that maps Q x Σ to Q,, q0 is one particular element of Q identified as the initial or stating state, and F ⊆ Q is the set of final or terminating states [Sudkamp, 1988]. The DFA can be viewed as a directed graph where the nodes are the states and the labeled edges are the transitions corresponding to inputs.

Random Testing: Model-based Stochastic Tests

Alan Jorgensen, Software Design Based on Operational Modes, Ph.D. thesis, Florida Institute of Technology:

> When taking this state model view of software, a different definition of software failure suggests itself: "The machine makes a transition to an unspecified state." From this definition of software failure a software defect may be defined as: "Code, that for some input, causes an unspecified state transition or fails to reach a required state."
>
> . . .
>
> Recent developments in software system testing exercise state transitions and detect invalid states. This work, [Whittaker, 1997b], developed the concept of an "operational mode" that functionally decomposes (abstracts) states. Operational modes provide a mechanism to encapsulate and describe state complexity. By expressing states as the cross product of operational modes and eliminating impossible states, the number of distinct states can be reduced, alleviating the state explosion problem.
>
> Operational modes are not a new feature of software but rather a different way to view the decomposition of states. All software has operational modes but the implementation of these modes has historically been left to chance. When used for testing, operational modes have been extracted by reverse engineering.

Random Testing: Thoughts Toward an Architecture

- We have a population of tests, which may have been sandboxed and which may carry self-check info. A test series involves a sample of these tests.
- We have a population of diagnostics, probably too many to run every time we run a test. In a given test series, we will run a subset of these.
- We have a population of possible configurations, some of which can be set by the software. In a given test series, we initialize by setting the system to a known configuration. We may reset the system to new configurations during the series (e.g. every 5th test).

Random Testing: Thoughts Toward an Architecture

- We have an execution tool that takes as input
  - a list of tests (or an algorithm for creating a list),
  - a list of diagnostics (initial diagnostics at start of testing, diagnostics at start of each test, diagnostics on detected error, and diagnostics at end of session),
  - an initial configuration and
  - a list of configuration changes on specified events.
- The tool runs the tests in random order and outputs results
  - to a standard-format log file that defines its own structure so that
  - multiple different analysis tools can interpret the same data.

Random / Statistical Testing

- Strengths
  - Regression doesn't depend on same old test every time.
  - Partial oracles can find errors in young code quickly and cheaply.
  - Less likely to miss internal optimizations that are invisible from outside.
  - Can detect failures arising out of long, complex chains that would be hard to create as planned tests.
- Blind spots
  - Need to be able to distinguish pass from failure. Too many people think "Not crash = not fail."
  - Executive expectations must be carefully managed.
  - These methods will often cover many types of risks, but will obscure the need for other tests less amenable to automation.

Random / Statistical Testing

- Blind spots
  - Testers might spend much more time analyzing the code and too little time analyzing the customer and her uses of the software.
  - Potential to create an inappropriate prestige hierarchy, devaluating the skills of subject matter experts who understand the product and its defects much better than the automators.

Random Testing: Interesting Papers

- Larry Apfelbaum, Model-Based Testing, Proceedings of Software Quality Week 1997 (not included in the course notes)
- Michael Deck and James Whittaker, Lessons learned from fifteen years of cleanroom testing. STAR '97 Proceedings
- Doug Hoffman, Mutating Automated Tests
- Alan Jorgensen, An API Testing Method
- Noel Nyman, GUI Application Testing with Dumb Monkeys.
- Harry Robinson, Finite State Model-Based Testing on a Shoestring.
- Harry Robinson, Graph Theory Techniques in Model-Based Testing.

Test Strategy

- "How we plan to cover the product so as to develop an adequate assessment of quality."
- A good test strategy is:
  - *Diversified*
  - *Specific*
  - *Practical*
  - *Defensible*

Test Strategy

- Makes use of test techniques.
- May be expressed by test procedures and cases.
- Not to be confused with test *logistics*, which involve the details of bringing resources to bear on the test strategy at the right time and place.
- You don't have to know the entire strategy in advance. The strategy can change as you learn more about the product and its problems.

Test Cases/Procedures

- Test cases and procedures should manifest the test strategy.
- If your strategy is to "execute the test suite I got from Joe Third-Party", how does that answer the prime strategic questions:
  - How will you *cover the product* and *assess quality*?
  - How is that *practical* and *justified* with respect to the *specifics* of this project and product?
- If you don't know, then your real strategy is that you're trusting things to work out.

Diverse Half-Measures

- There is no single technique that finds all bugs.
- We can't do any technique perfectly.
- We can't do all conceivable techniques.

Use **"diverse half-measures"--** lots of different points of view, approaches, techniques, even if no one strategy is performed completely.

Test Documentation Elements

- Lists
- Outlines
- Tables
- Matrices

Basic Test Documentation Components

Lists:

- Such as lists of fields, error messages, DLLs

Outlines: An outline organizes information into a hierarchy of lists and sublists

- Such as the testing objectives list later in the course notes

Tables: A table organizes information in two dimensions showing relationships between variables.

- Such as boundary tables, decision tables, combination test tables

Matrices: A matrix is a special type of table used for data collection.

- Such as the numeric input field matrix, configuration matrices

Traceability Matrix

	Var 1	Var 2	Var 3	Var 4	Var 5
Test 1	X	X	X		
Test 2		X		X	
Test 3	X		X	X	
Test 4			X	X	
Test 5				X	X
Test 6	X				X

Traceability Matrix

- The columns involve different test items. A test item might be a function, a variable, an assertion in a specification or requirements document, a device that must be tested, any item that must be shown to have been tested.
- The rows are test cases.
- The cells show which test case tests which items.
- If a feature changes, you can quickly see which tests must be reanalyzed, probably rewritten.
- In general, you can trace back from a given item of interest to the tests that cover it.
- This doesn't specify the tests, it merely maps their coverage.

Myers' Boundary Table

Variable	Valid Case Equivalence Classes	Invalid Case Equivalence Classes	Boundaries and Special Cases	Notes
First number	-99 to 99	> 99 < -99 non-number expressions	99, 100 -99, -100 / : 0 null entry	
Second number	same as first	same as first	same	
Sum	-198 to 198			Are there other sources of data for this variable? Ways to feed it bad data?

Revised Boundary Analysis Table

Variable	Equivalence Class	Alternate Equivalence Class	Boundaries and Special Cases	Notes
First number	-99 to 99 digits	> 99 < -99 non-digits expressions	99, 100 -99, -100 /, 0, 9, : leading spaces or 0s null entry	
Second number	same as first	same as first	same	
Sum	-198 to 198 -127 to 127	??? -198 to –128 128 to 198	??? 127, 128, -127, -128	Are there other sources of data for this variable? Ways to feed it bad data?

Note that we've dropped the issue of "valid" and "invalid." This lets us generalize to partitioning strategies that don't have the **concept** of "valid" -- for example, **printer** equivalence classes.

Equivalence Classes: A Broad Concept

The notion of equivalence class is much broader than numeric ranges. Here are some examples:

- Membership in a common group
  - such as employees vs. non-employees. (Note that not all classes have shared boundaries.)
- Equivalent hardware
  - such as compatible modems
- Equivalent event times
  - such as before-timeout and after
- Equivalent output events
  - perhaps any report will do to answer a simple the question: Will the program print reports?
- Equivalent operating environments
  - such as French & English versions of Windows 3.1

Variables Well Suited to Equivalence Class Analysis

Many types of variables, including input, output, internal, hardware and system software configurations, and equipment states can be subject to equivalence class analysis. Here are some examples:

- ranges of numbers
- character codes
- how many times something is done
  - (e.g. shareware limit on number of uses of a product)
  - (e.g. how many times you can do it before you run out of memory)
- how many names in a mailing list, records in a database, variables in a spreadsheet, bookmarks, abbreviations
- size of the sum of variables, or of some other computed value (think binary and think digits)
- size of a number that you enter (number of digits) or size of a character string
- size of a concatenated string
- size of a path specification
- size of a file name
- size (in characters) of a document
- size of a file (note special values such as exactly 64K, exactly 512 bytes, etc.)
- size of the document on the page (compared to page margins) (across different page margins, page sizes)

Variables Well Suited to Equivalence Class Analysis

- size of a document on a page, in terms of the memory requirements for the page. This might just be in terms of resolution x page size, but it may be more complex if we have compression.
- equivalent output events (such as printing documents)
- amount of available memory (> 128 meg, > 640K, etc.)
- visual resolution, size of screen, number of colors
- operating system version
- variations within a group of "compatible" printers, sound cards, modems, etc.
- equivalent event times (when something happens)
- timing: how long between event A and event B (and in which order--races)
- length of time after a timeout (from JUST before to way after) -- what events are important?
- speed of data entry (time between keystrokes, menus, etc.)
- speed of input--handling of concurrent events
- number of devices connected / active
- system resources consumed / available (also, handles, stack space, etc.)
- date and time
- transitions between algorithms (optimizations) (different ways to compute a function)
- most recent event, first event
- input or output intensity (voltage)
- speed / extent of voltage transition (e.g. from very soft to very loud sound)

Using Test Matrices for Routine Issues

- After testing a simple numeric input field a few times, you may prefer a test matrix to present the same tests more concisely.
- Use a test matrix to show/track a series of test cases that are fundamentally similar.
  - For example, for most input fields, you'll do a series of the same tests, checking how the field handles boundaries, unexpected characters, function keys, etc.
  - As another example, for most files, you'll run essentially the same tests on file handling.
- The matrix is a concise way of showing the repeating tests.
  - Put the objects that you're testing on the rows.
  - Show the tests on the columns.
  - Check off the tests that you actually completed in the cells.

Reusable Test Matrix

Numeric Input Field													
		Nothing	LB of value	UB of value	LB of value - 1	UB of value + 1	0	Negative	LB number of digits or chars	UB number of digits or chars	Empty field (clear the default value)	Outside of UB number of digits or chars	Non-digits

Examples of integer-input tests

- Nothing
- Valid value
- At LB of value
- At UB of value
- At LB of value - 1
- At UB of value + 1
- Outside of LB of value
- Outside of UB of value
- 0
- Negative
- At LB number of digits or chars
- At UB number of digits or chars
- Empty field (clear the default value)
- Outside of UB number of digits or chars
- Non-digits
- Wrong data type (e.g. decimal into integer)
- Expressions
- Space
- Non-printing char (e.g., Ctrl+char)
- DOS filename reserved chars (e.g., "\ * . :")
- Upper ASCII (128-254)
- Upper case chars
- Lower case chars
- Modifiers (e.g., Ctrl, Alt, Shift-Ctrl, etc.)
- Function key (F2, F3, F4, etc.)

Error Handling when Writing a File

- full local disk
- almost full local disk
- write protected local disk
- damaged (I/O error) local disk
- unformatted local disk
- remove local disk from drive after opening file
- timeout waiting for local disk to come back online
- keyboard and mouse I/O during save to local disk
- other interrupt during save to local drive
- power out during save to local drive
- full network disk
- almost full network disk
- write protected network disk
- damaged (I/O error) network disk
- remove network disk after opening file
- timeout waiting for network disk
- keyboard / mouse I/O during save to network disk
- other interrupt during save to network drive
- local power out during save to network
- network power during save to network

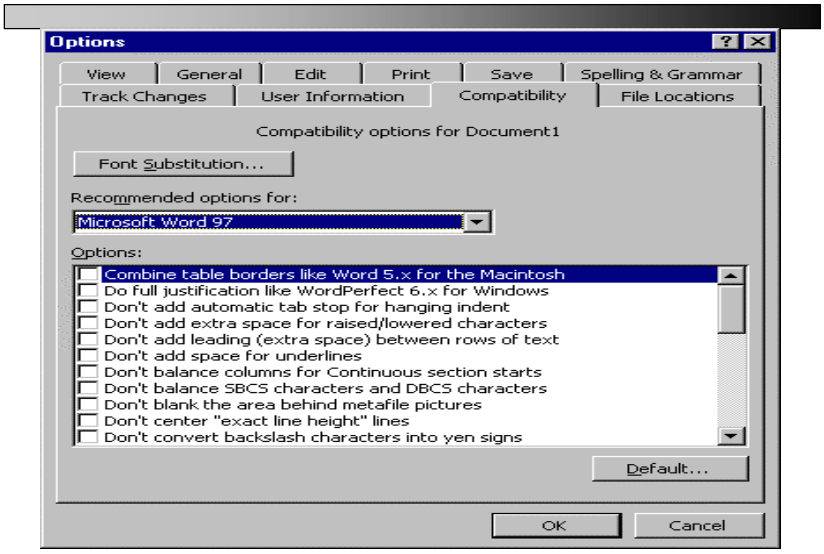
Routine Case Matrices

- **You can often re-use a matrix like this across products and projects.**
- **You can create matrices like this for a wide range of problems. Whenever you can specify multiple tests to be done on one class of object, and you expect to test several such objects, you can put the multiple tests on the matrix.**
- **Mark a cell green if you ran the test and the program passed it. Mark the cell red if the program failed.**
- **Write the bug number of the bug report for this bug.**
- **Write (in the cell) the automation number or identifier or file name if the test case has been automated.**

Routine Case Matrices

- Problems?
  - **What if your thinking gets out of date? (What if this program poses new issues, not covered by the standard tests?)**
  - **Do you need to execute every test every time? (or ever?)**
  - **What if the automation ID number changes? -- We still have a maintenance problem but it is not as obscure.**

Complex Data Relationships



Tabular Format for Data Relationships

Field	Entry source	Display	Print	Related variable	Relationship

Tabular Format for Data Relationships

Once you identify two variables that are related, test them together using boundary values of each or pairs of values that will trigger some other boundary.

- **This is not the most powerful process for looking at relationships. An approach like Cause-Effect Graphing is more powerful, if you have or can build a complete specification.**
- **I started using this chart as an *exploratory* tool for simplifying my look at relationships in overwhelmingly complex programs. (There doesn't have to be a lot of complexity to be "overwhelming.")**

Tabular Format for Data Relationships

•THE TABLE'S FIELDS

Field: *Create a row for each field (`Consultant`, `End Date`, and `Start Date` are examples of fields.)*

Entry Source: *What dialog boxes can you use to enter data into this field? Can you import data into this field? Can data be calculated into this field? List every way to fill the field -- every screen, etc.*

Display: *List every dialog box, error message window, etc., that can display the value of this field. When you re-enter a value into this field, will the new entry show up in each screen that displays the field? (Not always -- sometimes the program makes local copies of variables and fails to update them.)*

Print: *List all the reports that print the value of this field (and any other functions that print the value).*

Related to: *List every variable that is related to this variable. (What if you enter a legal value into this variable, then change the value of a constraining variable to something that is incompatible with this variable's value?)*

Relationship: *Identify the relationship to the related variable.*

Tabular Format for Data Relationships

Many relationships among data:

- **Independence**
  - **Varying one has no effect on the value or permissible values of the other.**
- **Causal determination**
  - **By changing the value of one, we determine the value of the other.**
  - **For example, in MS Word, the extent of shading of an area depends on the object selected. The shading differs depending on Table vs. Paragraph.**
- **Constrained to a range**
  - **For example, the width of a line has to be less than the width of the page.**
  - **In a date field, the permissible dates are determined by the month (and the year, if February).**
- **Selection of rules**
  - **Example, hyphenation rules depend on the language you choose.**

Tabular Format for Data Relationships

Many relationships among data:

- **Logical selection from a list**
  - **processes the value you entered and then figures out what value to use for the next variable. Example: timeouts in phone dialing:**
    - **0 on complete call 555-1212 but 95551212?**
    - **10 on ambiguous completion, 955-5121**
    - **30 seconds incomplete 555-121**
- **Logical selection *of* a list:**
  - **For example, in printer setup, choose:**
    - **OfficeJet, get Graphics Quality, Paper Type, and Color Options**
    - **LaserJet 4, get Economode, Resolution, and Half-toning.**

Look at Marick *(Craft of Software Testing)* for discussion of catalogs of tests for data relationships.

Data Relationship Table

- Looking at the Word options, you see the real value of the data relationships table. Many of these options have a lot of repercussions.
- You might analyze all of the details of all of the relationships later, but for now, it is challenging just to find out what all the relationships ARE.
- The table guides exploration and will surface a lot of bugs.
- ----------
- PROBLEM
- Works great for this release. Next release, what is your support for more exploration?

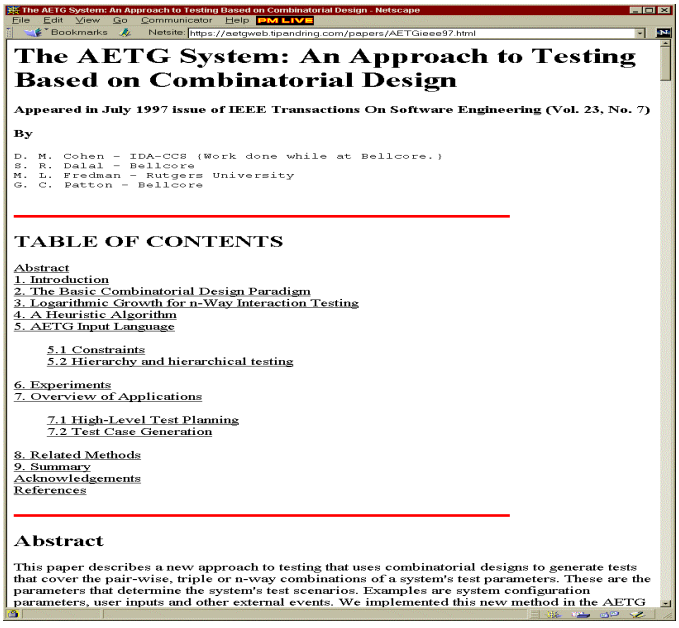
Testing Variables in Combination

Interesting papers.

- Cohen, Dalal, Parelius, Patton,"The Combinatorial Design Approach to Automatic Test Generation",IEEE Software, Sept. 96 http://computer.org:80/software/so1996/s5toc.htm
- Cohen, Dalal, Fredman, Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", IEEE Trans on SW Eng. Vol 23#7, July 97 http://computer.org:80/tse/ts1997/e7toc.htm
- OnLine requires IEEE membership for free access. See http://www.computer.org/epub/
- **Several other papers on AETG are available at https://aetgweb.tipandring.com/AboutAETGweb.html**
- Also interesting: http://www.stsc.hill.af.mil/CrossTalk/1997/oct/planning.html

Combination Chart

	Var 1	Var 2	Var 3	Var 4	Var 5
Test 1	Value 11	Value 12	Value 13	Value 14	Value 15
Test 2	Value 21	Value 22	Value 23	Value 24	Value 25
Test 3	Value 31	Value 32	Value 33	Value 34	Value 35
Test 4	Value 41	Value 42	Value 43	Value 44	Value 45
Test 5	Value 51	Value 52	Value 53	Value 54	Value 55
Test 6	Value 61	Value 62	Value 63	Value 64	Value 65



The AETG System: An Approach to Testing Based on Combinatorial Design - Netscape

File Edit View Go Communicator Help PM LIVE

Bookmarks Netsite: https://aetgweb.tipandring.com/papers/AETGieee97.html

The AETG System: An Approach to Testing Based on Combinatorial Design

Appeared in July 1997 issue of IEEE Transactions On Software Engineering (Vol. 23, No. 7)

By

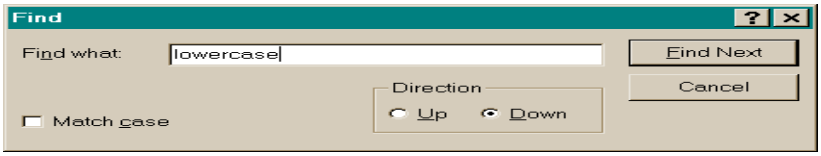
D. M. Cohen - IDA-CCS (Work done while at Bellcore.)
S. R. Dalal - Bellcore
M. L. Fredman - Rutgers University
G. C. Patton - Bellcore

TABLE OF CONTENTS

Abstract

This paper describes a new approach to testing that uses combinatorial designs to generate tests that cover the pair-wise, triple or n-way combinations of a system's test parameters. These are the parameters that determine the system's test scenarios. Examples are system configuration parameters, user inputs and other external events. We implemented this new method in the AETG

Combinations Exercise / Illustration



- Here is a simple Find dialog. It takes three inputs:
  - `Find what:` a text string
  - `Match case:` yes or no
  - `Direction:` up or down
- Simplify this by considering only three values for the text string, "lowercase" and "Mixed Cases" and "CAPITALS".
- (Note: To do a better job, we'd also choose input documents that would yield a "find" and a "don't find" for each case. The input document would be another variable or, really, the intended result (Find / Don't) would be the variable. We'll think about that again after the exercise.)

Combinations Exercise

1 How many combinations of these three variables are possible?

2 List ALL the combinations of these three variables.

3 Now create combination tests that cover all possible pairs of values, but don't try to cover all possible triplets. List one such set.

4 How many test cases are in this set?

Charts: References

You can find plenty of example charts in Bill Perry's books, such as Effective Methods for Software Testing (2nd Ed., Wiley). Several of these will probably be useful, though (like the charts in these notes) you'll have to adapt them to your circumstances.

Evaluating Your Plan: Context Free Questions

Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140) and Bach's Evaluation Strategies (Rapid Testing Course notes)*

- Can you solve the whole problem? Part of the problem?
- What would you like the resolution to be? Can you picture it?
- How much of the unknown can you determine?
- What reference data are you using (if any)?
- What product output will you evaluate?
- How will you do the evaluation?
- Can you derive something useful from the information you have?
- Have you used all the information?
- Have you taken into account all essential notions in the problem?
- Can you separate the steps in the problem-solving process? Can you determine the correctness of each step?
- What creative thinking techniques can you use to generate ideas? How many different techniques?
- Can you see the result? How many different kinds of results can you see?
- How many different ways have you tried to solve the problem?

Evaluating Your Plan: Context Free Questions

- What have others done?
- Can you intuit the solution? Can you check the results?
- What should be done?
- How should it be done?
- Where should it be done?
- When should it be done?
- Who should do it?
- What do you need to do at this time?
- Who will be responsible for what?
- Can you use this problem to solve some other problem?
- What is the unique set of qualities that makes this problem what it is and none other?
- What milestones can best mark your progress?
- How will you know when you are successful?
- How conclusive and specific is your answer?

QW2001 Workshop W2

Mr. John Paul
(Minjoh Technology Solutions)

Automating Software Testing: A Life-Cycle Methodology

Key Points

- Best practices for test automation
- A Case Study will be presented that covers how the ATLM was implemented on one particular project. This case study will address each phases of the Automated Testing Life-cycle. Students can bring their own project specific problems, which can be addressed during the tutorial.
- Acquiring management support
- Test tool evaluation and selection
- The automated testing introduction process
- Various tools used during the various life-cycle phases
- Automated and manual test planning and preparation
- Test procedure development guidelines
- Automation reuse analysis and reuse library

Presentation Abstract

Automating Software Testing: A Life-Cycle Methodology
This tutorial outlines the Automated Test Life-cycle Methodology, a structured process for designing, developing, executing and managing testing that parallels the System Development Life-cycle. It is based on the book titled "Automated Software Testing" co-authored by the instructor and published by AWL, ISBN 0-201-43287-0. Automated Testing Life-Cycle Methodology

How test teams introduce an automated software test tool on a new project is nearly as important as the selection of the most appropriate test tool for the project. A tool is only as good as the process being used to implement the tool. Over the last several years test teams have largely implemented automated testing tools on projects, without having a process or strategy in place describing in detail the steps involved in using the test tool productively. This approach commonly results in the development of test scripts that are not reusable, meaning that the test script serves a single test string but cannot be applied to a subsequent release of the software application. In the case of incremental software builds and as a result of software changes, these test scripts need to be recreated repeatedly and must be adjusted multiple times to accommodate minor software changes. This approach increases the testing effort and brings subsequent schedule increases and cost overruns.

The fallout from a bad experience with a test tool on a project can have a ripple effect throughout an organization. The experience may tarnish the reputation of the test group. Confidence in the tool by product and project managers may have been shaken to the point where the test team may have difficulty obtaining approval for use of a test tool on future efforts. Likewise, when budget pressures materialize, planned expenditures for test tool licenses and related tool support may be scratched.

By developing and following a strategy for rolling out and implement an automated test tool as part of the Automated Testing Life-cycle methodology, the test team can avoid having to make major unplanned adjustments throughout the test process. The tutorial "Automated Software Testing" addresses these various issues and their solutions. The ATLM describes how and where "Automated Software Testing" fits into the system development life-cycle.

About the Author

Founder and President of Minjoh Technology Solutions,Inc (www.minjohtech.com), a company dedicated to providing quality Information Technology Solutions to government and commercial agencies. He has a BS degree in Computer Science and 12 years of professional software development experience. His experience extends through all phases of software development life cycle. He has been a speaker/presenter at various professional seminars and conferences.

He is also a co-author of the best-selling book "Automated Software Testing", published by Addison-Wesley Pub Co; ISBN: 0201432870, July 1999. The book has been getting excellent reviews throughout the testing community (see also www.amazon.com). The book has now been translated into German and is available in German. The book is currently being translated into Japanese and should be available in Japanese middle of 2001. For other details about the book see website at www.autotestco.com.

Automated Testing Life-cycle Methodology

Welcome

by

John Paul

President

Minjoh Technology Solutions, Inc.

http://www.minjohtech.com

Email: johnpaul@minjohtech.com

Welcome!

Some Housekeeping:

2:00 pm – 3:15 First Half Tutorial

3:15 pm – 03:30 pm Break

3:30 pm – 5:00 pm Second Half Tutorial

If you have a cell phone or beeper,
please turn it off / on vibrate

Goal for this tutorial:

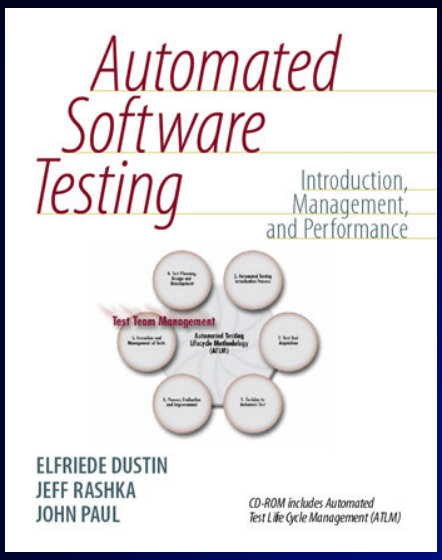
Apply information to your situation

How can we achieve this goal?

- Active Participation
- Ask yourself how each idea applies to your situation/your organization
- Prepare to disagree
- Develop follow-up items

Relax and enjoy yourself!

Book: Automated Software Testing



Book give away at the end of class….

http://www.autotestco.com

Quick Audience Survey:

What are you trying to get out of this tutorial?

What is your goal?

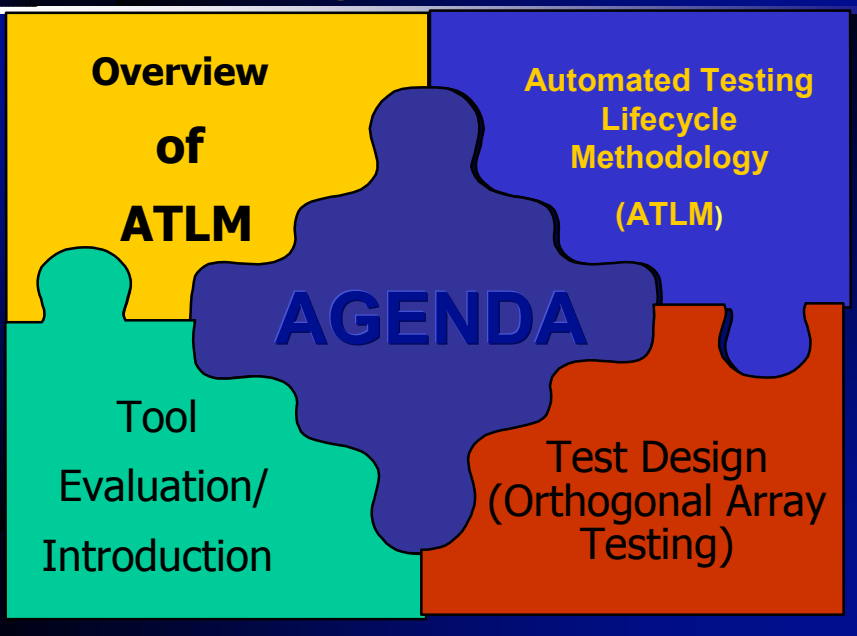
Explain in 30 seconds or less..

Quick Audience Survey:

Show of Hands:

- Use of Automated Tools? Y/N
- Mercury, Segue, Rational, RSW, Compuware, other?
- Environment – Windows, Unix, Mainframe, other?
- Test Experience – in years?

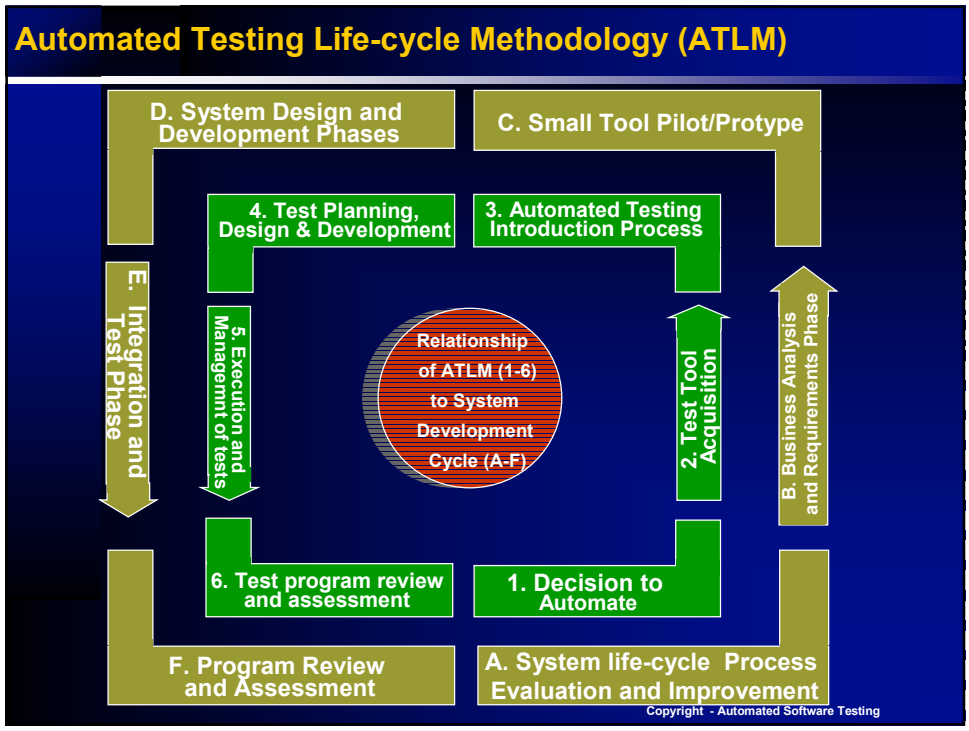
AGENDA

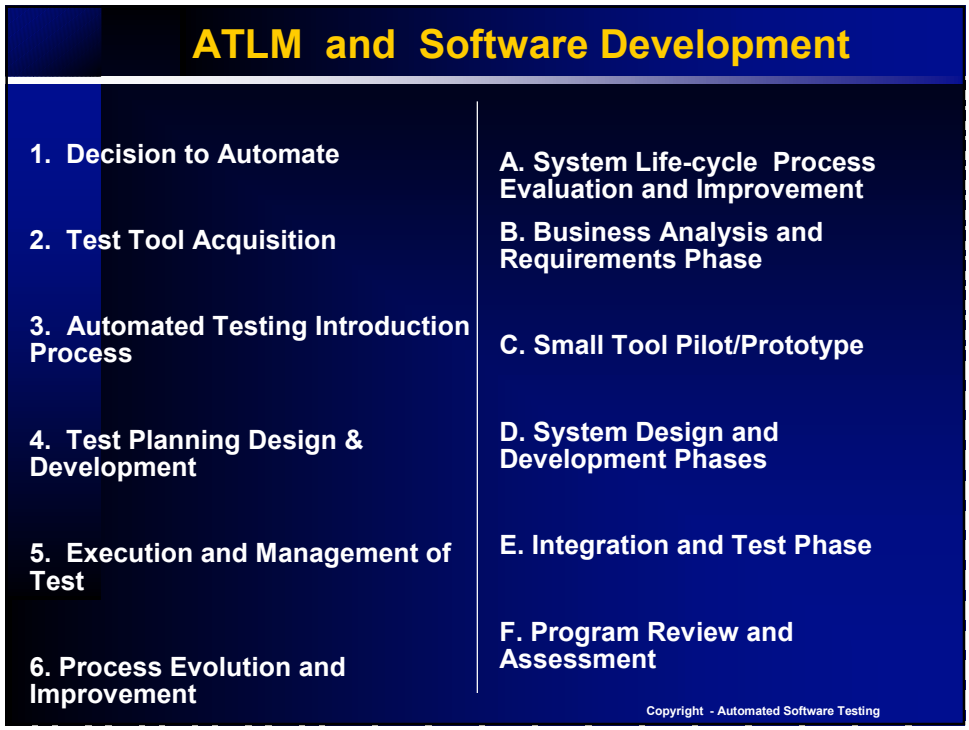


NASA Gives Up Search for Lost Mars Lander

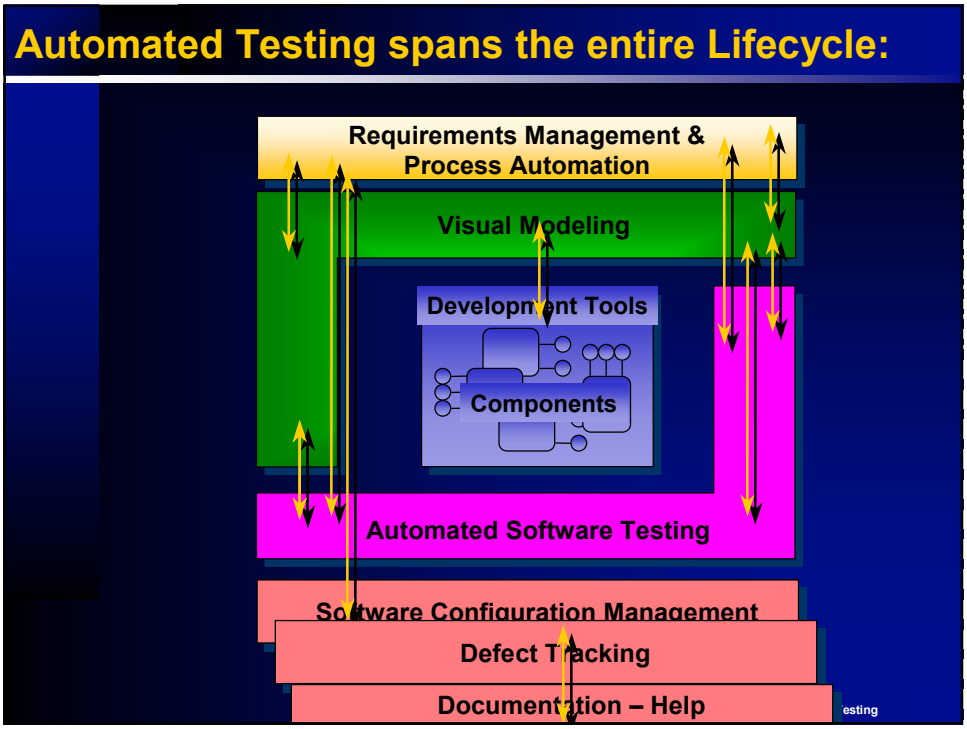
Message:

Defects are serious!

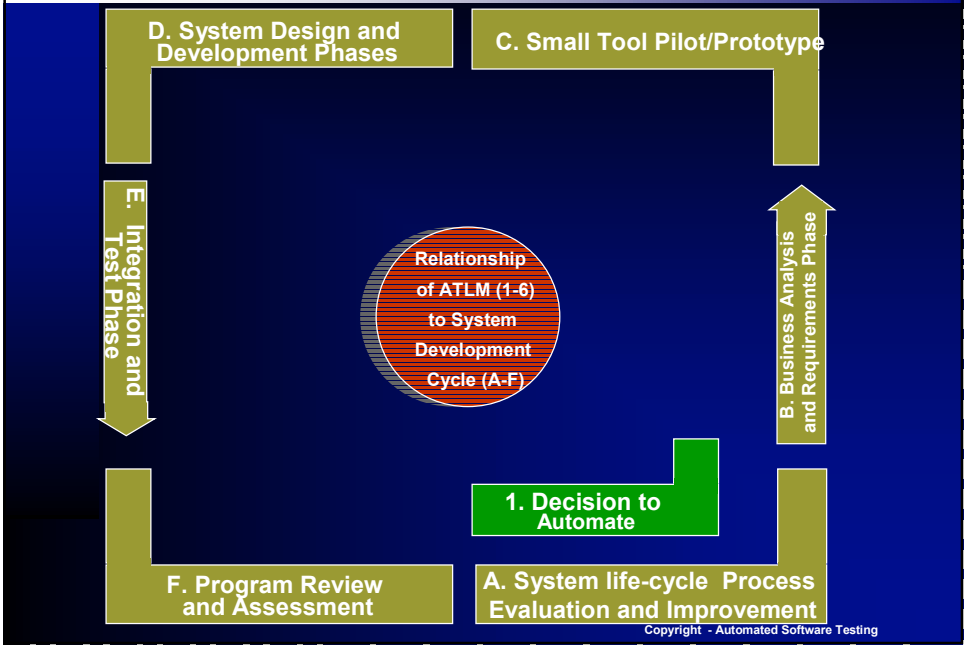

Automated Testing Life-cycle Methodology (ATLM)
D. System Design and Development Phases
C. Small Tool Pilot/Protype
4. Test Planning, Design & Development
3. Automated Testing Introduction Process
E. Integration and Test Phase
5. Execution and Managemnt of tests
Relationship of ATLM (1-6) to System Development Cycle (A-F)
2. Test Tool Acquisition
B. Business Analysis and Requirements Phase
6. Test program review and assessment
1. Decision to Automate
F. Program Review and Assessment
A. System life-cycle Process Evaluation and Improvement
Copyright - Automated Software Testing


ATLM and Software Development
1. Decision to Automate
2. Test Tool Acquisition
3. Automated Testing Introduction Process
4. Test Planning Design & Development
5. Execution and Management of Test
6. Process Evolution and Improvement
A. System Life-cycle Process Evaluation and Improvement
B. Business Analysis and Requirements Phase
C. Small Tool Pilot/Prototype
D. System Design and Development Phases
E. Integration and Test Phase
F. Program Review and Assessment
Copyright - Automated Software Testing

Automated Testing spans the entire Lifecycle:



Automated Testing Life-cycle Methodology (ATLM)

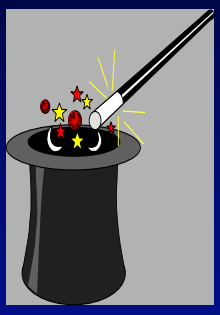


Exercise: Decision to Automate (Part I)

- What expectations do you have for automated testing?
- What expectations does your management have?
- Discuss!

Decision to Automate – Common Misconceptions

- Overcoming False Expectations for Automated Test
  - — Automatic Test Plan Generation
  - — Test Tool Fits All
  - — Immediate Test Effort reduction
  - — Immediate Schedule Reduction
  - — 100% Test Coverage

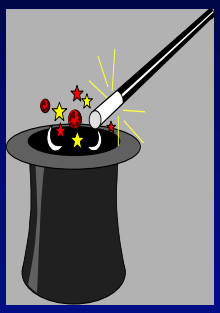


Development Life-cycle is in:

System Life-cycle Process Evaluation and Improvement

Decision to Automate – Common Misconceptions

- Overcoming False Expectations for Automated Test (continued)
  - — Universal Application of Test Automation



Development Life-cycle is in:

System Life-cycle Process Evaluation and Improvement

Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

>Production of a reliable System
>Improvement of the Quality of the Test Effort
>Reduction of Test Effort and Minimization of Schedule

Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

— **Production of a reliable System**
  —Improved requirements definition
  —Improved performance testing
  —Improved memory leak detection
  —Quality Measures and Test Optimization

Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

— **Production of a reliable System (cont)**
  —Improved partnership with development team
  —Improved system development life-cycle

Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

— **Improvement of the Quality of the Test Effort**
  —Improved Build Verification Testing (Smoke Test)
  —Improved Regression Testing
  —Improved Multi-platform Compatibility Testing
  —Improved Software Compatibility Testing
  —Improved Execution of Mundane Tests

Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

— **Improvement of the Quality of the Test Effort (cont)**
  —Improved Focus on Advanced Test Issues
  —Execution of Tests that Manual Testing can't accomplish
  —Ability to reproduce software defects
  —Enhancement of business expertise
  —After-hours standalone testing

Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

Reduction of the Test Effort and Minimization of Schedule

—Test Plan Development - Test Effort Increase
—Test Procedure Development - Test Effort Decrease
—Test Execution - Test Effort/Schedule Decrease
—Test Results Analysis - Test Effort/Schedule Decrease
—Error Status/Correction Monitoring - Test Effort/Schedule Decrease
—Report Creation - Test Effort/Schedule Decrease

Decision to Automate (pg 52)

- Case Study - Value of Test Automation Measurement

	Preparation **V**		Execution **D**		**N**	Expenditure **E** for **n** aut. tests:			
Test	**m**anual	**a**uto	**m**anual	**a**uto		1	5	10	20
Test 1	16	56	24	1	1.74	143%	45%	26%	15%
Test 2	10	14	2	0.1	2.11	118%	73%	50%	32%
Test 3	10	16	4.5	0.2	1.40	112%	52%	33%	20%
Test 4	20	28	1.5	0.2	6.15	131%	105%	86%	64%
Test 5	10	15	1	0.1	5.56	137%	103%	80%	57%
Test 6	10	15	1.5	0.1	3.57	131%	89%	64%	43%
Test 7	10	11.5	0.75	0.1	2.31	108%	87%	71%	54%
Test 8	10	11.5	0.5	0.1	3.75	110%	96%	83%	68%
Test 9	10	14	3	0.1	1.38	108%	58%	38%	23%
Test 10	10	10.6	0.5	0.1	1.50	102%	89%	77%	63%
Total	116	191.6	39.25	2.1	2.03	125%	65%	42%	26%

Result:
The measurements undertaken within these experiments show that a break-even point can already be attained by the 2nd regression test cycle (Ntotal =2.03).

Decision to Automate (pg 52)

•Value of Test Automation (after the fact)

Value of automated testing also has to be measured based on
- Bugs found
- Bugs not found – because the testers were too busy automating

Decision to Automate (cont'd)

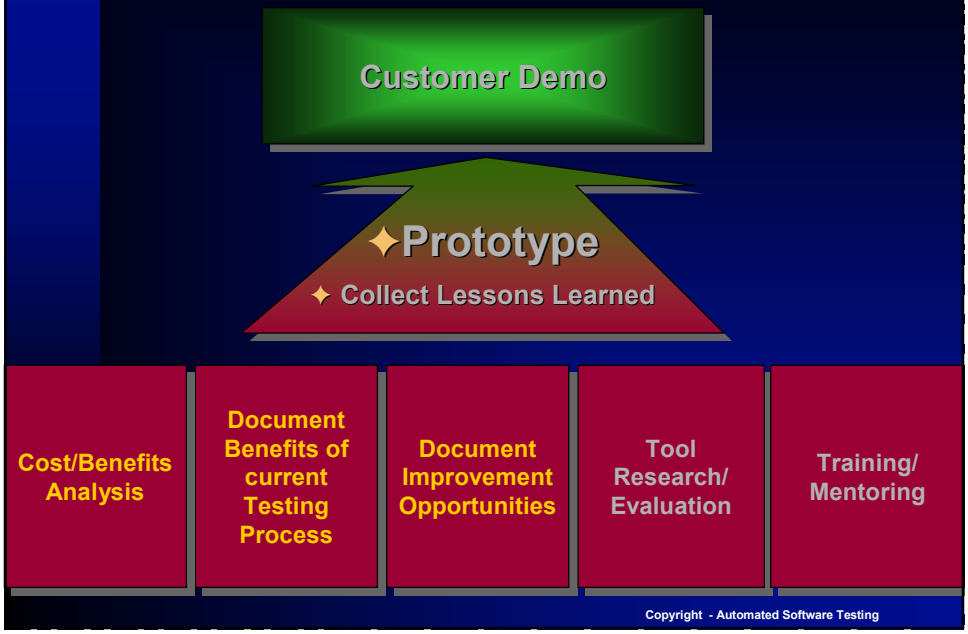
- Outline Benefits of Automated test
  - Production of a reliable System
  - Improvement of the quality of effort
  - Reduction of Test Effort and Minimization of Schedule
- Acquiring Management Support

Decision to Automate (cont'd)

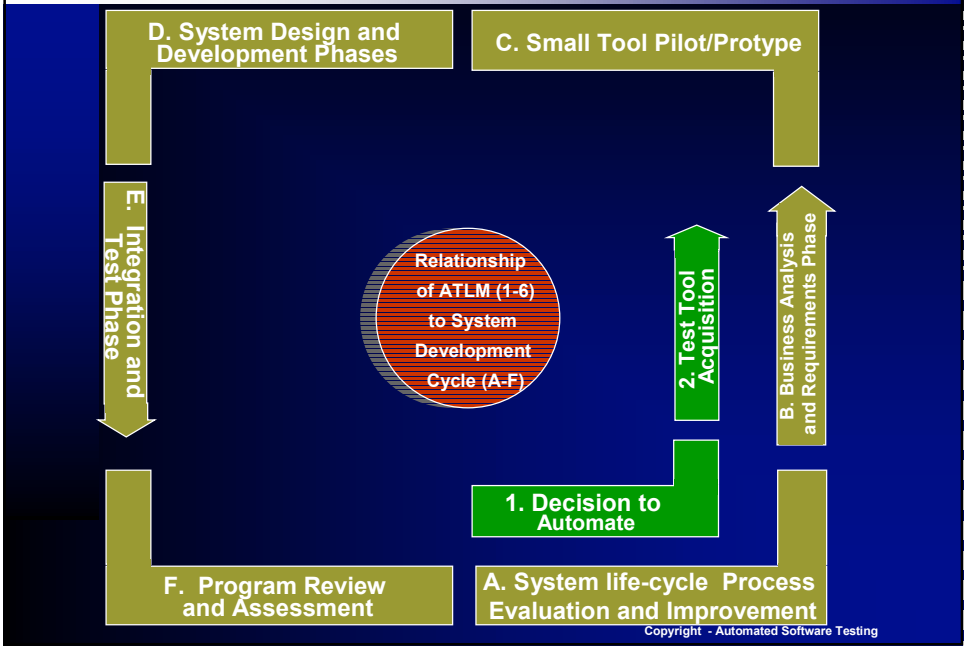
Acquiring Management Support

- Test Tool Proposal
  - Estimate the improvement opportunities
  - Approach for selecting the correct tool
  - Tool cost range
  - Additional time to introduce tool
  - Tool expertise
  - Tool training cost
  - Tool evaluation domain
  - Tool rollout process

Case Study: Decision to Automate/Tool Acquisition



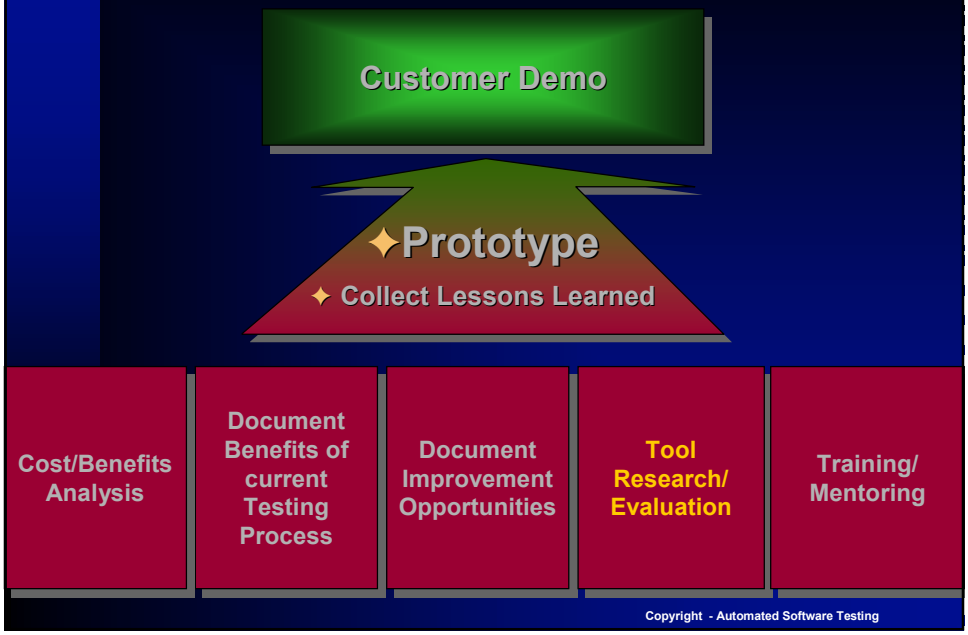
Automated Testing Life-cycle Methodology (ATLM)



Exercise – Brainstorming (2 – 3 min)

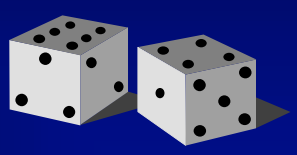
- Select a project on which you are currently working, or with which you might be working on in the future.
- List the most important factors to be considered when choosing a tool

Case Study: Decision to Automate/Tool Acquisition



Test Tool Acquisition

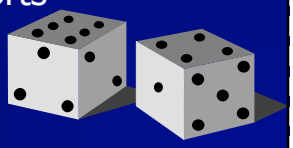
- Review System Engineering Environment
- Review Tools Available on the market
- Tool Research and Evaluation
  - — Reviewing the Test Life-Cycle Tools
- Tool Purchase



Development Life-cycle is in:
Business Analysis and Requirements Phase

Test Tool Acquisition

- **Review System Engineering Environment**
  - Gather Third-Party Input from Management, Staff, End-Users
  - Choose Tool Criteria Reflecting the System Engineering Environment
  - Define Level of Software Quality
  - Review Help Desk Problem Reports

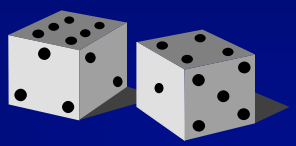


Development Life-cycle is in:

Business Analysis and Requirements Phase

Test Tool Acquisition

- **Review System Engineering Environment (cont)**
  - Budget Constraints
  - Types of Tests
  - Long-Term Investment Considerations
  - Test Tool Process
  - Avoid Shortcuts



Development Life-cycle is in:

Business Analysis and Requirements Phase

Test Tool Evaluation - Case Study

High-level Test Tool Selection Objectives:

- Vendor has sufficient market share to ensure vendors potential to stay in business
- Supports Web (and possibly VC++, VB) applications
- Provides automated test execution and results logging as well as reporting.
- Is compatible with Windows 2000, DHTML, HTML, Javascript, XML, and Java Applets
- Supports all testing phases and testing in the web environment (and desktop app written in VC++ ,VB):

Test Tool Evaluation (cont):

Detailed test feature and capability assessment

- Test Development Capability (Capture/Playback)
- Test Execution Capability
- Test Tool Integration Capability
- Test Reporting Capability
- Load/Stress Test Capability

• Cost Model

Test Tool Evaluation – A Case Study

- Tool xyz worked fine on one project with html, java, etc.
- New company, new project: dhtml – we actually discovered a bug with the tool –still waiting on the patch…. – had asked for prototype over and over……
- If buying multiple tools, make sure they install correctly and work together correctly
- Example: Micron computer needed BIOS upgrade – ReqPro didn't work

Very important: Evaluate tool in your environment

Test Development Capability (Capture/Playback)

	Weight	Rank (1 – 5)			
Criteria/Feature		**Rational**	**Segue**	**Mercury**	**RSW**
Test Language features					
Low Tool Learning Curve	7	2	3	2	5
Supports Windows 2000	10	5	5	5	5
Support for HTML, Dynamic HTML and Java Script	10	5	5	5	5
Supports Java Applets	10	5	5	5	5
Supports our report writer (selection not finalized….)	10				
Test Editor/Debugger Feature	8	5	5	5	0
Ease of Maintenance of Testbed	8	5	5	5	5
Support for Custom VC++ GUI Objects and embedded Stingray grids	10	0	0	5	0
Test Language DB Support					
Support ANSI SQL execution	7	5	5	5	0
Score		**279**	**286**	**329**	**225**

Test Execution Capability

Criteria/Feature	Weight (1 – 10)	Rank (1 – 5)			
		Rational	Segue	Mercury	RSW
Test Control features					
Centralized execution and control	10	4	4	4	4
Standalone Test Execution Automation	10	4	4	4	4
Distributed Test Execution					
Distributed Test Control, Synchronization, Execution	10	4	4	4	4
Support Synchronization of Multi Test Threads	10	4	4	4	4
Headless Backend Server Testing	10	4	4	4	4
Multi-Platform Testing Support	10	4	4	4	4
Test Suite Recovery Logic					
AUT State Management	10	4	4	4	4
Unexpected Error recovery	10	4	4	4	4
Test Management					
Allows for tracking of manual and automated test cases	8	1	1	4	0
Score		**328**	**328**	**352**	**320**

Test Tool Integration Capability

Criteria/Feature	Weight (1 – 10)	Rank (1 – 5)			
		Rational	Segue	Mercury	RSW
Interface with Rational Rose	5	4	0	4	0
Interface with Test Management Tool	5	4	4	4	0
Interface with Requirements Management Tool	5	4	0	4	0
Interface to defect tracking tool	5	4	4	4	0
Interface to configuration management tool	5	4	0	4	0
Score		**100**	**40**	**100**	**0**

Test Reporting Capability

	Weight (1 – 10)	Rank (1 – 5)			
Criteria/Feature		**Rational**	**Segue**	**Mercury**	**RSW**
Summary Level Reporting					
Error Filtering / Review Features	8	4	4	4	4
Metrics collections and presentations	8	4	4	5	4
Test Report Presentation					
Generate Graphs and Reports from Test Results	8	4	4	4	4
Reports Exportable to S/Excel	8	4	4	4	4
Score		**128**	**128**	**136**	**128**

Load, Stress and Performance Capability

	Weight (1 – 10)	Rank (1 – 5)			
Criteria/Feature		**Rational**	**Segue**	**Mercury**	**RSW**
Load and Stress Test Features					
Is tool non-intrusive	10	5	5	5	5
Support Microsoft Scripting (i.e. proxy java applets)	10	5	5	5	5
Support headless virtual user testing feature	8	5	5	5	5
Requires low overhead for VU	8	1	5	5	4
Scales to 500-1000 virtual users	5	3	5	5	4
Simulated IP Addresses for virtual Users	8	5	5	5	5
Centralized Load Test controller	8	5	5	5	5
Reused Scripts from Functional Test Suite	8	5	5	5	5

Load, Stress and Performance Capability (cont)

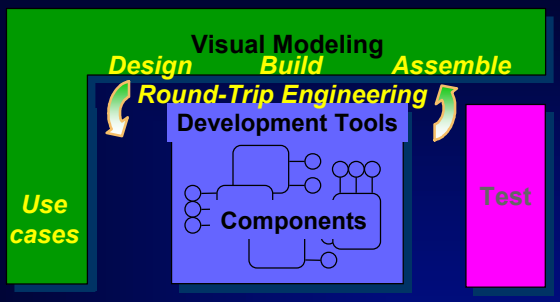
	Weight (1 – 10)	Rank (1 – 5)	5	4	4
Criteria/Feature		**Rational**	**Segue**	**Mercury**	**RSW**
Support for VC++ /embedded Stingray grids	10	0	0	0	0
Supports SSL recording	8	5	5	5	5
Support for Windows 2000	10	5	5	5	5
Performance Monitor Test Features					
Monitors different tiers (i.e. monitors web server, db server and app server separate)	8	5	5	5	5
Allows for monitoring of deployed application	8	0	5	5	4
Consulting Requirements					
Tech Support	8	4	1	4	4
No consulting needed	8	1	1	1	5
Score		**533**	**592**	**589**	**650**

Cost Model

Tool Suite includes	# of lic	Rational List	Segue List Price	RSW List Price	Mercury List	Cost for additional tools	
Test Management	5	yes	no	no	yes	N/A (see RM)	
Requirements Management (RM)	4	yes	no	no	no	RP $14,000	
Defect Tracking	4	yes	no	no	yes (5)	D (3) $29,500 VLS SF $13,500	
Configuration Management	4	yes	no	no	no	CQ $14,500	
Memory Leak Detection	1	yes	no	no	no	BC ~$818	
Capture/Playback	5	yes	3 licenses	1 license	yes		
Load/Stress Testing 100 VU	1	yes	250VU	yes	yes		
Maintenance (1 year)		Robot support	yes	yes	yes		
4 days on-site consulting		no	yes	no	no		
		80,811.90	65,000	**25,864.65**	67,019.00		

Key: A Well-Designed Architecture

- **Rational Rose**
- Support for ActiveX, Java, Corba
- Support for UML

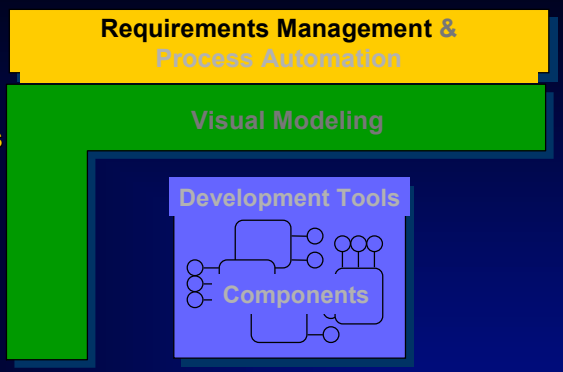


Key: Controlled Iterative Development Process

- **RequisitePro**

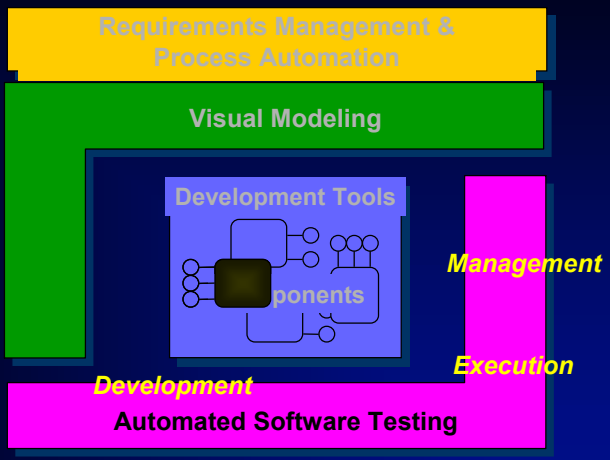
Or DOORS

- Organizes, tracks & controls requirements
- Reduces cost and risk



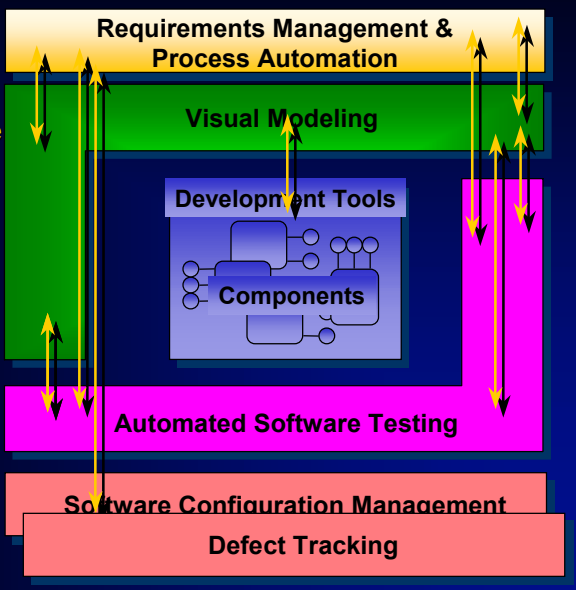
Key: Automated Testing

- **Robot**
- Client/server & web functional testing
- Leading ActiveX testing
- **•Winrunner**
- VC++, embedded Stingray grids



Integrated Suite of Tools - Enterprise wide

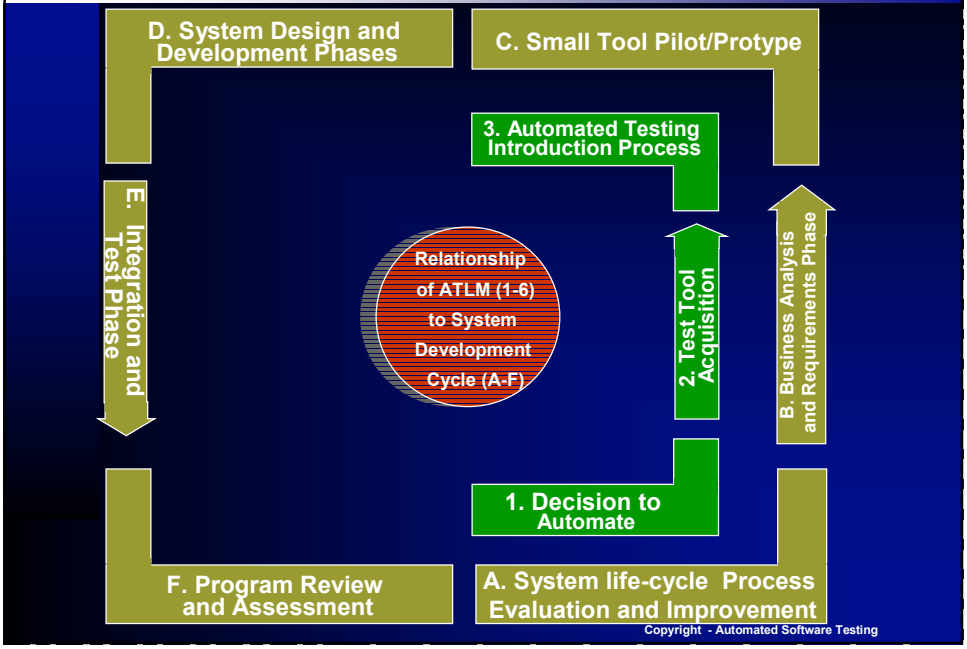
- DT - Visual Intercept (integrates with VS) – by Elsinore
- CM - Visual Source Safe (integrates with VS)



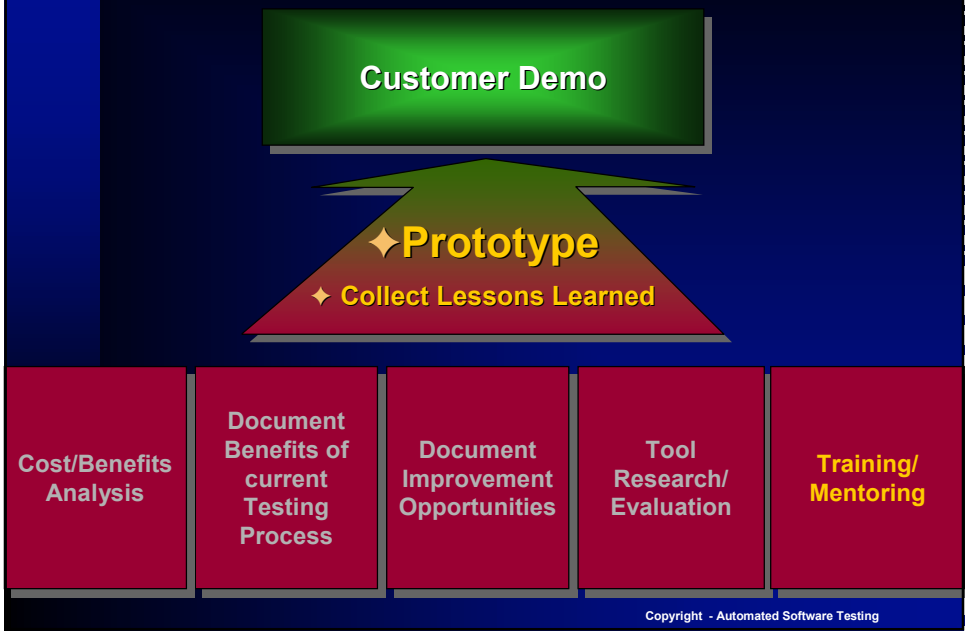
Load Testing Tools

- Conduct Load Testing using Load Testing Tool
- Example: Performance Studio/Loadrunner
- Reuse Scripts developed using Capture Playback tool - not VU users
- Other example: MS WAS free load testing tool

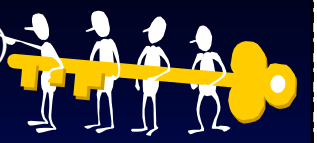
Automated Testing Life-cycle Methodology (ATLM)



Case Study: Decision to Automate/Tool Acquisition



Automated Testing Introduction Process



- Test Process Analysis (pg 110)
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management

Development Life-cycle is in:
Small Tool Pilot/Prototype

Testing Process Analysis

Does process meet these defined prerequisites?

- Clear Goals
- Objectives
- Methodology
- Strategy
- Is the testing process communicated and visible?
- Resource Commitment from Management
  - **Qualified Skillful People**
  - **Training $$**
  - **Time**
- User Involvement

Project Failure Rates

•In 1997 American companies spent $250 to $300 BILLION for software projects

•$100 BILLION spent for canceled projects

•$45 BILLION spent on projects which significantly exceeded time and budget estimates

(Standish Group)

Top Reasons For Failure

- Incomplete requirements and specifications
- Changing requirements and specifications
- Lack of efficient testing

(Standish Group and other studies)

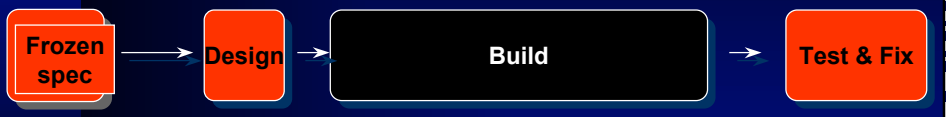
Lack of QA Process

Relative Cost To Fix An Error

Research by IBM, et. al.

Phase In Which Found	Cost Ratio
Requirements	1
Design	3-6
Coding	10
Development Testing	15-40
Acceptance Testing	30-70
Operation	40-1000

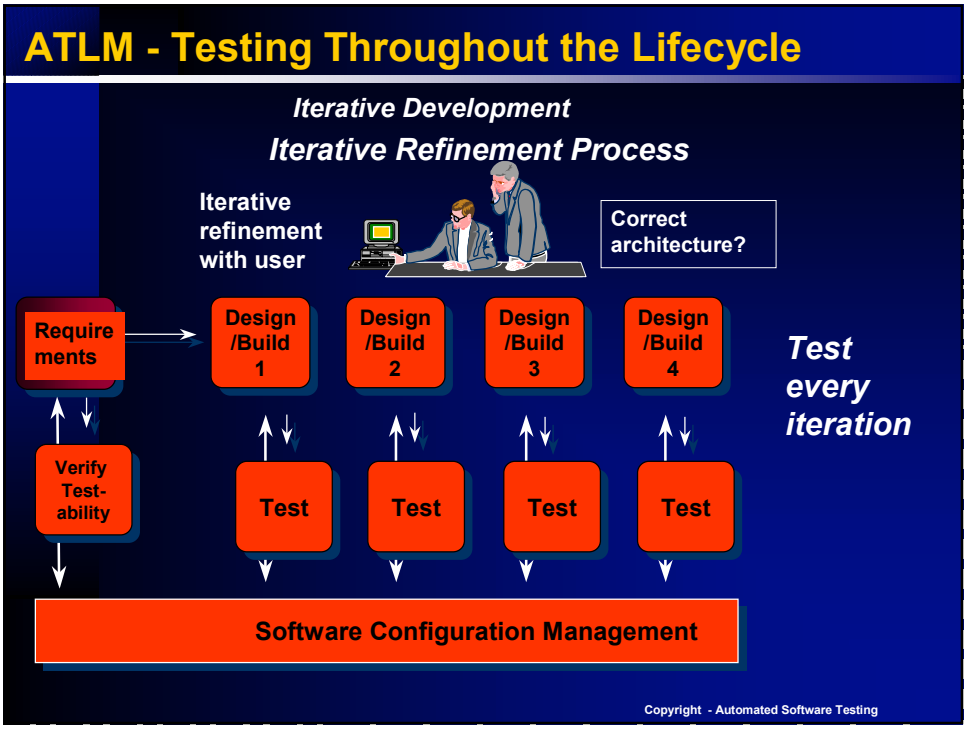
Conventional Testing Process:



ATLM - Testing Throughout the Lifecycle

Iterative Development

Iterative Refinement Process



Automated Testing Introduction

- System Testing needs to begin very early as requirements are being tested
- Criteria for quality requirements
  - Correctness, Completeness, Consistency, Testability

Requirements Management Tools – Case Study

Testers created….

- **~ 5000 Test Procedure Steps**

Requirements Management Tools (cont)

- ~ 500 Testable Requirements
- ~ 5000 Test Procedure Steps
- Imagine, creating a traceability matrix manually!

Requirements Management Tools (cont)

Imagine...

- Monitoring Progress of Test Procedure Execution (5000 Test Procedure Steps)
  - Use RM tool

Requirements Management (RM) Tools

Tool not only used for RM but for TM

- Test Procedures located in one central repository
- Multiple Testers can be assigned a section of functionality of system to write test procedures
- Multiple Testers can access a tool simultaneously without affecting anyone else (allows for locking)
- History of updates is maintained in RM tool (who, what, when)

Automated Testing Introduction Process

- Test Process Analysis
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management

Automated Testing Introduction Process

- Test Process Analysis
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management

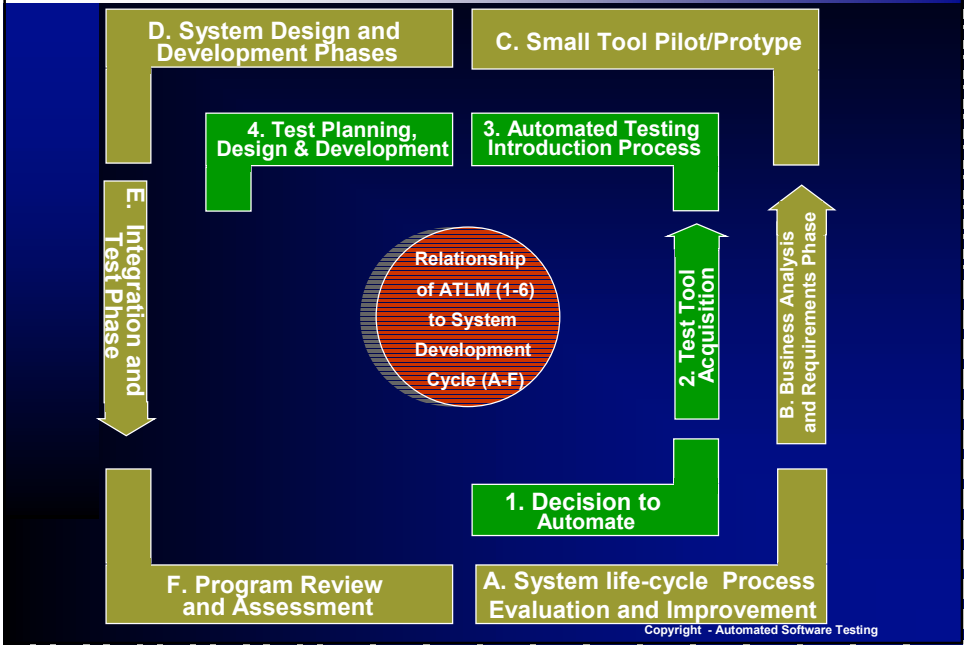
Exercise

- In your opinion what makes a good tester?
- List 2 – 3 points

Automated Testing Introduction Process

- Test Process Analysis
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management

Automated Testing Life-cycle Methodology (ATLM)



Exercise - Brainstorming

- What information should be in a test plan?

Test Planning Design & Development

- Test plan documentation (pg 219) - CD
  - — Test Program Scope
  - — Test Requirement Management
  - — Test Environment

- Test requirements Analysis
  - — Development-Level Test Analysis
  - — System-Level Test Analysis

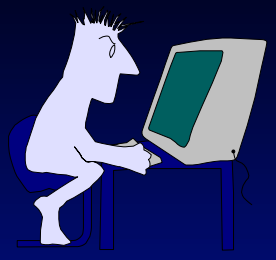
Test Planning Design & Development

- Test plan documentation
- Everything discussed so far needs to be in the test plan
- Goals, Objectives, Strategies, Roles and Responsibilities, Test Environment
- Take a look at test plan sample

Caveat: It's not always feasible to come up with an extensive test plan

Test Planning Design & Development

- Test plan documentation
  - Test Program Scope
  - Test Requirement Management
  - Test Environment



- Test requirements Analysis
  - Development-Level Test Analysis
  - System-Level Test Analysis

Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - **Black-Box Techniques (System-Level Tests)**
  - **Test Design Documentation**
  - Test procedure definition
  - **Automated vs Manual Test Analysis**
  - **Automated Test design standards**

Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - Black-Box Techniques (System-Level Tests)
  - Test Design Documentation
  - Test procedure definition
  - Automated vs Manual Test Analysis
  - Automated Test design standards

•Black-Box Techniques (System-Level Tests)

Functional Testing

- Main concepts
  - Based on requirements
  - Based on use cases
  - Discussed later…..

•Black-Box Techniques (System-Level Tests)

Boundary Value Testing

- Main concepts
  - Errors congregate at the boundaries between valid and invalid input
  - Tests using boundary values are highly effective
  - Tests of both valid and invalid input are needed
  - Closely related to equivalence partitioning

•Black-Box Techniques (System-Level Tests)

Equivalence

- Values that are on the same side of a boundary are members of the same equivalence class
- If 99 is valid and 100 is not valid, 101 will also be invalid, as will 102, 103,…
- No point to testing many members of the same equivalence class

•Black-Box Techniques (System-Level Tests)

Risk analysis

- Not in itself a testing technique, but a way to prioritize techniques and test cases

Risk Prioritization

- 20% of test cases will uncover 80% of the problems

•Black-Box Techniques (System-Level Tests)

Exploratory Testing

- **Useful for testing with limited (or no) specification**
- **Allows the tester to write specifications and get developers or marketers to confirm or deny**

Case Study: Test Planning and Design

- Exhaustive testing becomes almost impossible, always very expensive
- Derive Test Cases using specific test coverage techniques

  for example:

  – Orthogonal Array Testing (OATS)

Case Study: Orthogonal Arrays

- Derived from IE manufacturing techniques
- For experimentation it allows the effects of several parameters to be determined efficiently[1]
- For testing it allows test parameter values to be determined efficiently & uniformly
- Is an important technique in Robust Design[1]

[1] Quality Engineering using Robust Design by M.S. Phadke

Case Study: Orthogonal Arrays (cont)

OATS Test Technique

- Purpose is to assist in the selection of appropriate combinations of factors to provide maximum coverage from a test procedure with a minimum of number of cases
- OATs selects test cases so as to maximize the interactions between independent measures (all pairwise combinations)

Case Study: Orthogonal Arrays (cont)

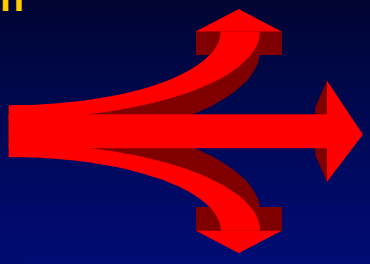
OATS Array Interpretation

- System parameters are array columns
  - Parameter values must be mapped to array number values
- Tester can chose valid values for unassigned levels in effort to minimize test
  - redundant test cases are eliminated
  - add cases based on known risk areas
- Each remaining row in the orthogonal array specifies one specific test case

Case Study: Orthogonal Arrays (cont)

Array Determination

- Max # of values or states for all parameters
- Number of parameters or factors

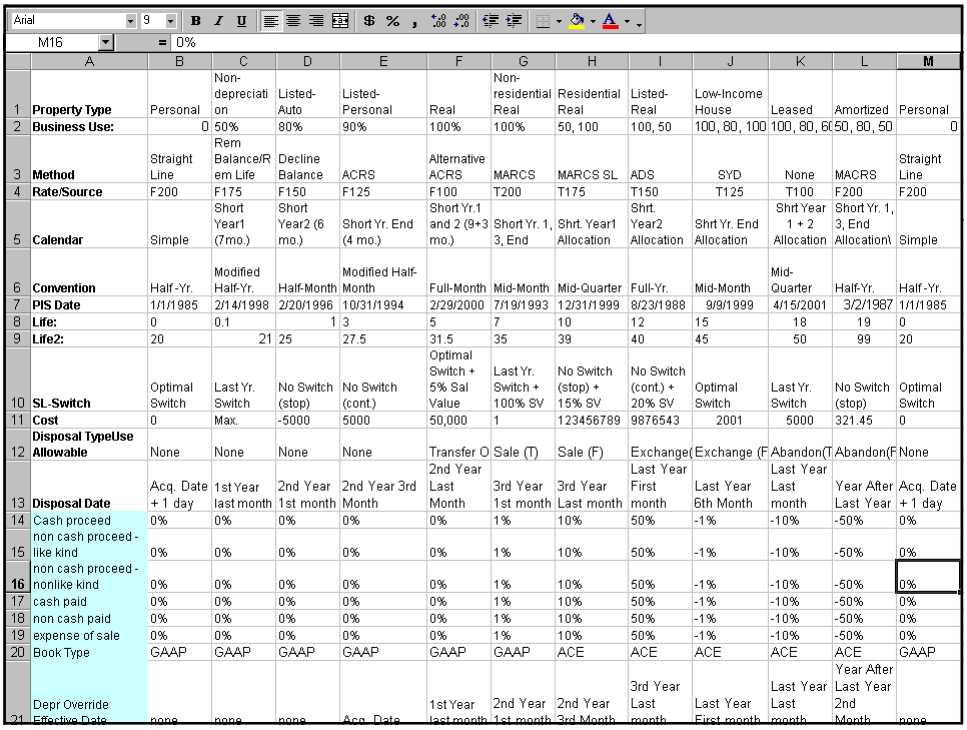


An Example Array

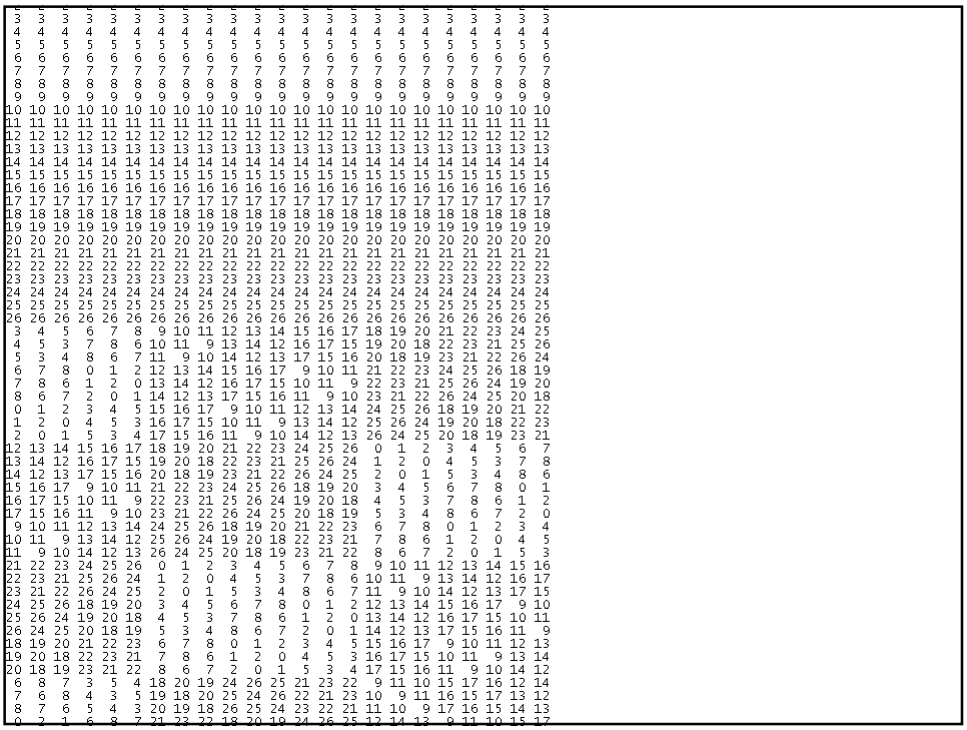
- Consider an example in which there are three factors and each factor has three levels.
- Testing all possible combinations of the three factors would require 27 test cases.
- But, testing pair-wise combinations will result in only 9 cases required to test the interactions of the independent factors

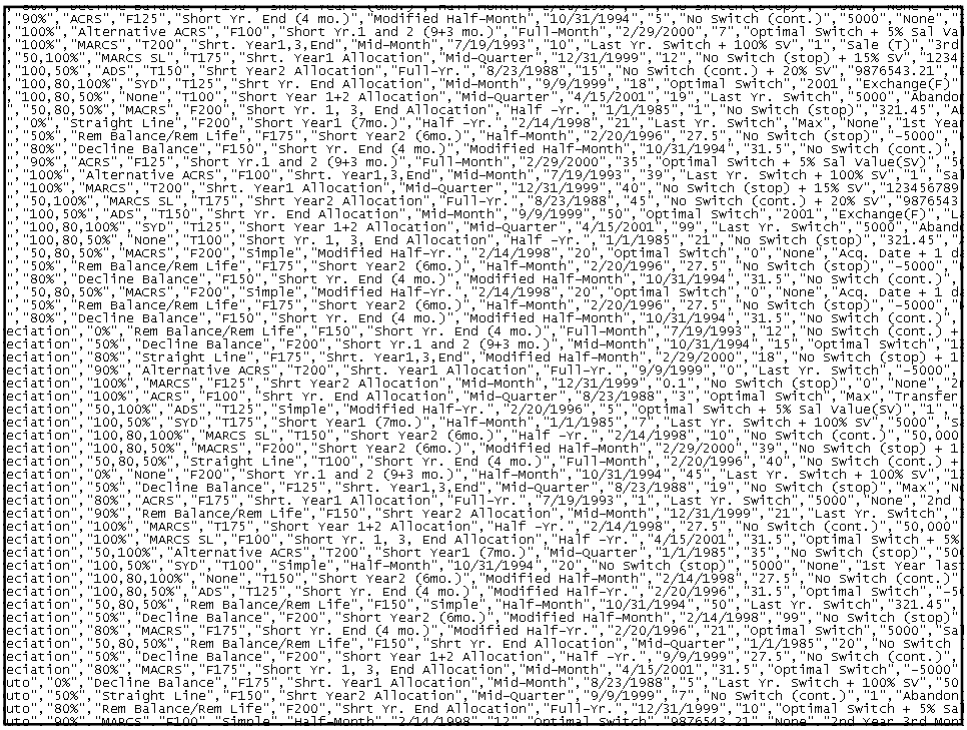
An Example Array

	A	B	C
1	1	1	3
2	1	2	2
3	1	3	1
4	2	1	2
5	2	2	1
6	2	3	3
7	3	1	1
8	3	2	3
9	3	3	2



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Property Type**	Personal	Non-depreciation	Listed-Auto	Listed-Personal	Real	Non-residential Real	Residential Real	Listed-Real	Low-Income House	Leased	Amortized	Personal
2	**Business Use:**	0	50%	80%	90%	100%	100%	50, 100	100, 50	100, 80, 100	100, 80, 6	50, 80, 50	0
3	**Method**	Straight Line	Rem Balance/Rem Life	Decline Balance	ACRS	Alternative ACRS	MARCS	MARCS SL	ADS	SYD	None	MACRS	Straight Line
4	**Rate/Source**	F200	F175	F150	F125	F100	T200	T175	T150	T125	T100	F200	F200
5	**Calendar**	Simple	Short Year1 (7mo.)	Short Year2 (6 mo.)	Short Yr. End (4 mo.)	Short Yr.1 and 2 (9+3 mo.)	Short Yr. 1, 3, End	Shrt. Year1 Allocation	Shrt. Year2 Allocation	Shrt Yr. End Allocation	Shrt Year 1 + 2 Allocation	Short Yr. 1, 3, End Allocation\	Simple
6	**Convention**	Half-Yr.	Modified Half-Yr.	Half-Month	Modified Half-Month	Full-Month	Mid-Month	Mid-Quarter	Full-Yr.	Mid-Month	Mid-Quarter	Half-Yr.	Half-Yr.
7	**PIS Date**	1/1/1985	2/14/1998	2/20/1996	10/31/1994	2/29/2000	7/19/1993	12/31/1999	8/23/1988	9/9/1999	4/15/2001	3/2/1987	1/1/1985
8	**Life:**	0	0.1	1	3	5	7	10	12	15	18	19	0
9	**Life2:**	20	21	25	27.5	31.5	35	39	40	45	50	99	20
10	**SL-Switch**	Optimal Switch	Last Yr. Switch	No Switch (stop)	No Switch (cont.)	Optimal Switch + 5% Sal Value	Last Yr. Switch + 100% SV	No Switch (stop) + 15% SV	No Switch (cont.) + 20% SV	Optimal Switch	Last Yr. Switch	No Switch (stop)	Optimal Switch
11	**Cost**	0	Max.	-5000	5000	50,000	1	123456789	9876543	2001	5000	321.45	0
12	**Disposal TypeUse Allowable**	None	None	None	None	Transfer O	Sale (T)	Sale (F)	Exchange(	Exchange (F	Abandon(T	Abandon(F	None
13	**Disposal Date**	Acq. Date + 1 day	1st Year last month	2nd Year 1st month	2nd Year 3rd Month	2nd Year Last Month	3rd Year 1st month	3rd Year Last month	Last Year First month	Last Year 6th Month	Last Year Last month	Year After Last Year	Acq. Date + 1 day
14	Cash proceed	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
15	non cash proceed - like kind	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
16	non cash proceed - nonlike kind	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
17	cash paid	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
18	non cash paid	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
19	expense of sale	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
20	Book Type	GAAP	GAAP	GAAP	GAAP	GAAP	GAAP	ACE	ACE	ACE	ACE	ACE	GAAP
21	Depr Override Effective Date	none	none	none	Acq. Date	1st Year last month	2nd Year 1st month	2nd Year 3rd Month	3rd Year Last month	Last Year First month	Last Year Last month	Year After Last Year 2nd Month	none



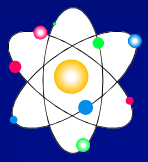


Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - Black-Box Techniques (System-Level Tests)
  - Test Design Documentation
  - Test procedure definition
  - Automated vs Manual Test Analysis
  - Automated Test design standards

Case Study: Test Planning and Design

- Evaluate what to automate (pg. 262)
  - Not everything can be/should be automated
  - Automated Tool Expertise is required



Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - Black-Box Techniques (System-Level Tests)
  - Test Design Documentation
  - Test procedure definition
  - Automated vs Manual Test Analysis
  - Automated Test design standards

Test Procedure Definition

TEST NAME: Installation Routine Testing

Date Executed: **Tester's Initials:**
Automated/Manual:
Test Procedure Writer: ED
Test Objective: Test the Installation Routine for ND and New Database Technology
Pre-Conditions/Assumptions:

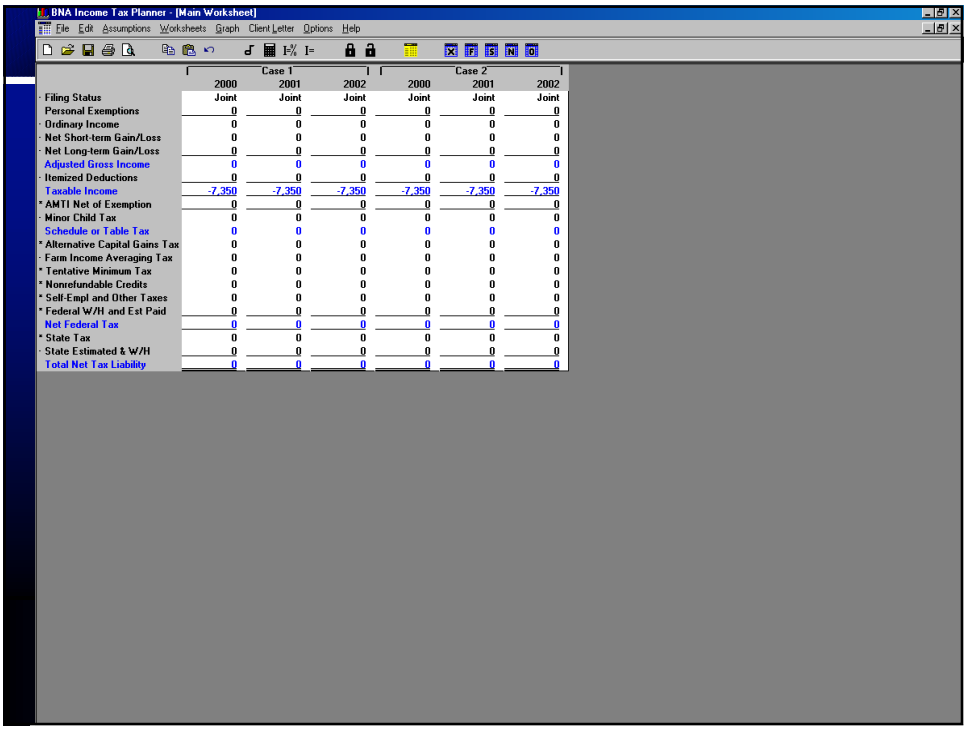
Repeat tests using following setup:

Test Procedure Definition

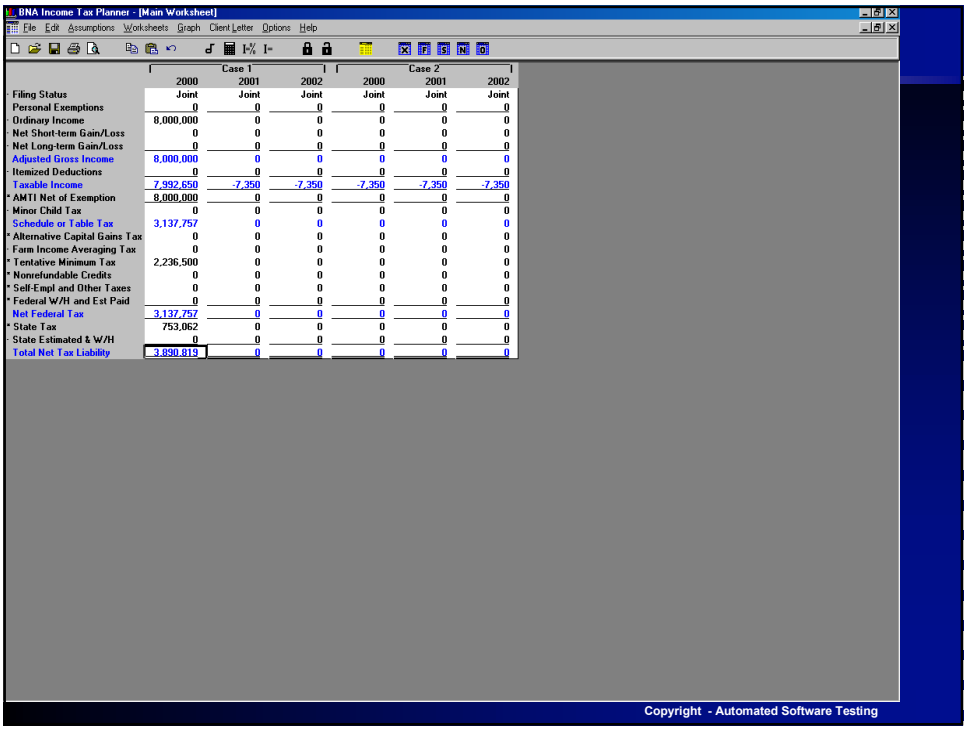
Setup	OS	Printers HP 3si, HP4, HP5, HP8100	Antivirus Software	Configuration		
.	Win 95	all		Clean Machine	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE inst
						w/o prior NDEE
				MSOffice +	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE inst
						w/o prior NDEE
.	Win98 2nd Edition	all	McAfee	Clean Machine	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE inst
						w/o prior NDEE
				MSOffice +	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE inst
						w/o prior NDEE
.	NT SP4	all	McAfee	Clean Machine	ND	prior ND install
						w/o prior ND

Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - Black-Box Techniques (System-Level Tests)
  - Test Design Documentation
  - Test procedure definition
  - Automated vs Manual Test Analysis
  - Automated Test design standards



	Case 1			Case 2		
	2000	2001	2002	2000	2001	2002
Filing Status	Joint	Joint	Joint	Joint	Joint	Joint
Personal Exemptions	0	0	0	0	0	0
Ordinary Income	0	0	0	0	0	0
Net Short-term Gain/Loss	0	0	0	0	0	0
Net Long-term Gain/Loss	0	0	0	0	0	0
Adjusted Gross Income	0	0	0	0	0	0
Itemized Deductions	0	0	0	0	0	0
Taxable Income	-7,350	-7,350	-7,350	-7,350	-7,350	-7,350
AMTI Net of Exemption	0	0	0	0	0	0
Minor Child Tax	0	0	0	0	0	0
Schedule or Table Tax	0	0	0	0	0	0
Alternative Capital Gains Tax	0	0	0	0	0	0
Farm Income Averaging Tax	0	0	0	0	0	0
Tentative Minimum Tax	0	0	0	0	0	0
Nonrefundable Credits	0	0	0	0	0	0
Self-Empl and Other Taxes	0	0	0	0	0	0
Federal W/H and Est Paid	0	0	0	0	0	0
Net Federal Tax	0	0	0	0	0	0
State Tax	0	0	0	0	0	0
State Estimated & W/H	0	0	0	0	0	0
Total Net Tax Liability	0	0	0	0	0	0



Capture/Playback records hard-coded values

- .....
- Window SetContext, "Name=frmMDI", ""
-
- Window SetContext, "Name=frmWsTemplate", ""
- GenericObject Click, "Name=Spread", "Coords=213,138"
-
- Window SetContext, "Name=frmMDI", ""
- GenericObject Click, "Class=MDIClient;ClassIndex=1", "Coords=194,474"
-
- Window SetContext, "Name=frmWsTemplate", ""
- GenericObject DblClick, "Name=Spread", "Coords=212,64"
- InputKeys "**8000000**"
- GenericObject Click, "Name=Spread", "Coords=297,58"

- End Sub

Capture/Playback records hard-coded values

- REPLACE HARD CODED VALUES WITH VARIABLES
- READ DATA FROM FILE
- PUT EXPECTED RESULTS INTO FILE
- PUT OPERANDS INTO FILE

Case Study: Test Planning and Design

- Goals of test procedure development is for scripts to be reusable, portable, maintainable.
  - Modularity
  - Data outside of test procedure (manual and automated)
  - Not written on detailed level

Test Planning Design & Development (cont'd)

- Test Development
  - — Set up Test Environment
  - — Automation Framework Reuse Analysis
  - — Test Procedure Development/Execution Schedule
  - — Calibration of the test tool
  - — Compatibility and work around solutions
  - —Test Procedure inspections and Peer reviews
  - — Test Procedure Configuration management
  - — Reusable Test procedures

Process Evolution and Improvement

- Post Release - Test Process Improvement
  - – Documenting Lessons learned
  - – What worked and what did not ?
  - – How would you do things differently ?
  - – Reviewing standards for future projects

Development life-cycle is in:
Test Program Review and Assessment

ATLM

ATLM will help sort out tool issues

- What tools are available
- How to convince management to buy the tool
- How to evaluate tools
- How to incorporate tools into project and how to manage automated testing process
- Automated Test Design, Development, Execution
- Lessons Learned

details described in book "Automated Software Testing"

ATLM - Summary:

- Automated Testing needs to be parallel to system development
- Evaluate Testing Tools based on your environment and circumstances
- Training, training, training
- Manage expectations

- **see www.stqemagazine.com (Sep/Oct '99) article on "Automated Testing Lessons Learned"**

Remember:

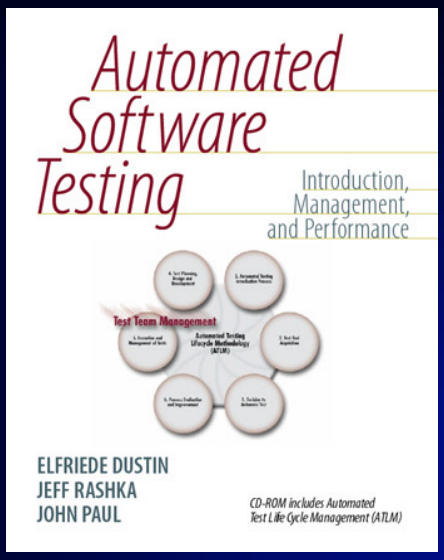
If you want a high quality software system, you must ensure each of its parts is of high quality

Watts Humphrey

- Discovery lands at the end of mission STS-60

Automated Testing Life-cycle Methodology

Book: Automated Software Testing



Lucky Winner Is…..

http://www.autotestco.com

QW2001 Workshop W3

Mr. Robert A. Sabourin
(AmiBug.Com)

Bug Priority And Severity

Key Points

- Bugs
- SQA Management
- SQA Process

Presentation Abstract

In this interactive half day tutorial the concepts of defect priority and severity are explored.

The fundamental question in software engineering "How do you know when you are finished?" is examined.

The journey begins on a freezing cold Canadian Winter day on an elevator ride during which Robert Sabourin accidentally overheads some strangers discussing problems with the most important software project taking place in the company! The class invited to help Robert in his job and to effectively define and identify characteristics of the problem, most importantly the priority and severity of the issue!

The concept of the four-quadrants of priority and severity are taught and the class is clearly shown how business factors influence the quadrant of a bug! The class includes a review of some actual defect arrival graphs from recent commercial product development efforts and provides an answer to the fundamental question of software engineering. Practical aspects of bug tracking, defect logging and bug review meetings are included.

About the Author

Robert Sabourin has been involved in all aspects of development, testing and management of software engineering projects. Robert graduated from McGill University in 1982. Since writing his first program in 1972, Robert has become an accomplished software engineering management expert. He is presently the President of AmiBug.Com, Inc.; a Montreal-based international firm specializing in software engineering and and software quality assurance training, management consulting and professional development. AmiBug helps companies set up software engineering and quality assurance teams and process through a combination of training and management consulting. Robert was the Director of Research and Development at Purkinje Inc where he was charged with developing

world class critical medical software used by clinicians at the point of care. Previously, Robert managed Software Development at Alis Technologies for over ten years. He has built several successful software development teams and champions the implementation of "light effective process" to achieve excellence in delivering on-time, on-quality, on-budget commercial software solutions.

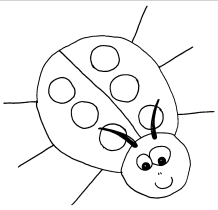
Robert has championed many complex international multilingual software development and globalization efforts involving several intricate business partnerships and relationships including international government (Czech, Egypt, France, Morocco, Algeria...) and commercial entities (Microsoft, IBM, AT&T, HP, Thompson CSF, Olivetti...). Systems included concurrent coordinated multilingual multiplatform product releases.

Robert's pioneering work with Infolytica Corporation led to the development of the first commercially available platform independent graphics standard GKS and several toolkits which allowed for cross platform development and porting of complex CAD, Graphics, Analysis and Non-Destructive Simulation systems.

Robert is a frequent guest lecturer at McGill University where he relates theoretical aspects of Software Engineering to real world examples with practical hands-on demonstrations.

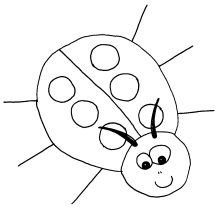
In 1999, Robert completed a short book illustrated by his daughter Catherine entitled"I Am a Bug" (ISBN 0-9685774-0-7).

Robert has received professional recognition for many accomplishments over the years. At TEPR 2000 - award for best electronic patient record product to EHS using the Purkinje CNC component. Byte Middle-East's 1992 Product of the Year for the AVT-710 product family achieving a ZERO FIELD REPORTED software defect rate with over 15,000 units installed. (Project involved over 27-man month's effort!); Quebec Order of Engineers' recognition for creating and managing the Alis R&D Policy Guide - Development Framework and process.



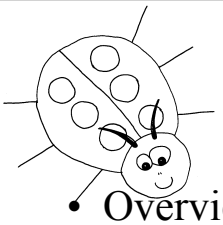
Bug Priority and Severity

Robert Sabourin
President
AmiBug.Com, Inc.
Montreal, Canada
rsabourin@amibug.com
www.amibug.com



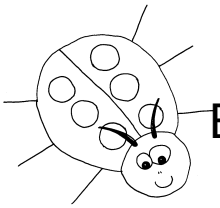
Bug Priority and Severity

Elevator Parable



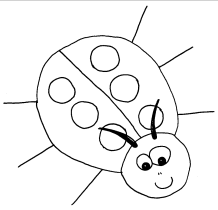
Bug Priority and Severity

- Overview
  - Introductions
  - Elevator Parable
  - Quadrants of priority and severity
  - Example definitions
  - Defect arrival curves and metrics
  - Practical aspects Track/Log/Review
  - Bug Reporting
  - Fundamental question of software engineering



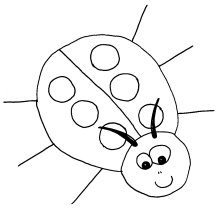
Bug Priority and Severity

- Robert Sabourin , *Software Evangelist*
- President
- AmiBug.Com Inc.
- Montreal, Quebec, Canada
- rsabourin@amibug.com



AmiBug.Com, Inc.

- Software Development & SQA Consulting
- Services
  - Training, Coaching and Professional Development
  - Light Effective Process
  - Team Building and Organization
  - We help people to get things done!



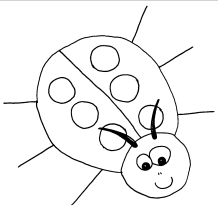
I am a Bug

Robert & Catherine Sabourin

ISBN: 0-9685774-0-7

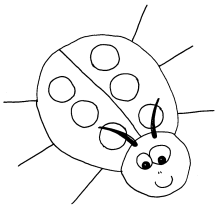
www.amazon.com
www.fatbrain.com

In the style of a children's book. Explains elements of software development process in a fun easy to read format.



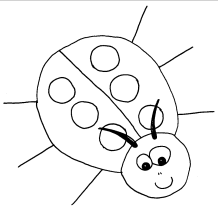
Fundamental Question

- How do you know when you are finished?



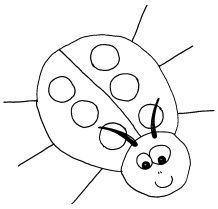
Crosby on Quality

- "Quality is defined as conformance to requirements"
- "Quality is not a measure of GOODNESS"
  - Phil B. Crosby, *Quality is Free*

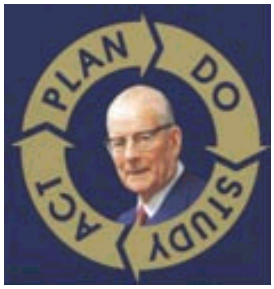


Dr. Edwards Deming

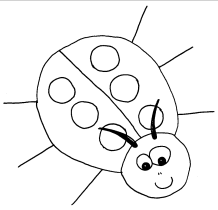
- "Management of quality needs quality management"



Deming Quality approach (PDCA)

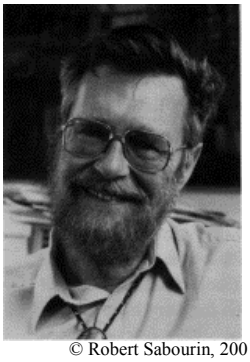


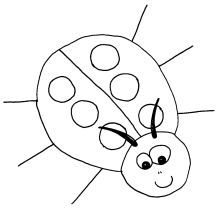
- Plan, Do Check, and Act:
  - ☑Plan what you want to implement.
  - ☑Do the pilot implementation.
  - ☑Check the results of the pilot.
  - ☑Act on the results by tweaking the process before the next project.



Edsger W. Dijkstra

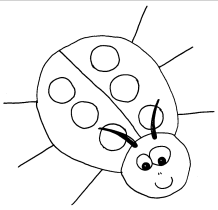
- "Program testing can be used to show the presence of bugs, but never to show their absence"





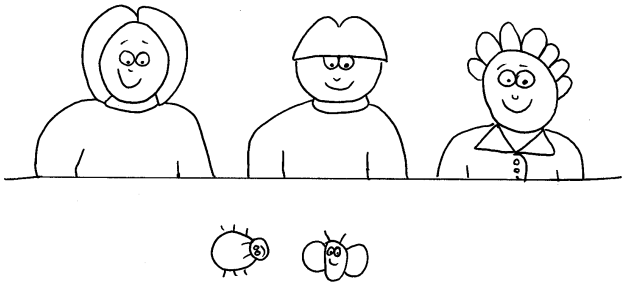
Definition of a Bug

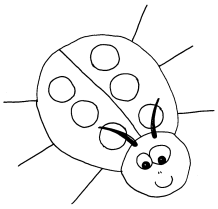
- To make our job more fun, whenever we have a *concern with software*, we call it a "bug".



Bug Priority and Severity

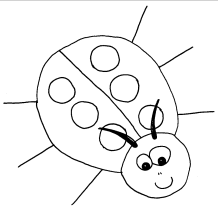
- It's all about people! (and the occasional bug too)





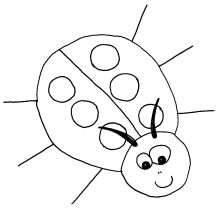
Purpose

- What is the purpose of testing?



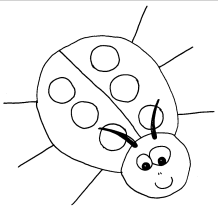
Purpose

- Common definition of the purpose of testing:
  - Our purpose is to find bugs before our customers do!



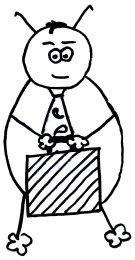
Purpose

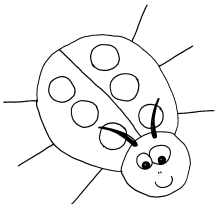
- Broader definition:
  - The role of testing is to provide objective input to facilitate business decisions (wise smart and good decisions)
  - keeps internal stakeholders aware of all the issues/concerns that relate to shipping a product



Microsoft® SDD Team Model

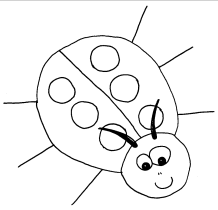
- Testing Role defined simply and clearly in the sense of SQA not corporate QA
  - Ensure all concerns are KNOWN to team
  - Develop testing strategy and plans
  - *FACILITATE BUSINESS DECISIONS*
  - *PROVIDE OBJECTIVE INFORMATION*



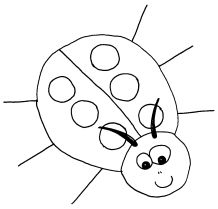


A note about parables

- Teaching
- Learning
- Retaining
- Applying knowledge
- Share experiences



The Elevator Parable

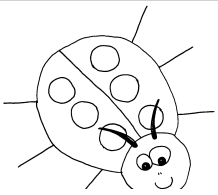


The Elevator Parable Weather in Montreal

Montreal, Canada

Month	Average high	Average low	Warmest ever	Coldest ever	Average dew point	Average precipitation
JAN.	21	7	52	-31	7	2.8
FEB.	24	10	59	-22	9	2.6
MARCH	35	21	70	-17	18	2.8
APRIL	51	35	84	9	31	2.9
MAY	65	47	90	25	43	2.7
JUNE	73	56	91	36	53	3.3
JULY	79	61	93	43	59	3.4
AUG.	76	59	95	39	58	3.6
SEP.	66	50	90	28	50	3.3
OCT.	54	39	79	19	38	3
NOV.	41	29	68	3	28	3.5
DEC.	27	13	59	-26	14	3.4

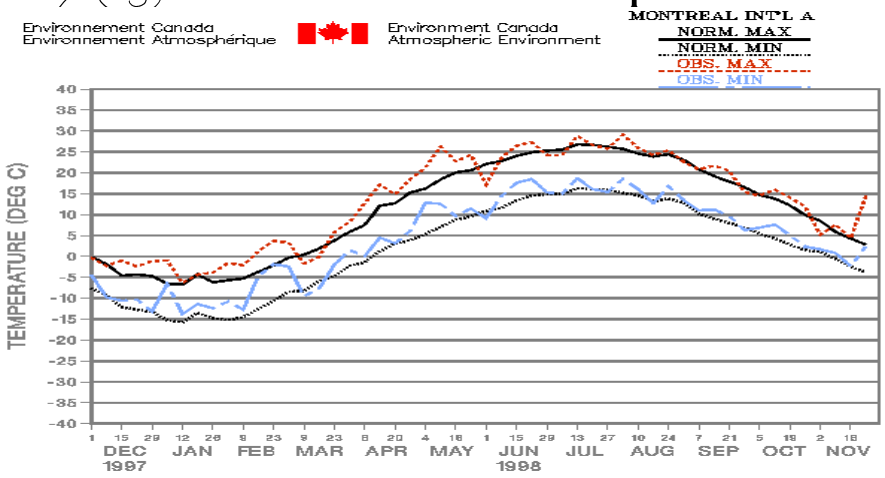
Latitude: 45 degrees, 28 minutes north
Longitude: 73 degrees, 45 minutes east

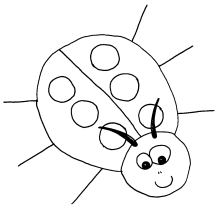


The Elevator Parable Montreal Temperature

Environnement Canada
Environnement Atmosphérique

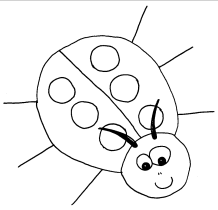
Environment Canada
Atmospheric Environment





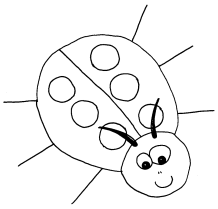
The Elevator Parable

- CNC
  - highest priority project, new business and technical model
- Profile
  - critical last minute feature requested by customer for CNC
- GPF
  - windows general protection fault, *crash*



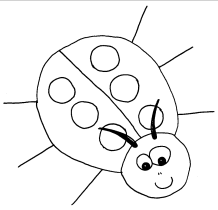
Bug Priority

- How important is it?
  - Urgent
  - Not Urgent



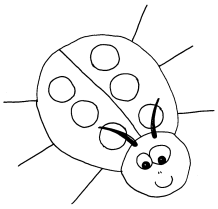
The Elevator Parable

- Define Priority Scheme
  - P1
    - _____
  - P2
    - _____
  - P3
    - _____



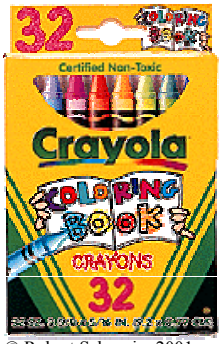
The Elevator Parable

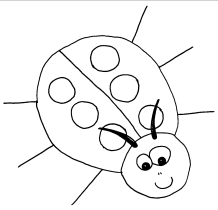
- Priority Scheme
  - P1
    - Fix it now
  - P2
    - Fix it later
  - P3
    - Do not fix it



Crayons

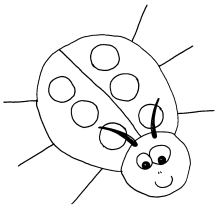
- Fun to draw pictures



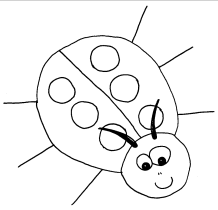


The Elevator Parable

Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	



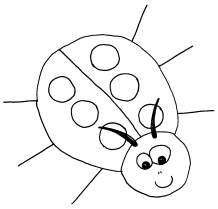
The Elevator Parable



The Elevator Parable

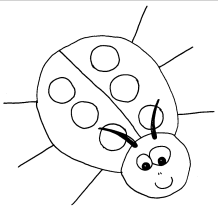
- News from The Boss
- *Listen to The Boss*

- "CNC customer timetable has changed"
- "We can wait for product delivery"



The Elevator Parable

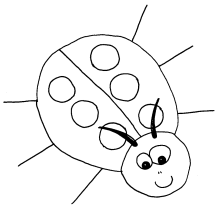
Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	



The Elevator Parable

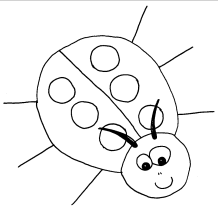
- News from The V.P. Finance
- *Listen to The Shareholders*

- "Capitalize CNC for *January*"
- "*Value* of work in capital when finished"
- "Policy"



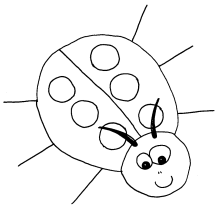
The Elevator Parable

Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	



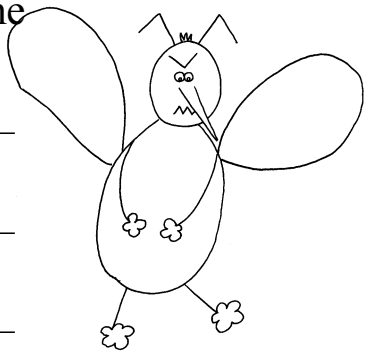
Bug Severity

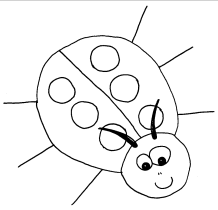
- How much damage it causes
  - severe
  - not severe



The Elevator Parable

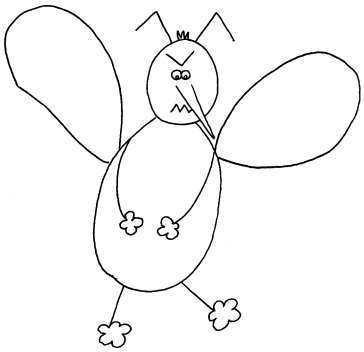
- Define Severity Scheme
  - S1
    - _____
  - S2
    - _____
  - S3
    - _____

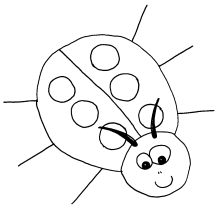




The Elevator Parable

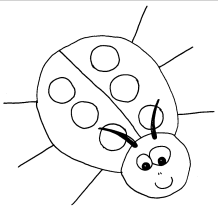
- Severity Scheme
  - S1
    - Unusable no straight forward work around
  - S2
    - Work around possible
  - S3
    - Cosmetic





The Elevator Parable

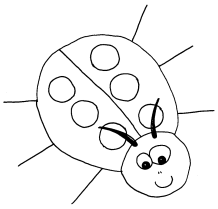
Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	



The Elevator Parable

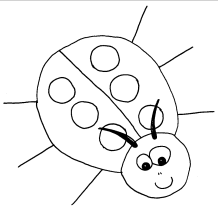
- News from User Education
- News from Product Management
- *Listen to The Users*

- "End Users cannot tolerate GPFs"
- "End Users are Doctors"



The Elevator Parable

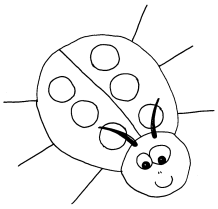
Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	



The Elevator Parable

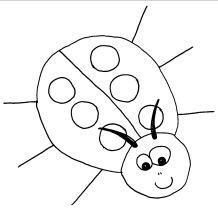
- News from The Development Lead
- *Listen to The Folks who write the code*

- "Profiler is for expert sys admin"
- "Profiler is a prototype"
- "Profiler will not be used by docs"
- "Profiler is an editor for INI files"



The Elevator Parable

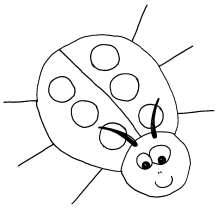
Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	



The Elevator Parable

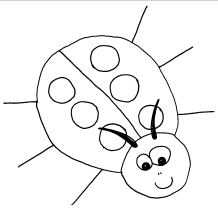
- News from The Developer
- *The Guru*

- "Profiler prototype was demoed"
- "To real sys admin folks"
- "They really loved it"
- "Crashed during demo"
- "Work in process"
- "Things are super!"



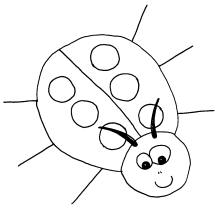
The Elevator Parable

Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	

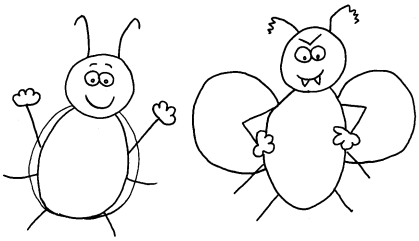


The Elevator Parable

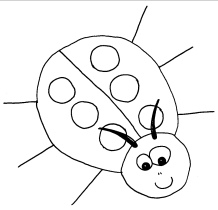
Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	



The Elevator Parable Moral

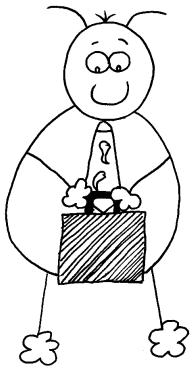


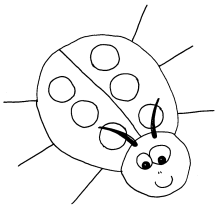
Bugs are not Good or Bad



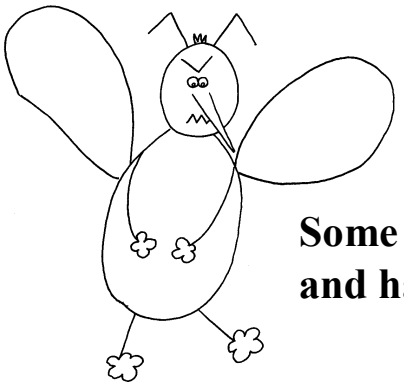
The Elevator Parable Moral

Some bugs are important and have a high priority!

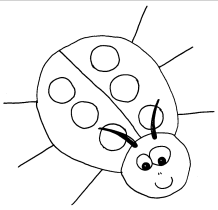




The Elevator Parable Moral

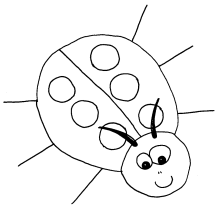


Some bugs are dangerous and have a high severity!



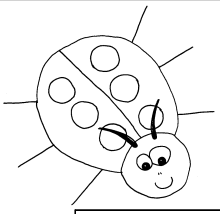
The Elevator Parable Moral

- Setting the priority and severity of a bug is a business decision
- Changing business conditions impact the priority and severity of a bug!
  - Always review previous decisions in light of changing business context
  - ensure staff assigning priority and severity are aware of all relevant business drivers



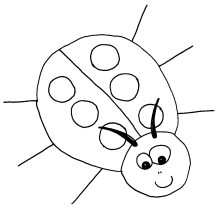
The Elevator Parable Moral

- And remember … don't loose any sleep over rumors you overhear in elevators!



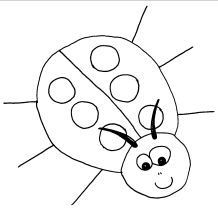
Bug Quadrants

Urgent Severe	Urgent Not Severe
Not Urgent Severe	Not Urgent Not Severe



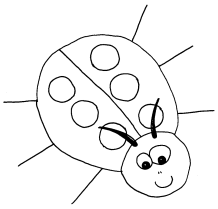
Business Decisions

- SQA:
  - objective input
- Development:
  - technical implementation
- Product Management:
  - customer driven requirements



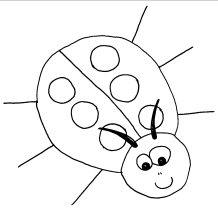
Quadrant Changing

- Same technical bug can be in a different quadrant depending on the business context
- Monitor business drivers!
- Focus find and fix quadrant -1- bugs high priority/high severity



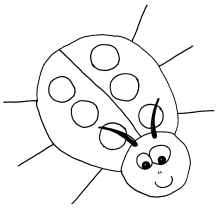
Priority and Severity Examples

- Note
  - some of the examples may apply to only one or few development teams within larger organization
  - often schemes are project based and are not used consistently company wide
  - some schemes presented may be out of use at the time of publication



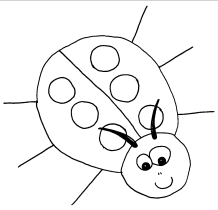
Priority and Severity Examples

- Purkinje, Inc.
- Priority
  - P1 Fix in current release
  - P2 Fix in current release if possible
  - P3 Fix in subsequent release
  - P4 Do not fix
  - P5 Feature request



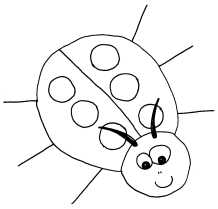
Priority and Severity Examples

- Purkinje, Inc.
- Severity
  - S1 System Crasher - Data Destroyer
  - S2 Major Problem
  - S3 Minor Problem
  - S4 Trivial



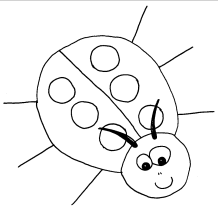
Priority and Severity Examples

- Software System Testing and Quality Assurance, Boris Beizer
- Severity
  - 10 levels
  - MILD , MODERATE, ANNOYING, DISTURBING, SERIOUS, VERY-SERIOUS, EXTREME, INTOLERABLE, CATASTROPHIC, INFECTIOUS



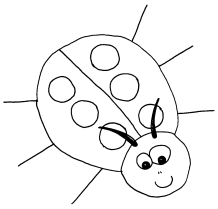
Priority and Severity Examples

- Measures for Excellence, Putman & Mayers
- Severity:
  - CRITICAL prevents further execution
  - SERIOUS subsequent answers grossly wrong
  - MODERATE behavior partially correct
  - COSMETIC tolerable



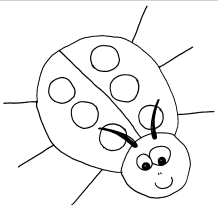
Priority and Severity Examples

- Practical Software Metrics for Project Management and Process Improvement, Robert Grady
- Severity:
  - CRITICAL
  - SERIOUS
  - MEDIUM
  - LOW



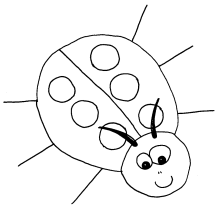
Priority and Severity Examples

- Testing Computer Software, Kaner, Falk and Nguyen
- Priority example:
  - 1 Fix immediately
  - 2 Fix as soon as possible
  - 3 Must fix before next milestone
  - 4 Must fix before final
  - 5 Fix if possible
  - 6 Optional - use your own judgement



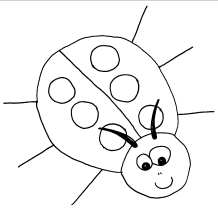
Priority and Severity Examples

- Testing Computer Software, Kaner, Falk and Nguyen
- Severity example:
  - 1 Minor
  - 2 Serious
  - 3 Fatal



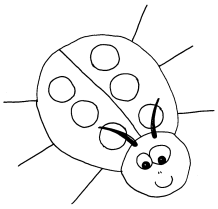
Priority and Severity Examples

- Microsoft Secrets, Cusumano, Selby
- Typical Severity Scheme:
  - S1 bug causes product to halt (“crash”) or be inoperable
  - S2 bug causes a feature to be inoperable and an alternative (“work-around”) solution is not possible
  - S3 bug causes a feature to be inoperable and a work-around solution is possible
  - S4 bug is cosmetic or minor



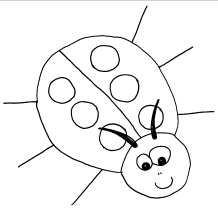
Priority and Severity Examples

- Net BSD
  - Severity
    - critical
    - serious
    - non-critical
  - Priority
    - high
    - medium
    - low



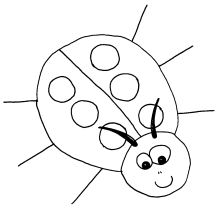
Priority and Severity Examples

- Managing the Software Process, Humphrey
- Quote (page 312)
  "… Since software defects are a major concern in both testing and operation, it is natural to use them as one key process measurement. Software defects (and the bugs that identify them) can be categorized as follows:
- - severity Measures the actual or anticipated impact of a defect on the user's operational environment. Typically such measures are valuable in establishing service priorities. …"



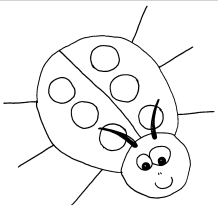
Priority and Severity Examples

- Software Project Survival Guide, McConnell
  - Defect Count Example
    - critical defects
    - serious defects
    - cosmetic defects
    - etc



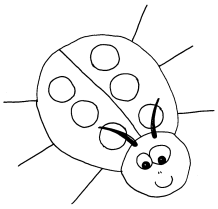
Priority and Severity Examples

- Dynamics of Software Development, McCarthy
  - Triage ruthlessly
    - "The severity of the bug. A pertinent question to ask, especially in the end-game, is whether you would recall the product if this bug were discovered after the product is shipped. Is it a showstopper?"

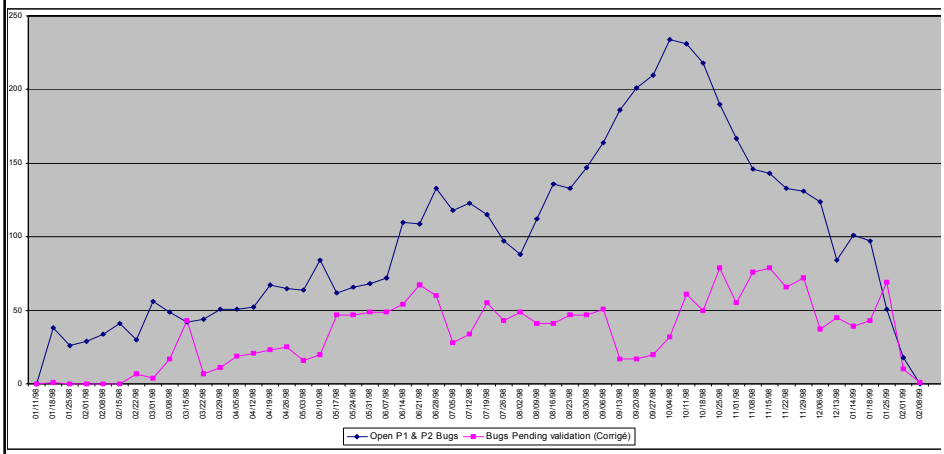


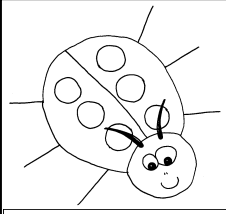
Priority and Severity Examples

- Software Inspection, Gilb, Graham
  - Definition of Severity
    - "The classification of an issue based on the estimated future cost to find and fix a defect at a later stage, if not fixed at this stage. The alternatives are *minor* (about the same), *major* (substantially greater), *critical* (product or project threatening later)."

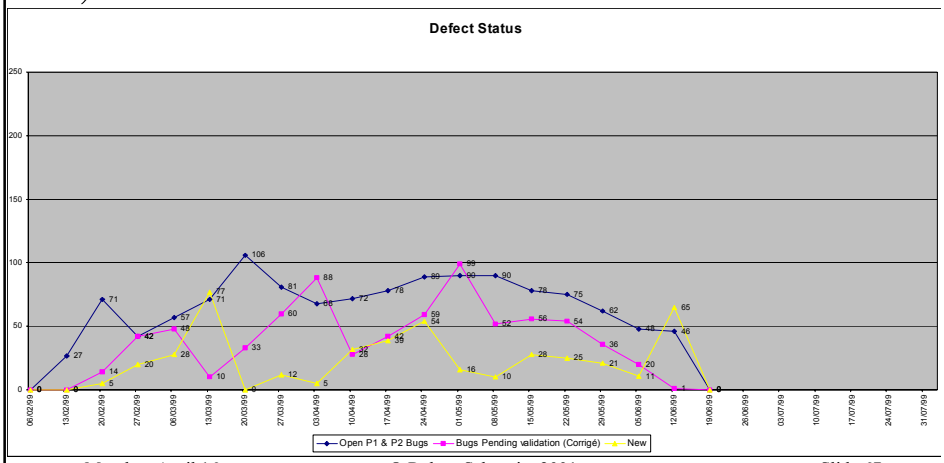


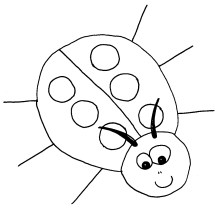
CNC 2.10 Final



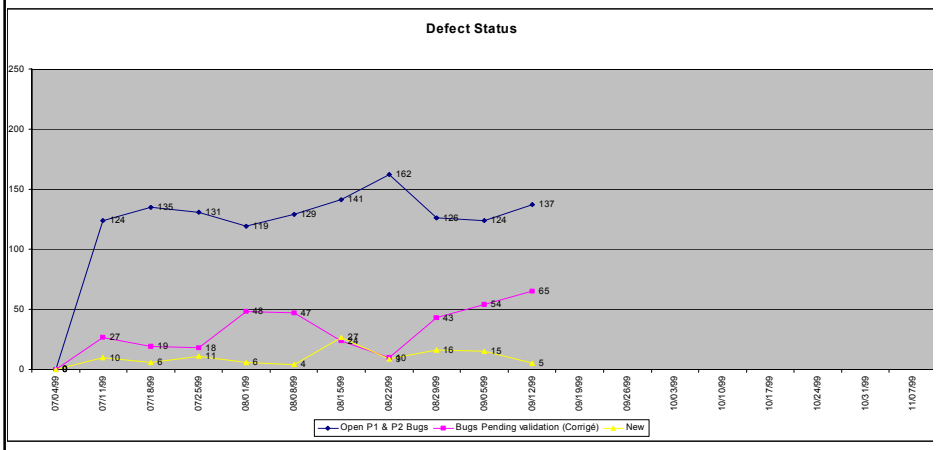


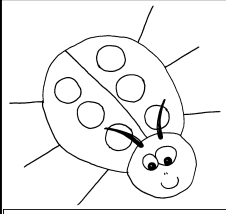
CNC 2.20 Final



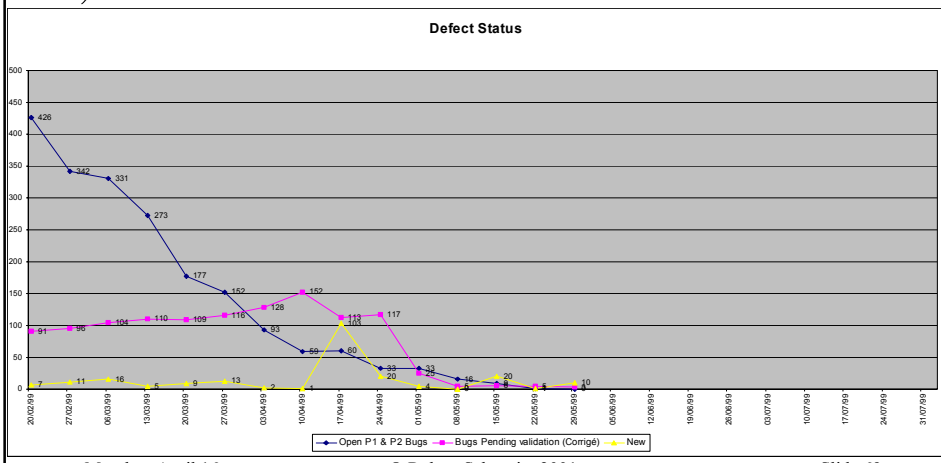


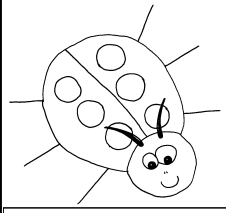
CNC 2.3 In Progress



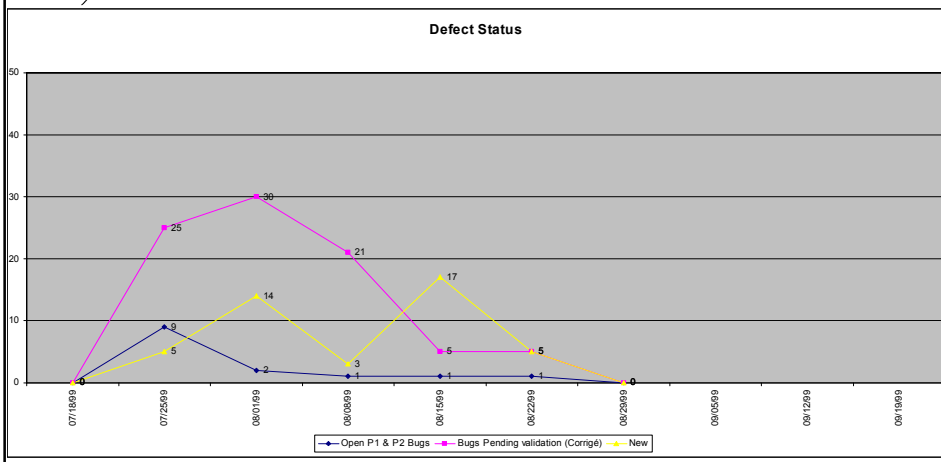


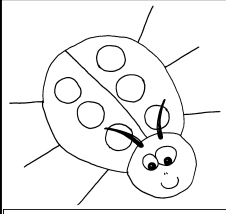
CCM 1.36 Final



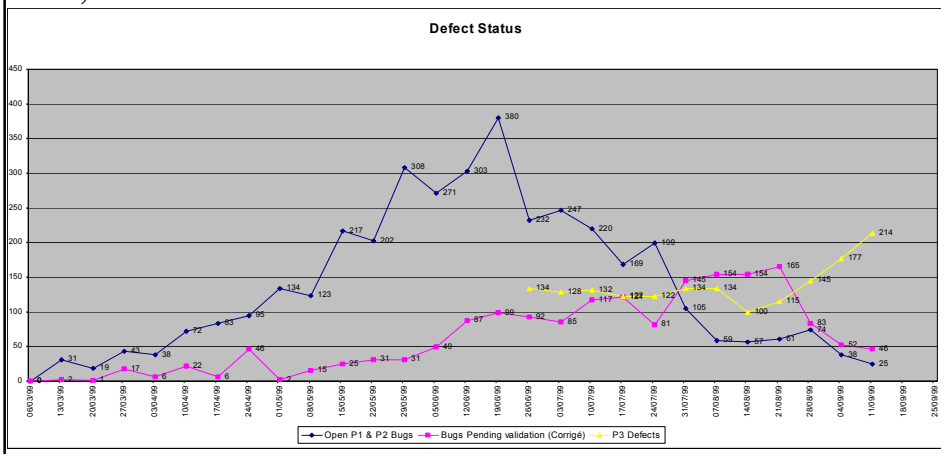


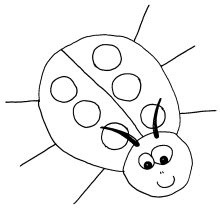
CCM 1.37 Final



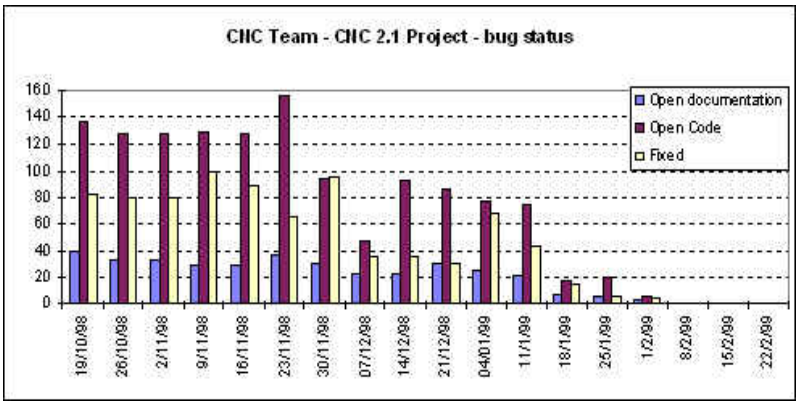


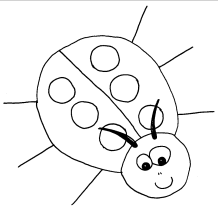
DCI 2.0 Tail End



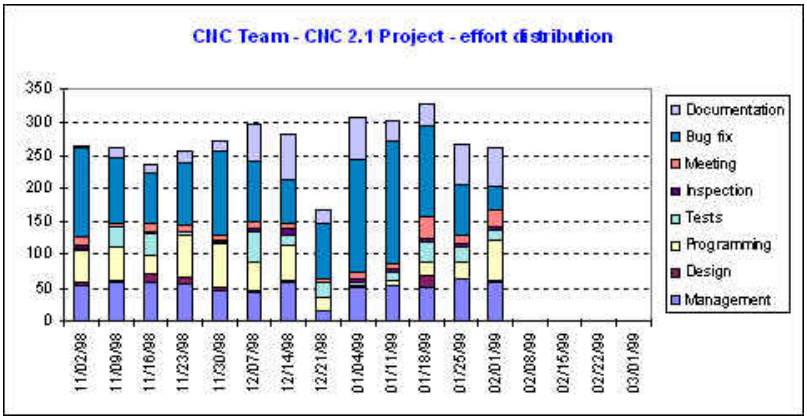


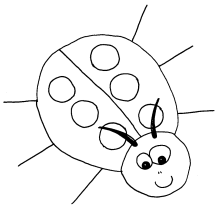
CNC 2.1 Project Data



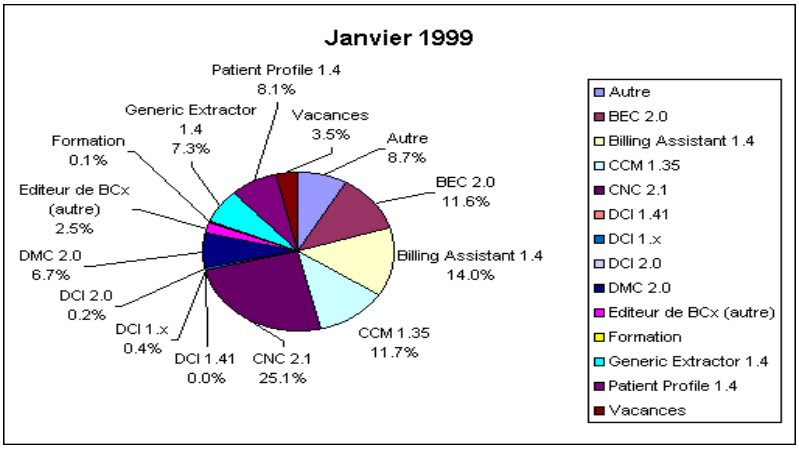


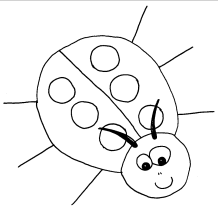
CNC 2.1 Project Data



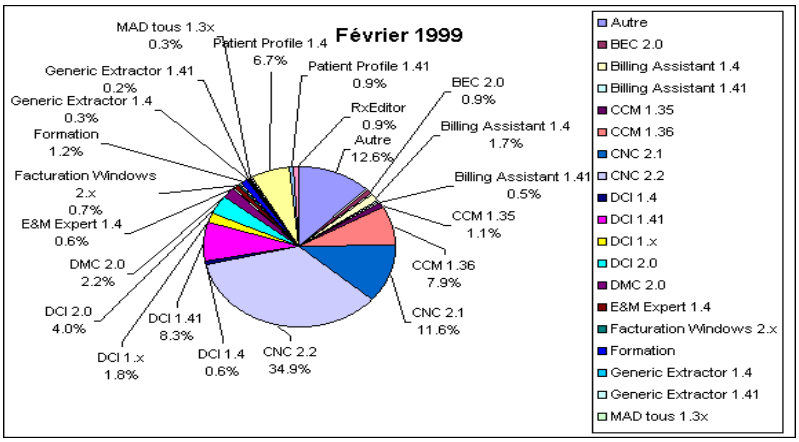


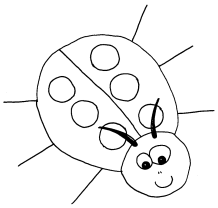
SQA Effort Distributions





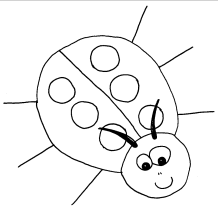
SQA Effort Distributions





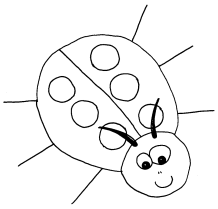
Practical Aspects

- Tracking
  - Publicize data in graphical or tabular form as much as possible, practical and politically acceptable
  - doors, coffee machines, near the laser printer, near the photocopier
  - update status regularly (at least in sync with builds to SQA)



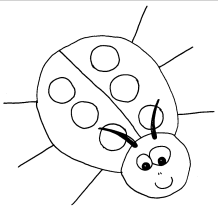
Practical Aspects

- Tracking
  - Ensure there is a way for the project test lead to confirm a bug and validate the description as being clear and complete before allowing others to see it (keep it private until it has been reviewed by the lead or a reasonably senior peer)



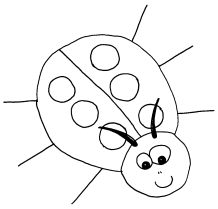
Practical Aspects

- Tracking
  - make bug list available to anyone who may have an interest in the project (READ ONLY)
  - provide training seminars about how to read a bug list without any panic
    - if it is in the list then we know about it and therefore we can make a rational decision about what to do about it



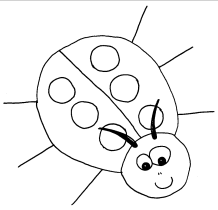
Practical Aspects

- Tracking
  - recommend email notifications in form of executive summary periodically or whenever a major change takes place
  - if someone outside your development organization reports a bug it is good politics to let them know the bug number assigned to it so they can track it by polling the bug list!



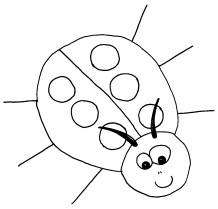
Practical Aspects

- Logging
  - best practice is to have a company or project standard form to complete
  - many examples in industry and samples provided in all commercial bug tracking software
  - train staff in bug logging
  - keep personal testing notes for later reference and reminders



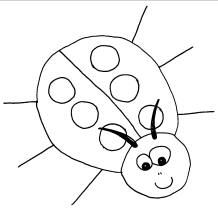
Practical Aspects

- Logging
  - how to repeat bug
  - classification of bug
  - element of test plan
  - version, configuration, build number
  - attachments, screen shots, files
  - severity (priority is decided later at bug review meeting)



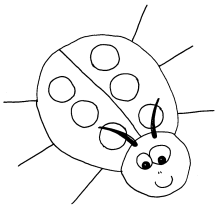
Practical Aspects

- Logging
  - keep in mind when logging the bug that you may be called on short notice into a very intense meeting in progress to explain or demonstrate the problem
  - be prepared
  - review entry with at least one person preferably a test lead or senior peer



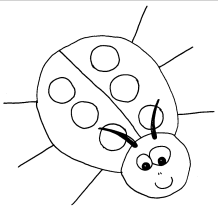
Practical Aspects

- Bug Review Meeting
  - bug review meetings are among the most important activities in a software development project
  - decisions are made as to what the priorities are for all bugs in the system
  - when business conditions have changed a review of all lower priority bugs is needed to reclassify as required



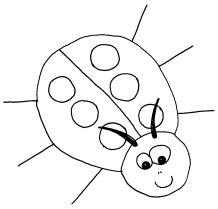
Practical Aspects

- Bug Review Meeting
  - all attendees should have access to the bug list before the meeting and have sufficient lead time to know what priorities they would assign to the bugs
  - assume approximately 2 hours are required per person reviewing about a weeks worth of NEW UNCLASSIFIED BUGS



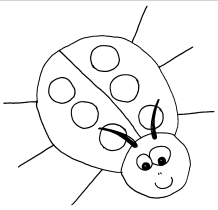
Practical Aspects

- Bug Review Meeting
  - if more than 2 hours review time is needed then increase the frequency of your bug review meetings
  - as few people as possible should be attending the bug review meeting
    - development lead
    - product manager
    - sqa lead



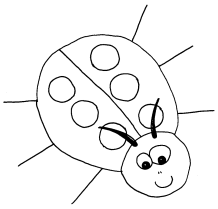
Practical Aspects

- Bug Review Meeting
  - other staff should be available on demand to help clarify technical or business issues related to the bug
  - run bug review meeting objectively
    - avoid finger pointing
    - avoid assigning blame
    - be objective and unemotional



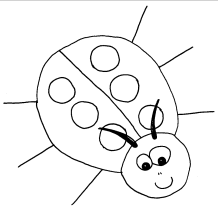
Practical Aspects

- Bug Review Meeting
  - agree on an order to review the bugs
    - for low volume the easiest is sequentially
    - logical order could be by function and then from most dangerous to least dangerous severity
    - order should be rational
  - moderator should be diplomatic
  - training in how to run a meeting



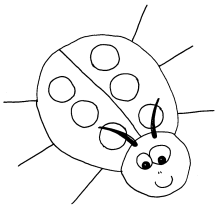
Practical Aspects

- Bug Review Meeting
  - can be run similar to a defect logging meeting of a formal inspection (Gilb style) but with more discussion encouraged to come to a business decision especially on gray subjective areas about what product users would could or should do!



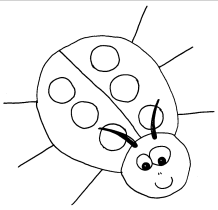
Practical Aspects

- Bug Review Meeting
  - all stakeholders should be notified of decisions made in Bug Review Meetings
    - developers
    - testers
    - technical writers
  - ensure a process exists to notify staff that they have work to do related to fixing a problem.



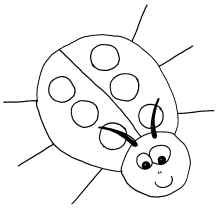
Practical Aspects

- Bug Review Meeting
  - although email and electronic notification is very popular I recommend communicating in person (or by phone or by some form of electronic conferencing)



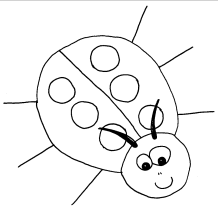
Bug Tracking Systems

- Never loose a bug
  - a bug tracking system is a database in which all bugs discovered or reported about an application are collected
  - typically a *"test lead"* is responsible for managing the *"bug list"* for an application

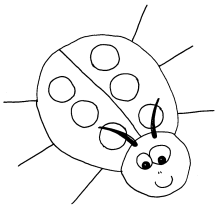


Bug Tracking Systems

- Never loose a bug
  - a bug tracking system is a database in which all bugs discovered or reported about an application are collected
  - typically a *"test lead"* is responsible for managing the *"bug list"* for an application

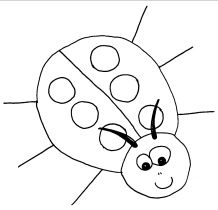


Bug Reports



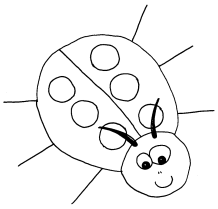
Bug Reporting

- The "bug" flow is something like this
  - bug is discovered in testing or reported from the field
  - a bug report form is completed
  - the bug report form is reviewed
  - the bug report is added to the bug list
  - a decision is made, at a bug review meeting, about whether the bug should be fixed
  - if the bug is fixed then the software is re-tested to reconfirm that the bug has indeed been fixed
  - if the bug is not fixed (on purpose!) then a description of the work around is published or made available to help desk staff



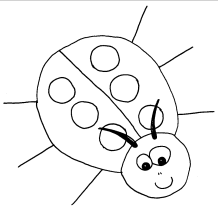
Bug Reporting

- Potential Audience of Bug Report
  - other testers
  - test leads
  - developers
  - development leads
  - product managers
  - customer support team members
  - technical writers



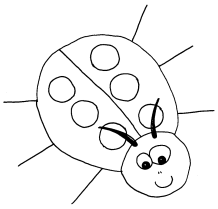
Bug Reporting

- Most important reason to report the bug
  - provide input to decision makers regarding status of product
  - decision to ship is based on status of open bugs
  - a ship decision is among the most important in the entire software development process
  - if a decision is made to fix the bug the description had better help the developer get the job done!
  - it is critical to have high quality bug report information!



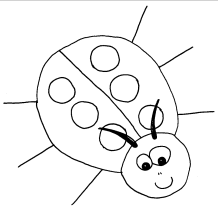
Bug Reporting

- Effective bug reports
  - explain how to reproduce the problem
  - describe the problem in a reasonable number of steps
  - minimize the amount of additional questions raised on reading it!



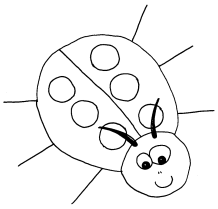
Bug Reporting

- Effective bug reports
  - description should be:
    - complete
    - clear
    - objective
    - not confrontational



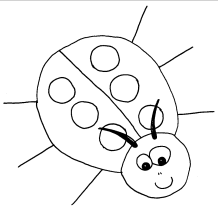
Some Typical Bug Report Fields

- Bug Number
  - a unique number assigned to the bug
  - bug numbers should never be reused
  - it is a good idea to have unique numbers across all products in an organization to avoid any future possible accidental confusion
  - usually generated automatically



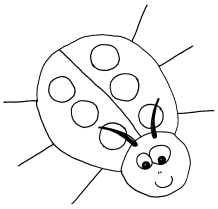
Some Typical Bug Report Fields

- Application
  - Name of the application the bug is about
  - especially useful if you have a series of applications or applets!



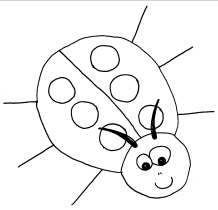
Some Typical Bug Report Fields

- Version and Build Number
  - which build of which product
  - which version
  - sometimes this value is found in help about box
  - if no build number is available use a date time stamp of build or equivalent



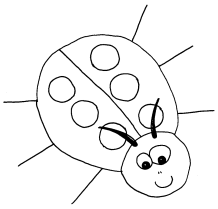
Some Typical Bug Report Fields

- Problem Type
  - describes the Type of problem found
  - hardware, software, documentation, help
  - depends on application environment
  - may include suggested enhancement or questions if not really a problem but a concern raised during testing!



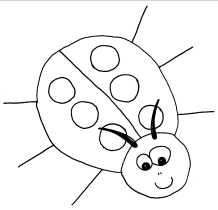
Some Typical Bug Report Fields

- Proposed Severity
  - how serious is the problem
  - this is usually entered based on the policy of the project and could change based on business context
  - Usually numeric scheme S1 … Sn where S1 is most severe



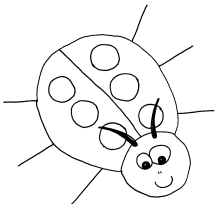
Some Typical Bug Report Fields

- Attachments
  - list of additional attachments
  - files
  - screen captures
  - database
  - additional information to help facilitate decision making regarding keep or fix



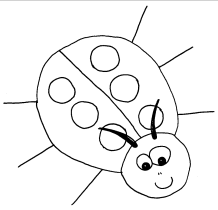
Some Typical Bug Report Fields

- Problem Summary
  - short clear description of the problem
  - usually used in executive summary of bug status
  - *"program crashes when saving using an invalid file name"* for example



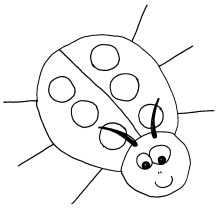
Some Typical Bug Report Fields

- Can Problem be reproduced
  - Yes, No or Sometimes
  - especially useful for the case of field reported problems
  - generally testing team should have a "yes" here!



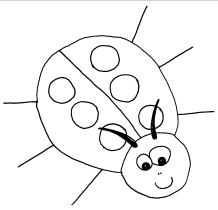
Some Typical Bug Report Fields

- Problem Description and Steps to Reproduce It
  - detail description of the problem
  - clear step by step description of how to repeat it
  - how to get to appropriate system state to reproduce the problem



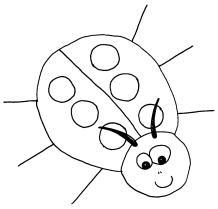
Some Typical Bug Report Fields

- Suggested Fix
  - sometimes you may be able to propose a fix!
  - It may be ignored - but if it makes sense and you are qualified do not hesitate!



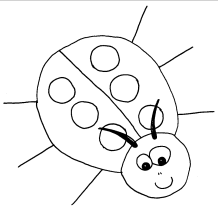
Some Typical Bug Report Fields

- Reported By
  - your name
  - your department or other identification
- Dates
  - date found
  - date reported



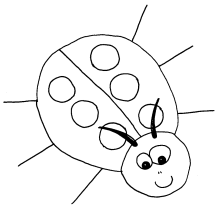
Some Typical Bug Report Fields

- Platform
  - operating system
  - client, server descriptions
  - versions of environment software
  - browser
  - windows version



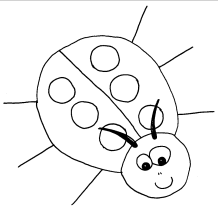
Some Typical Bug Report Fields

- Functional Area
  - part of application
  - useful for extracting data, all bug reports related to this Functional Area



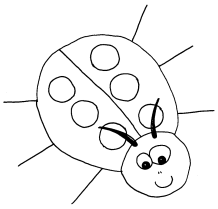
Some Typical Bug Report Fields

- Comments or Notes
  - anyone working on the bug or reviewing it can add comments
  - helps keep up with added information without revising descriptions as bug is worked on!



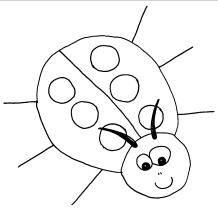
Some Typical Bug Report Fields

- Status
  - has the bug been reviewed
  - is it Open
  - is it Closed
  - is it Pending Review
  - … whatever works in your company



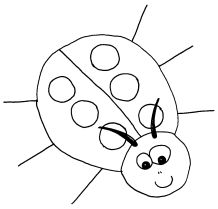
Some Typical Bug Report Fields

- Priority
  - how urgent is this bug
  - when (if ever) will it be fixed
  - result of bug review meeting



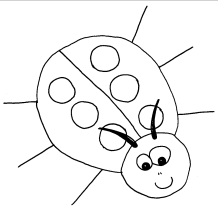
Bug Descriptions

- Simple
  - do not use complex grammar structures or ambiguous wordings to describe the bug
  - use short clear phases
  - point form lists are great



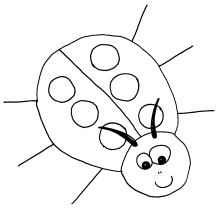
Bug Descriptions

- Rational
  - the bug description should make sense to all readers
  - if you find a bizarre set of keystrokes which reproduce the problem in a consistent way "great" … but please indicate that that is only one of several ways ...



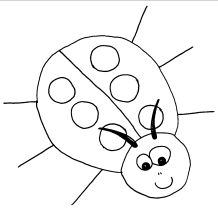
Bug Descriptions

- Unemotional
  - do not get too passionate in the description
  - be clear and business like in tone



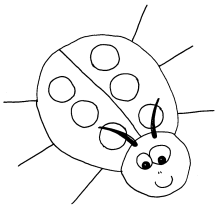
Bug Descriptions

- Objective
  - no finger pointing
  - your job is to give objective input to help people make business decisions



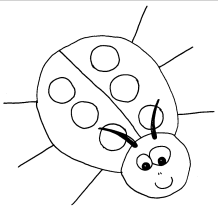
Bug Descriptions

- Review
  - have a peer or lead review every description to ensure it is clear and objective
  - be prepared to defend the description



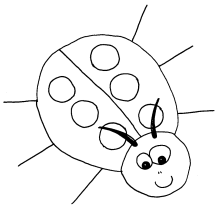
Bug Descriptions

- Consequences
  - what are the possible consequences of this bug to the system user?
  - Are there more important consequences?



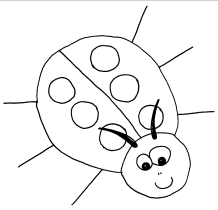
Bug Advocacy

- Work by Cem Kaner
  - " … a bug report is a tool that you use to sell the programmer on the idea of spending her time and energy to fix a bug …"



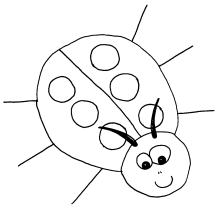
Bug Advocacy

- Bug report should
  - sell the need to fix the bug
  - motivate the bug fixer
  - motivate the business decision
  - overcome objections



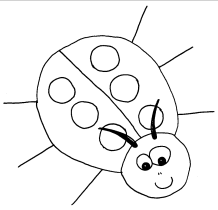
Bug Advocacy

- "The best tester is the one who gets the most bugs fixed!"
  - Cem Kaner
    - Bug Advocacy Workshop
    - Software Quality Week - May 2000



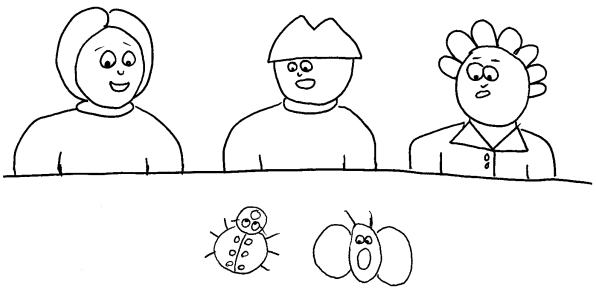
Example Bug Flow

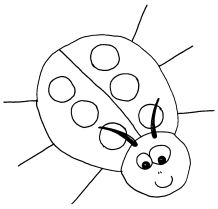
Never Loose a Bug!



Finished?

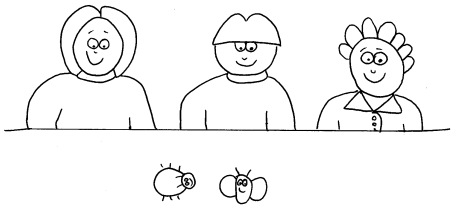
- How do you know you are finished?

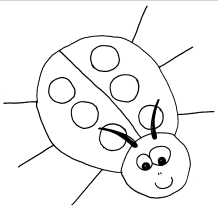




You know you are finished when …

- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) ...

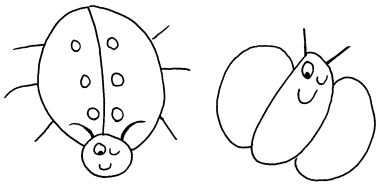




You know you are finished when …

- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) …

At least for now!



QW2001 Workshop W4

Ms. Johanna Rothman & Ms. Elizabeth Hendrickson
(The Rothman Consulting Group)

Grace Under Pressure: Handling Sticky Situations in Testing

Key Points

- How to make sure they heard what you said and not what they wanted you to say.
- How to handle difficult situations and obstinate people.
- What to do when you don't feel like anyone is listening.
- How to say "no" and make it stick.

Presentation Abstract

In this workshop Elisabeth Hendrickson and Johanna Rothman examine a series of difficult interactions between testers, test leads, developers, and managers, demonstrating proven techniques for presenting bad news, saying "no," and influencing others' behavior when you have no authority over them.

About the Author

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people.

Johanna publishes "Reflections", an acclaimed quarterly newsletter about managing product development. Johanna's handbook, "Hiring Technical People: A Guide to Hiring the Right People for the Job," has proved a boon to perplexed managers, as have her articles in Software Development, Cutter IT, IEEE Computer, Software Testing and Quality Engineering, and IEEE Software.

Johanna is the founder and principal of Rothman Consulting Group, Inc., and is a member of the clinical faculty of The Gordon Institute at Tufts University, a practical management degree program for engineers.

Elisabeth Hendrickson is the Director of Quality Engineering at Aveo Inc., an

Application Service Provider. Aveo Inc. offers Attune, a pre-emptive technical support service whose mission is to help companies communicate the right information to the right customer at the right time. Prior to joining Aveo, Elisabeth was the founder of Quality Tree Consulting, a software quality assurance consulting firm. As a consultant, Elisabeth provided services to Application Service Providers as well as more traditional independent software vendors.

Grace Under Pressure: Handling Sticky Situations in Testing

Elisabeth Hendrickson
Quality Tree Software, Inc.
925-426-9726
esh@qualitytree.com
www.qualitytree.com

Johanna Rothman
Rothman Consulting Group, Inc.
781-641-4046
jr@jrothman.com
www.jrothman.com

What's the Problem?

- Testing is full of pressure situations
- We're people, so we constantly find sticky situations

If you can keep your head while others are losing theirs…. -- Rudyard Kipling

- Easier said than done

 www.qualitytree.com www.jrothman.com 2

People Under a Little Pressure May

- Focus
  - Ignore anything unrelated to the current problem
  - Become “intense”
- Feel challenged and exhilarated
- Work hard
- Push others

 www.qualitytree.com www.jrothman.com 3

People Under a LOT of Pressure May

- Play CYA games
- Play politics
- Yell
- Blame
- Yield
- Snap
- Ignore requests
- Curl up under their desks
- Demonstrate bizarre, uncharacteristic behavior

 www.qualitytree.com www.jrothman.com 4

When Sticky Situations Explode

- No one gets anything productive done: they're too busy dealing with the situation to deal with the work.
- The result can damage more than one project: it can permanently damage the team and ultimately the company.
- As a manager, your job is to handle the situation to make the outcome as good as possible and minimize the damage—in a way that works for you.
- We're going to discuss a specific problem-solving technique

Situation 1: Ready to Ship?

You're the test manager on a project that has been having serious problems since it was first delivered. Your group has found several serious bugs and documented them. So far, the bug review committee has decided to defer all of them. One of your testers just found a really awful bug that results in data corruption. The executives still expect the software to ship Monday. Now what?

These Responses are NOT Helpful

- Placate the decision makers:
  - You say to yourself: "They're not going to fix anything anyway. So we'll ship bad software. Oh well."
  - Note: Do this enough and you won't have enough Tums.
- Blame others:
  - To your manager: "HOW CAN YOU POSSIBLY CONSIDER SHIPPING THIS PIECE OF !@#$%!!!?!?!?!?!?"
  - To the developers: "If you weren't such bad programmers, this wouldn't be a problem!"
  - To the executives: "YOU'RE JUST A BUNCH OF QUALITY NEANDERTHALS!"
  - Note: Blaming is best done with a pointed finger and a loud voice

 www.qualitytree.com www.jrothman.com 7

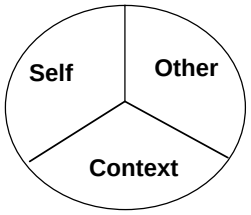
Activity

- Get into groups of three
  - Each person take a role: test manager, executive, observer
  - Try the ready-to-ship scenario
  - Try making the interaction as painful, nasty, vicious as possible
- Debrief

 www.qualitytree.com www.jrothman.com 8

Recognizing Congruence and Incongruence

- Self: We consider our own needs and capabilities
- Other: We consider needs and capabilities of other people
- Context: The reality of the situation



 www.qualitytree.com www.jrothman.com 9

What Every Child Knows

- School teaches us what to do if we are on fire:
  - Stop
  - Drop
  - Roll
- Apply this to work. When the situation is on fire:
  - Center: Stop and think. Assess your own state of mind and reactions
  - Enter: Enter into the other's context
  - Turn: Turn the situation back into a healthy situation (Roll out the flames) and solve the real problems causing the fires

 www.qualitytree.com www.jrothman.com 10

Center

- Close your door (if you have one)
- Breathe
- Mentally drop out while the conversation continues around you
- Take a walk
- Get coffee/water/soda
- Excuse yourself to go to the restroom
- Find an empty conference room or office and hide
- Say what you want to say to an empty office, car, room.
- If all else fails, pretend to pass out and collect your thoughts while they call 911

 www.qualitytree.com www.jrothman.com 11

Center, Part 2: Assess Yourself

- What are you feeling right now? Fear? Anger? Resentment? Secret satisfaction and the desire to say "I told you so?"
- Are you thinking in terms of fault, culprits, accountability?
- Are you trying to figure out how you'll keep executives from pointing fingers at your group?
- Are you feeling overwhelmed by it all and want to ignore it until it goes away?
- Are you exhibiting any physical signs of stress (nausea, tensed muscles, clenched teeth)?
- You're human, expect human reactions. Learn from them.

 www.qualitytree.com www.jrothman.com 12

Your Physical and Mental Responses Are Helpful Clues

- Recognize how your body reflects your state of mind.
- Notice any discrepancies between your physical and mental states.
- Whether your responses are mean spirited or well intentioned, they are your responses. Honor them.

 www.qualitytree.com www.jrothman.com 13

Enter and Reframe Your Thoughts and Feelings

- Everyone involved is doing what they believe to be best
- Everyone, including us, has a right to an opinion
- We can express what we want to say in a way that makes it more likely to be heard
- We are not powerless nor are we all-powerful
- We are not totally ignorant nor are we omniscient
- We, along with everyone else on the project, are human and we all deserve to be treated with dignity and respect

 www.qualitytree.com www.jrothman.com 14

Relax

- Recognize tension as “fight or flight” instinct
- Breathe deep
- Clear your mind
- Count to 10, 50, 100, or 1000 as needed
- Find all the tensed muscles and methodically relax each one

 www.qualitytree.com www.jrothman.com 15

Enter, continued: Assess the Situation

- Is it as bad as it seems?
- What’s the best possible outcome?
- What’s the worst possible outcome?
- What options are available?
- Who is really responsible for the decisions in this case?
- What information do I need?
- What information do I have that others need?

 www.qualitytree.com www.jrothman.com 16

Turn Possibilities

- Speak your mind, honestly, but calmly and without accusation, blame, or capitulation
- Acknowledge if you don't have any answers
- Speak for yourself; let others speak for themselves
- Own your opinions: "I think…", "I believe…"
- Watch your tone of voice and body language
- Watch the other person to see how they're responding to you

 www.qualitytree.com www.jrothman.com 17

Situation 2: You're a Target

Your manager has hauled you into her office and is berating you for allowing bad software to ship. She's not letting you get a word in edgewise; she just seems to want you there as a human target. What to do?

 www.qualitytree.com www.jrothman.com 18

Center Yourself

- Excusing yourself may be awkward.
- Try to stop the conversation:
  - Hold up your hand
  - Stand up if sitting; sit down if standing
  - Interrupt
- If that fails, stop paying attention:
  - Ignore her; focus elsewhere
- If all else fails, leave the room. You do not need to stay there to be abused.

 www.qualitytree.com www.jrothman.com 19

Enter

- What's the real problem?
- What are your manager's real concerns?
- Is your manager taking everything into account: her, you, and the context?
- What's affecting your manager's behavior? Perhaps she's just been a target for her manager.
- If you were to do everything over again from the beginning, what might you do differently?
- If you had more control in the situation, what might you change?

 www.qualitytree.com www.jrothman.com 20

Turn

- Re-engage in the conversation with your boss in a way that allows you to speak as well.
- Acknowledge her concerns and tell her what you think you heard her say.
- If you have any ideas about how to address her concerns in the future—whether things you control or things she controls—present them now.
- Help her understand that treating you badly is not OK with you.
- If you cannot get her to treat you with respect, walk away.

 www.qualitytree.com www.jrothman.com 21

A Note on Bad Managers

- If your manager treats you as a target on a regular basis, you have an additional problem: her behavior. How you react to her behavior is completely up to you.
  - You can choose to address your concerns with upper management or HR
  - You can cope with the nastiness
  - You can leave.
- You own your destiny.
  Your manager does not.

 www.qualitytree.com www.jrothman.com 22

Activity

- Get into groups of three
  - Each person take a role: manager, target, observer
  - Try the you're a target scenario
- Special instructions:
  - **If you are the manager**, remember that your goal is to take out all your frustrations, anger, etc. on the poor hapless target. This is your opportunity to pretend to be a total jerk. Resist your natural tendency to be nice, to respond positively, to mellow out.
  - **If you are the target**, your goal is to center, enter, and turn the manager in a more positive direction. The manager will resist you. Don't give up too easily. And in the end, remember that the other person playing the manager is just playing a part.
  - **If you are the observer**, watch what happens. Do not participate, but be ready to debrief the manager and target.
  - **Switch roles**.
- Debrief

 www.qualitytree.com www.jrothman.com 23

Situation 3: No Information

You are supposed to meet with the project team to decide what to do about the release, but you can't get any information out of the tester working on the project. The tester reports to you. He's been filing bug reports, but not as many as you expected to see. So far, he's managed to avoid giving you any results from the planned test cases. You're beginning to worry that he's not actually doing what he was supposed to do. You've called him into your office to understand where he is in his testing. How do you get the information you need?

 www.qualitytree.com www.jrothman.com 24

Center, Enter, Turn

- Before you begin speaking, stop and assess your state of mind.
- Pose the problems to him as they affect you:
  - I don't think I have enough information about the state of the software to make a good decision.
  - I am concemed about the lack of documented test results.
- Give him a chance to center before responding.
- Find a way to get what you need. Consider all your options.

 www.qualitytree.com www.jrothman.com 25

Activity

- Get into groups of three
  - Each person take a role: test manager, tester, observer
  - Try the no information scenario
- Debrief

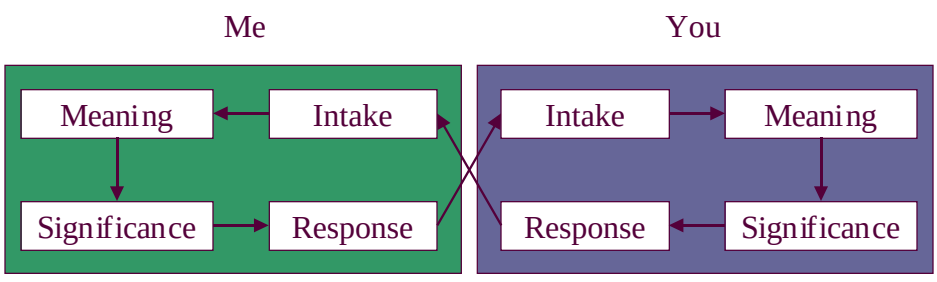
 www.qualitytree.com www.jrothman.com 26

Situation 4: Resisting Change

You are the Quality Engineering manager. The Development manager is your peer. You've noticed some patterns in the sources of bugs and have been trying to introduce processes to prevent these common bugs. The Development manager is resisting, "No, we don't need to change anything in development. I need you to have a positive attitude toward my development group. And you also need to find the bugs faster."

 www.qualitytree.com www.jrothman.com 27

Anatomy of an Interaction



 www.qualitytree.com www.jrothman.com 28

What You Say v. What They Hear

- The other person may interpret what you said as:
  - What they wanted to hear
  - What they expected to hear
  - Only the parts that don't get filtered out ("selective hearing")
  - What they last heard in a similar situation
  - What they're most afraid you'll say
- Be aware of mismatches between what you thought you said and what they appear to be reacting to.
- You might have the same inaccurate interpretations of what the other person is saying.

 www.qualitytree.com www.jrothman.com 29

Activity

- Get into groups of three
  - Each person take a role: quality engineering manager, development manager, observer
  - Try the resisting change scenario
- Debrief

 www.qualitytree.com www.jrothman.com 30

Other Situations

- Making a ship/no ship decision with unclear release criteria
- Getting real, useful feedback on test plans when no one has time to give it to you ("Whatever you think is probably fine. Just go with it.")
- Dealing with an irate tester whose bugs have all been deferred
- Others?

Activity

- Get into groups of three
  - Choose a situation you're having trouble with that you want to practice
  - Try one with your group
- Debrief

References

- Congruent Leadership Change Shop Readings (see http://www.geraldmweinberg.com)
  - Weinberg, Gerald, "Congruence in Software Management"
  - Weinberg, Gerald, McLendon, Jean, Weinberg, Daniela, "Notes on Congruence"
- Fisher, Roger, et al, *Getting to Yes: Negotiating Agreement Without Giving in*, Penguin USA, 1991

QW2001 Standby Paper SB Technology

Mr. Michael K. Jones, Assistant Professor
(Dromedary Peak Consulting/Western International University)

Test Strategy Derivation

Key Points

- Test strategy derivation is a necessity to efficiently and effectively test.
- The sources and foundations of a test strategy include more than requirements.
- Critical dependencies and their leverage points must be focused on.

Presentation Abstract

Test strategy derivation must always be performed but literature on the subject is minimal at best, other than a traditional discussion of the merits of top-down versus bottom- up with regards to integration. In the increasing complex systems found in today's businesses, both off- line and on the web, an efficient and effective test strategy must be a goal that is reached for a system to be comprehensively and accurately tested. Simply "banging on the box" or "flailing at the machine" will not do the job.

This paper will define what a test strategy is and what it must cover. It will review the necessary sources and foundations for a test strategy, i.e. system requirement specifications and system architecture documents, and discuss how the test strategy can possibly be obtained in tandem with them. The paper will cover the critical dependencies of a test strategy and explore the leverage points that may be found among them. In particular the integration strategy and build documentation will be analyzed for their relationship to a successful test strategy. Test documentation will be then be reviewed to show where and how this information is best noted and preserved.

About the Author

Mr. Jones has a Master of Science in Computer Information Science and a Master of Business Administration. He has been through software engineering training at Boeing, Texas Instruments, and McDonnell Douglas in the past. He has worked in the software industry since 1976 and is currently the Chief Consultant at Dromedary Peak Consulting, which provides analytical direction and operational support for business. He is also an Assistant Professor in Information Technology at Western International University. His courses include Advanced Software Engineering, Advanced C Programming, Information Resource Management, Internet Business Strategy, and Web Application Development. Some of his published articles include: "Pragmatic Software Configuration Management in the

E-World", "Pragmatic Software Testing in the E-World", "Software Configuration Management for the Web", "Report from Captain QA from the Web", and "Four Conceptual Attributes for Successful Web Applications".

Test Strategy Derivation

Michael K. Jones
Chief Consultant
Dromedary Peak Consulting
Mesa, Arizona USA
And
Assistant Professor
Western International University
Phoenix, Arizona USA

Test strategy derivation is necessary if computer systems are to be comprehensively and accurately tested.

What is a test strategy?

- Beizer (1995) – focus on bug removal
  Koomen and Pol (1999) – same
  Pol and Van Veenendaal – same
- Hetzel (1988) – test process techniques and infrastructure
- Perry (1995) – reflection of SDLC
- Dustin, Rashka, Paul (1999) – automation
- Beizer (1990) – white box testing option

All of these authors miss the total picture of what testing is supposed to accomplish and consequently what a test strategy should be.

A test strategy should:

1) Maximize defect identification through increased isolation

2) Validate requirement delivery with tangible proofs

Due to the additional tasks of effective process and cost efficiency, a test strategy should also:

3) Provide effectiveness in testing an unknown against a known by instrumenting incremental block testing.

4) Improve cost efficiency by decreasing test jigs to the minimum number.

A test strategy must not only cover the “what” of testing, but also the “how”.

Sources for the derivation of the test strategy:

Not Just

- Source code
- User’s Guide

But also

- Technical Requirements Document
- Service Level Agreements
- System Architecture Document
- Design Documents
- Program or Project Plan Document
- Integration Strategy Document
- System Build Document

Technical Requirements Document – Each and every requirement must be addressed individually. Not only correct responses to correct inputs must be measured, but correct responses to incorrect inputs must be dealt with.

Service Level Agreements (SLA's) – May impose additional requirements. Should always be checked due to contractual liability.

System Architecture Document – Reveals the true structure and modularity of system. Linkages and interdependencies will be clarified, and consequently leverage points, too

Design Documents – Describe the algorithms and mechanics of each component. May be embedded in code, but can still be utilized.

Program or Project Plan – May have an impact if viewing by clients is called out prior to release, or if incremental release of functionality is stated.

Integration Strategy Document – Should help the testing sequence. However, integration was affected by needs of developers. With that said, don't lose sight that this is evidence of successful integration.

Build Document – Is the recommended integration construction sequence. May be modified due to testing extingencies, but feedback should be given to build engineer.

Test strategy should be documented in the System Test Plan. Test strategy Information captured should include:

- Missions or goals
- Coverage
- Sources
- Sequence of testing
- resources

In conclusion, a well prepared test strategy will provide competent testing, and deliver a successful system to the users.

For feedback and/or further communication, please contact me at:

Michael K. Jones
Dromedary Peak Consulting
1451 East 8th St.
Mesa, AZ 85203 USA
Phone: 480-833-0927
Cell: 480-363-7527
Email: mjones@dromedarypeak.com
jonesaz@uswest.net

Test Strategy Derivation

By
Michael K. Jones
Chief Consultant
Dromedary Peak Consulting
Mesa, Arizona USA
And
Assistant Professor
Western International University
Phoenix, Arizona USA

Test strategy derivation must always be performed if testing is to be accomplished in a time and cost conscious manner. In the highly complex systems found in use today in information technology departments and web applications, a competent and pragmatic method must be employed to derive an efficient and effective test strategy. This strategy is always necessary if computer systems are to be comprehensively and accurately tested. Simply "banging on the box" or "flailing at the machine" will not do the job.

What is a test strategy? A literature search on the subject reveals that authors provide a wide range of opinions on this subject. Boris Beizer (1995) states that a test strategy should be focused on bug removal (page 8). Tim Koomen and Martin Pol (1999) state in a similar fashion, "the aim of the test strategy is finding the most important defects as early and as cheaply as possible" (page 83). In a work with another author, Martin Pol with Erik Van Veenendaal (1998), states in a similar vein, "the strategy should aim at optimizing the total amount of undetected defects" (page 56).

However, Bill Hetzel (1988) utilizes the term test strategies to describe test process techniques and infrastructure such as specification and design validation, structured module testing, top-down testing, test case library, test planning and reporting, and test facility and support (pages 242-243). In contrast to those two schools of thought, William Perry (1995) finds test strategy to be a reflection of whatever software development life cycle is used on a project (page 23). In yet another varying opinion, Elfriede Dustin, Jeff Rashka, and John Paul (1999) see automation as meeting for a test strategy (page 9). And finally, referring to Boris Beizer, in an earlier book (1990), he used test strategy as a white box testing option, i.e. path coverage (page 74).

All of these authors miss the total picture of what testing is supposed to accomplish and consequently what a test strategy should be. Testing should not be focused on identifying defects in the system, but should also provide corroboration that current requirement fulfillment has occurred. As a result, a testing strategy should:
1) Maximize defect identification through increased isolation.
2) Validate requirement delivery with tangible proofs.

However, testing strategy has other tasks due to the additional masters of effective process and cost efficiency. A test strategy should also:
3) Provide effectiveness in testing an unknown against a known, to promote confidence enlargement and risk reduction, by instrumenting incremental block testing with foundation and layers there upon.
4) Improve cost efficiency by decreasing the number of required test jigs to the minimum number needed to perform the total testing of the system.

All of these missions of a test strategy directly bear on what a test strategy must cover. A test strategy has to address not only the what, but the how of testing.

The strategy utilized for testing should be designed to measure requirements fulfillment, design integrity, and incorrect responses. The strategy has to do all of this in a systematic fashion by building confidence through always testing incremental functionality against validated and previously integrated functionality. By focusing this process on a sequence based on an analysis of the system architecture, minimal stubs and drivers, test jigs and tools, will have to be built, and needless costs will consequently be avoided.

The necessary sources for the derivation of the test strategy are not just the code of the delivered system and the user's guide, but all the documentation used to document the design and construction of the system. This documentation may be comprised of:
1) Technical Requirements Document
2) Service Level Agreements
3) System Architecture Document
4) Design Documents
5) Program or Project Plan Document
6) Integration Strategy Document
7) System Built Document

The Technical Requirements Document is the starting point for defining a test strategy. This document also goes under the names of System Requirements Specification, Software Requirements Specification, Functional Specification, or Software Requirement Document among others. All of the requirements found in this document must be addressed individually at separate points in the test strategy. Not only must correct responses for correct inputs in the implementation of the requirements be planned for, but also correct responses to incorrect inputs must be seen to as well. That is, the strategy must deal with boundary value analysis and error handling capability.

Service Level Agreements (SLA's) are another document that must be considered in laying out the test strategy. These SLA's, if in effect when the system is deployed, may impart additional requirements for functionality, performance, security, availability, and other system attributes if their content was not stated in the Technical Requirements Document. It is always best to include the SLA's in the analysis for the test strategy and to make sure that their subject matter has been reviewed.

The System Architecture Document must always be included as a source in preparing the test strategy for the system. This document, by delineating and describing the logical and physical architectures of the system will reveal the true structure and modularity of the system. Linkages, and interdependencies, as especially clarified through the review of data flow diagrams, may be used to construct the sequence of testing steps to be performed, and leverage points as a consequence. These leverage points are points that shape decisions that result in a tighter sequence of testing with less steps and more confidence in their foundations.

Design Documents are the documents that describe the algorithms and mechanics found in each component of the system. This documentation, in some cases, may be embedded in the code itself, but can be utilized all the same. The information, whatever the source, will affect the sequence of data processing as to the return of correct results. The consideration of design documentation can result in fewer steps for testing when their absence in prior integration may result in the need for more test jigs, and the strategy sequence avoids that need.

The programmer project plan also has an impact on the system test strategy and its derivation if the project plan calls out the viewing by clients of functionality prior to release, or incremental release of functionality to the client or users. If this viewing or incremental release is called out, the functionality should always be reviewed against the architecture of the system to gain understanding of dependencies and leverage points. In some cases, this viewing and consequent effect on test strategy may be able to be avoided if the viewing is not contractual and better options for the client or users can be demonstrated.

The Integration Strategy Document should help determine the testing sequence and test strategy. However, it is important to remember that the integration was affected by the needs of the developers and may have been tempered by the actual logistics of code availability due to schedule delays. With that said, a successful integration is prima facia evidence of a successful integration test of components, minimal as that may be.

The Build Document, in a similar sense, and as the recommended integration construction sequence for the system, provides a path for testing. This sequence may be modified for the test strategy, if a review of the architecture shows efficiencies to be gained through restructuring. Such restructuring, if found, should be fed back to the build engineer so that the System Build Document may be revised.

Documentation of the test strategy should be captured in the System Test Plan. In the System Test Plan, information should be recorded that carefully lists the goals or missions of the test strategy, the coverage, the sources, the sequence of testing as to the use of which test procedures at what point in the testing, and the resources necessary for implementation. The test procedures themselves should be reserved for the documentation concerning the specific activities they will accomplish. Any system testing direction should be referred to in the System Test Plan.

In conclusion, the derivation of the test strategy is an activity that must always be well thought out if all the viability of the system is to be reality without the expense of undue

time or money. By consideration of all of the missions of testing, and devoting adequate and effective test strategy will result. In the end, a well-prepared test strategy will provide competent testing, and always deliver a successful system to the users.

References

Beizer, Boris (1990). Software Testing Techniques. Long, England: Thomson Press.

Beizer, Boris (1995). Black-Box Testing – Techniques for Functional Testing of Software and System. New York: John Wiley and Sons.

Dustin, Elfriede; Rashka, Jeff; Paul, John (1999). Automated Software Testing – Introduction, Management, and Performance. Reading, Massachusetts: Addison-Wesley.

Hetzel, Bill (1988). The Complete Guide to Software Testing, (Second Edition). New York: John Wiley and Sons.

Koomen, Tim; Pol, Martin (1999). Test Process Improvement – A Practical Step-By-Step Guide to Structured Testing. Harlow, England: Addison-Wesley.

Perry, William (1995). Effective Methods for Software Testing. New York: John Wiley and Sons.

Pol, Marin; Van Veenendaal, Erik (1998). Structured Testing of Information Systems. Deventer, Netherlands: Kluwer.

QW2001 Standby Paper SB Management

Ms. Johanna Rothman
(The Rothman Consulting Group)

Successful Test Management: 10 Lessons Learned

Key Points

- Two main jobs of management
- Gain insight into how people respond to different management styles and missions
- Ideas for creating an effective work environment
- Ideas for rewarding and retaining staff

Presentation Abstract

Many engineering managers came to management through the technical ranks. Although they may have had plenty of engineering training and mentoring, they frequently have to learn management skills the hard way, through trial and error. This talk describes some technical management tips and tricks learned through trial and error, focused on test managers and their particular issues.

About the Author

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people.

Johanna publishes "Reflections", an acclaimed quarterly newsletter about managing product development. Johanna's handbook, "Hiring Technical People: A Guide to Hiring the Right People for the Job," has proved a boon to perplexed managers, as have her articles in Software Development, Cutter IT, IEEE Computer, Software Testing and Quality Engineering, and IEEE Software.

Johanna is the founder and principal of Rothman Consulting Group, Inc., and is a member of the clinical faculty of The Gordon Institute at Tufts University, a practical management degree program for engineers.

Successful Test Management: Ten Lessons Learned

Johanna Rothman
Rothman Consulting Group, Inc.
781-641-4046
jr@jrothman.com
www.jrothman.com

The Test Manager's Lament

- Started as an individual contributor
- No training
- "The people stuff is hard to do"
  - "the Management stuff isn't easy either"

 www.jrothman.com jr@jrothman.com 2

Managers Have Two Primary Roles

- Get the best work out of your people
- Create an environment that enables people to work

- Develop a strategy for how you're going to do this

 www.jrothman.com jr@jrothman.com 3

1. Know What They're Paying You to Do

- What's your mission—the reason they hired you? Mine generally is:

Assess the state of the product under development at any time and report on that state

- Other missions could be:

Find the Big Bad Bugs before our customers do

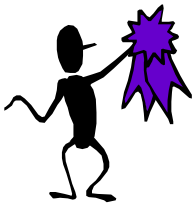
Create warm fuzzy feelings about our software

Test this beast until they pry it from my unwilling fingers

 www.jrothman.com jr@jrothman.com 4

2. Hire the Best People for the Job

- A manager's greatest point of leverage is in hiring appropriate staff
- "Best" is not necessarily synonymous with "Similar"
- Develop a hiring strategy
- Learn to interview successfully, so you can hire people who can do the job well (stars)



 www.jrothman.com jr@jrothman.com 5

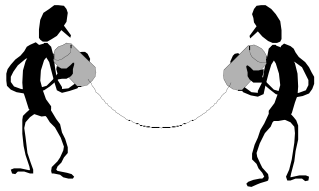
3. Uninterrupted Weekly Time With Each Person

- You need to know what your organization or project is doing
- You need to know what the people are doing, so you can create performance reviews

 www.jrothman.com jr@jrothman.com 6

One-on-Ones

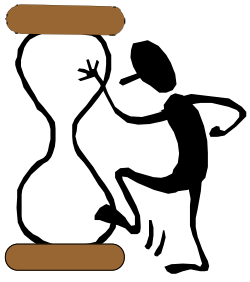
- With everyone, at a regular uninterrupted time
- We talk about
  - Their accomplishments
  - Their issues
  - If they need my help
  - Anything else they need to discuss



 www.jrothman.com jr@jrothman.com 7

But, I Don't Have Time to Meet With Everyone…

- You already are
- If you plan time, you can reduce the number of unplanned interruptions
- How will you give timely feedback on performance?



 www.jrothman.com jr@jrothman.com 8

4. Assume the Person Knows How to Do Their Job

- You *used* to know how to do the job
  - Do you really still know how?
- You hired the people because you thought they could do the work. Let them...
  - Give them assignments
  - Ask if they need help
  - Don't interfere
    - Sneak up behind them and ask "How's it going?
    - Micromanagement
    - Inflicting advice
- Choose a metric to know when you are stuck
  - This works for you too!

Successful (and Helpful) Managers

- Assign the work
  - Do they understand the work to do?
  - Do they have the tools required?
- Decide when to check in
- Supply help when requested, and not before

5. Treat People the Way They Want to Be Treated

- Everyone likes different projects
  - Specific tasks vs. general information
  - New complex problems vs. immediate success
- Everyone is motivated by different things
  - Money is not the only reward
  - Private thank-you's
  - Public recognition
  - M&Ms
  - Movie tickets
  - Team party
  - Ask the group

 www.jrothman.com jr@jrothman.com 11

6. Emphasize Results, Not Time

- Hours working do not positively correlate with productivity
- Permit (Force?) people to only work 40 hours per week
  - When they work longer, they do non-work things
  - Productivity goes down
  - If you keep people working only 40 hours per week, they work on work things

 www.jrothman.com jr@jrothman.com 12

Managing in a High-Interruption Environment

- "You can't get anything done here between 9am and 5pm"
- Can you cancel meetings?
- Can you or your staff reorganize the work?
- Observe results and obstacles to results
  - Easier to give accurate performance evaluations

Reward Results

- Plan for a 40-hour week, and reward the work done in that time

7. Admit Your Mistakes

- Mistakes are embarrassing
- If you admit mistakes, people respect you more
- Don't deny or ignore mistakes

 www.jrothman.com jr@jrothman.com 15

8. Commit to Projects After Asking Your Staff

- "Can we have this next month?"
- Even if you've already considered the request, the answer is "No"
  - In the moment, you might confuse this request with another request
  - There may be other implications you haven't considered, since it's no longer the same time you first considered this request.

 www.jrothman.com jr@jrothman.com 16

Don't Train Your Management to Ask You for an Answer

- Your staff will know that you think:
  - I want to know what it will really take you to do this work
  - I'm not afraid to tell my management what it will take

 www.jrothman.com jr@jrothman.com 17

9. Plan Training Time in the Workweek

- Engineering is constantly changing
- People generally like getting training
- Many inexpensive ways
  - Brown bag lunches
  - Periodic talks from other groups
  - Present projects across the company
  - In-house tool "user group" meeting
  - Outside experts
    - Professional consultants or speakers
    - Knowledgeable friend or colleague

 www.jrothman.com jr@jrothman.com 18

10. Plan the Testing

- Plan what you (your test group) can do
  - If you want to do more, plan how
- Develop test strategies for each product
- Develop release criteria for each project

 www.jrothman.com jr@jrothman.com 19

What Your Group Can Do

- What capabilities does your group have?
- What capabilities do you want to have in the group?

 www.jrothman.com jr@jrothman.com 20

Test Strategies

- Not every product is tested the same way
- As the manager, you get to choose how to test

Release Criteria

- Focus on what's *critically* important to this project
- Measurable ways to assess product ship decisions
- Assess risk of shipping
  - You aren't a gatekeeper
  - You don't have to stop shipment

Testers Can Make Great Managers

- Manage your management career the way you plan and develop tests
  - Develop a strategy
  - Identify how to manage your staff
  - Observe your own work
  - Make corrections and continue
- You don't have to be perfect
- Do enough right to help people do their best work in an environment they can work in

 www.jrothman.com jr@jrothman.com 23

QW2001 Standby Paper SB Applications

Dr. John D. Musa
(Consultant)

More Reliable Software Faster And Cheaper -- An Overview

Key Points

- SRE process overview
- Developing operational profiles to describe how customers will use your product
- Using operational profiles to increase development efficiency
- Determining the reliability / availability your customers need for your product
- Preparing for test, executing test, and guiding test

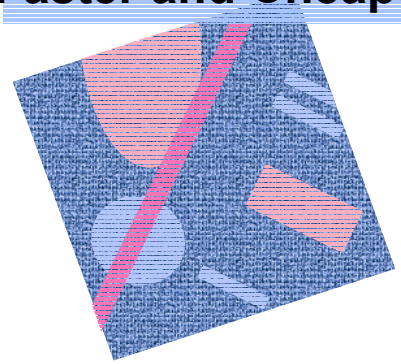
Presentation Abstract

Software reliability engineering (SRE) can help those who are stressed out by competitive pressures to produce more reliable software faster and cheaper. It is a standard, proven, widespread best practice with substantial benefits that has been used successfully by organizations such as Alcatel, AT&T, Bellcore, CNES (France), ENEA (Italy), Ericsson Telecom, France Telecom, Hewlett Packard, Hitachi, IBM, Lockheed-Martin, Lucent Technologies, Microsoft, MITRE, Motorola, NASA's Jet Propuls ion Laboratory and Space Shuttle Project, Nortel, Raytheon, Saab Military Aircraft, Tandem Computers, US Air Force, and US Marine Corps.

About the Author

John D. Musa is one of the creators of SRE, with more than 30 years varied and extensive experience as a software development practitioner and manager. Principal author of the highly-acclaimed pioneering book Software Reliability and author of the practical Software Reliability Engineering, Musa has published more than 100 papers on SRE. Elected IEEE Fellow in 1986 for many seminal contributions, he was recognized in 1992 as the leading contributor to testing technology. His leadership has been noted by every recent edition of Who's Who in America and American Men and Women of Science. Musa, widely recognized as a leader in SRE practice, initiated and led the effort that convinced AT&T to make SRE a "Best Current Practice." Musa has helped a wide variety of companies with a great diversity of software-based products deploy SRE. He is an experienced international speaker and teacher (over 200 major presentations) A founder of the IEEE Technical Committee on SRE, he is closely networked with SRE leaders, providing a broad perspective.

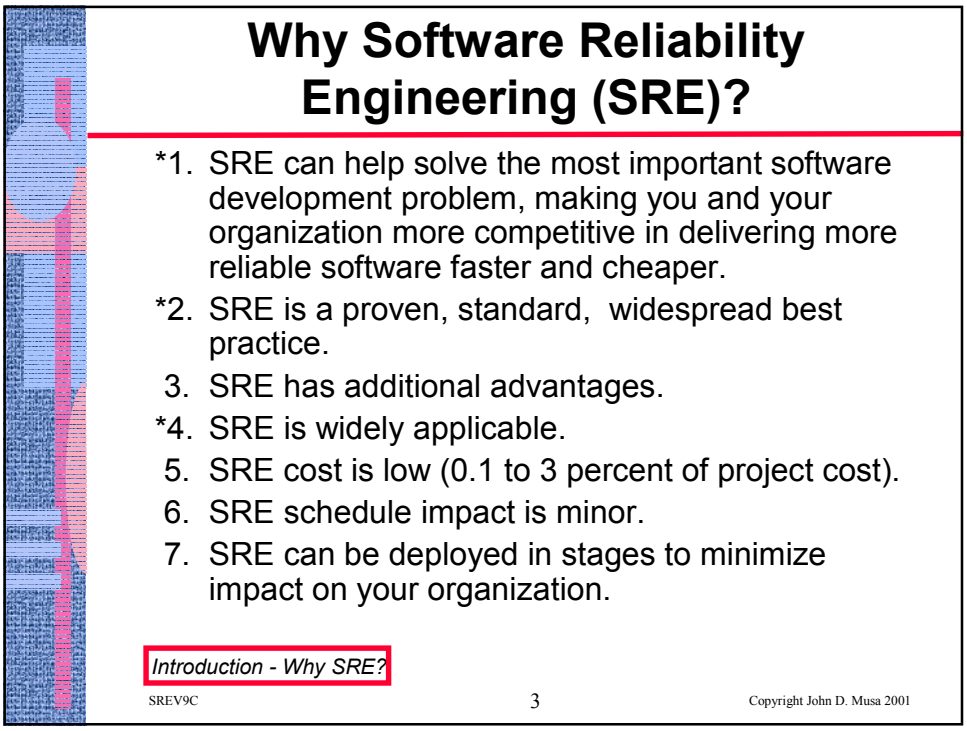
More Reliable Software Faster and Cheaper – An Overview



John D. Musa
j.musa@ieee.org

Outline

1. Why software reliability engineering (SRE)?
2. SRE process (with illustration)

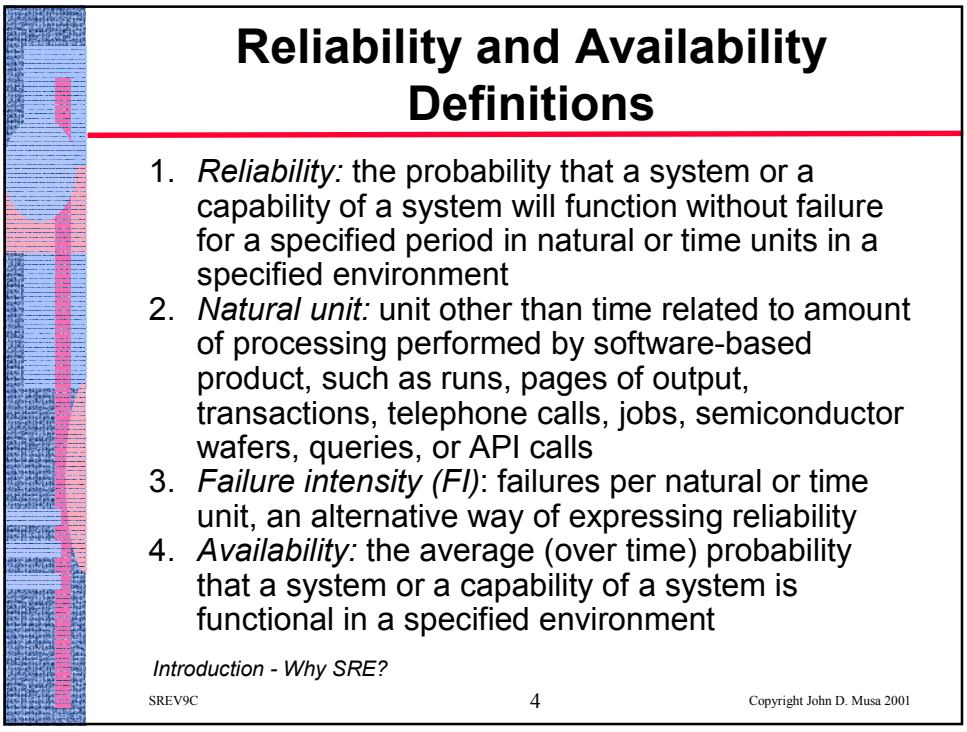


Why Software Reliability Engineering (SRE)?

*1. SRE can help solve the most important software development problem, making you and your organization more competitive in delivering more reliable software faster and cheaper.
*2. SRE is a proven, standard, widespread best practice.
3. SRE has additional advantages.
*4. SRE is widely applicable.
5. SRE cost is low (0.1 to 3 percent of project cost).
6. SRE schedule impact is minor.
7. SRE can be deployed in stages to minimize impact on your organization.

Introduction - Why SRE?

SREV9C 3



Reliability and Availability Definitions

1. *Reliability:* the probability that a system or a capability of a system will function without failure for a specified period in natural or time units in a specified environment
2. *Natural unit:* unit other than time related to amount of processing performed by software-based product, such as runs, pages of output, transactions, telephone calls, jobs, semiconductor wafers, queries, or API calls
3. *Failure intensity (FI)*: failures per natural or time unit, an alternative way of expressing reliability
4. *Availability:* the average (over time) probability that a system or a capability of a system is functional in a specified environment

Introduction - Why SRE?

SREV9C 4

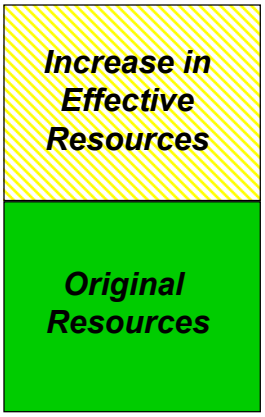
More Reliable Software Faster and Cheaper

1. Customers want (in order):
   A. Reliability and/or availability
   B. Faster delivery
   C. Lower cost
2. Success in meeting demands affects market share, profitability
3. Demands conflict, causing risk and overwhelming pressure
4. The practice of SRE resolves conflicting demands efficiently by quantitatively guiding development and test of software-based systems

Introduction - Why SRE? - Problem

SREV9C 5

How Does SRE Work?

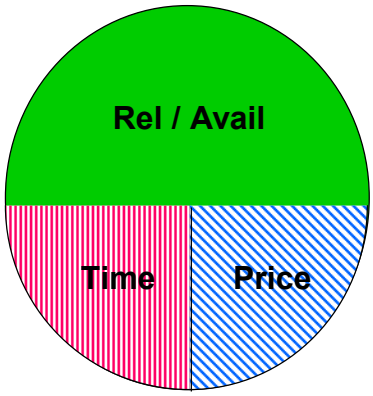


1. Increase effective resources
   A. Quantitatively characterize expected use
   B. Focus resources on most used and most critical functions
   C. Maximize test effectiveness by making test highly representative of field

Introduction - Why SRE? - Problem

SREV9C 6

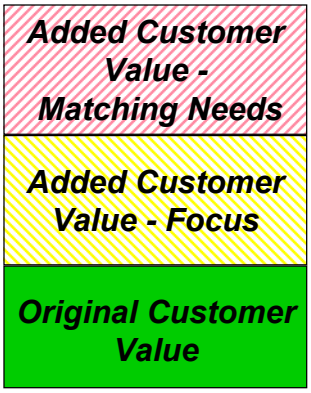
How Does SRE Work?



2. Apply resources to maximize customer value
   A. Set quantitative objectives for major quality characteristics (reliability and/or availability, delivery time, price)

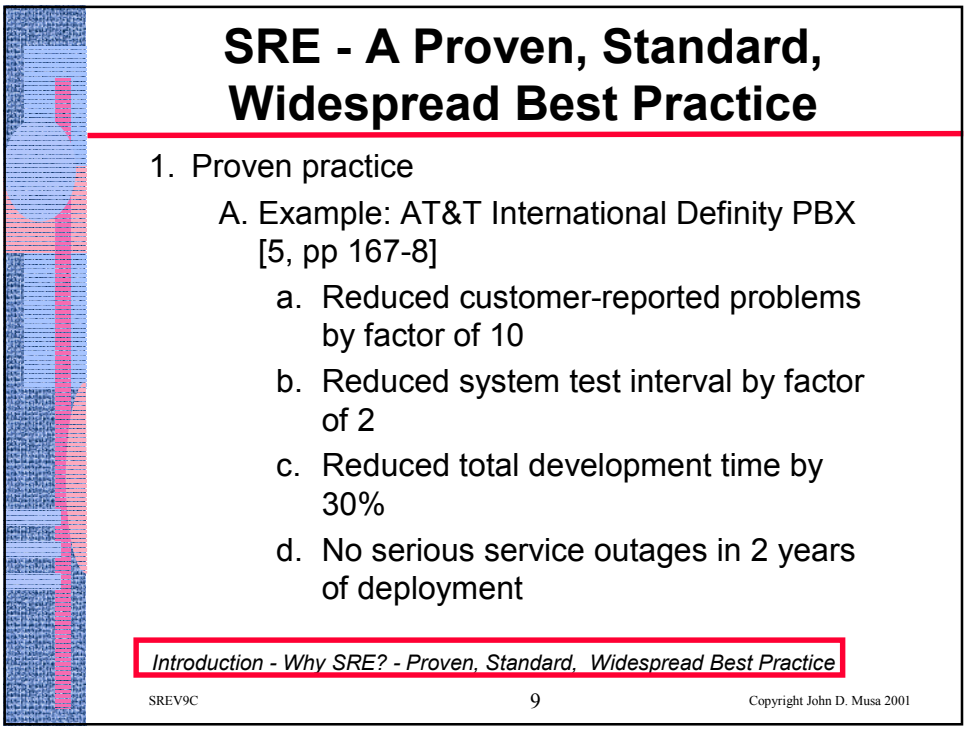
Introduction - Why SRE? - Problem
SREV9C 7

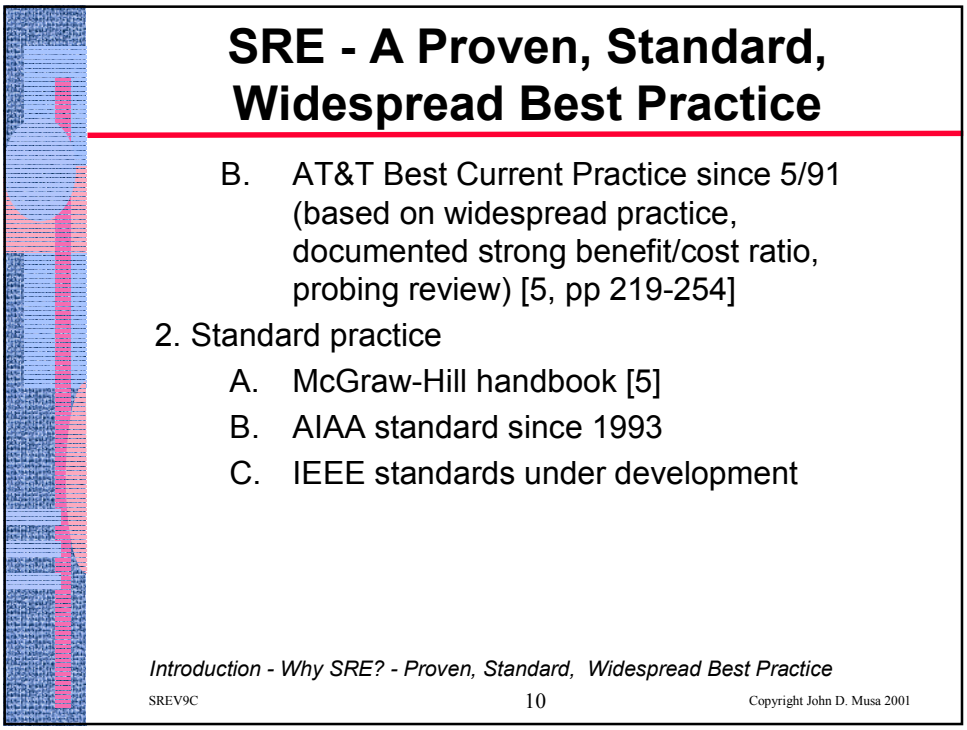
How Does SRE Work?

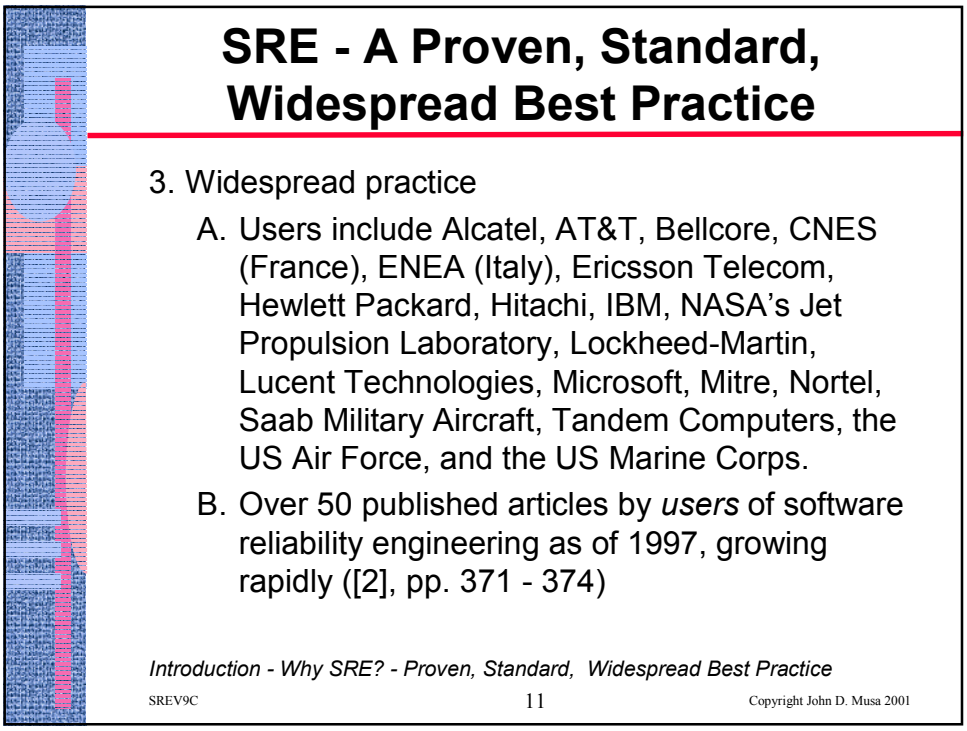


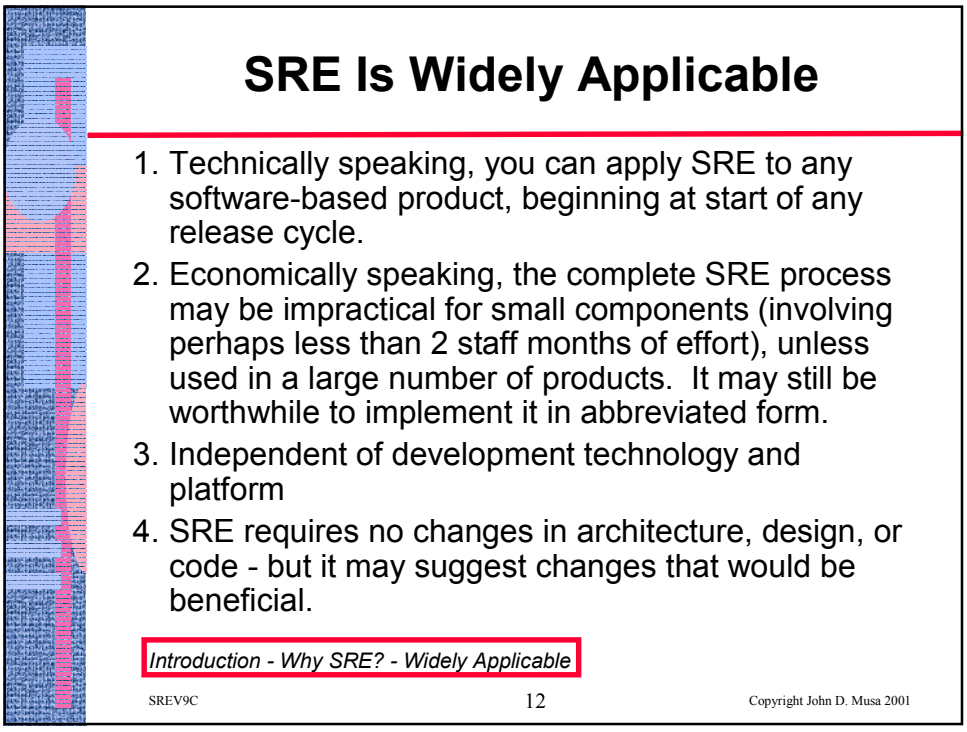
B. Engineer strategies to meet objectives
C. Track reliability in system test against objective as one of the release criteria

Introduction - Why SRE? - Problem
SREV9C 8

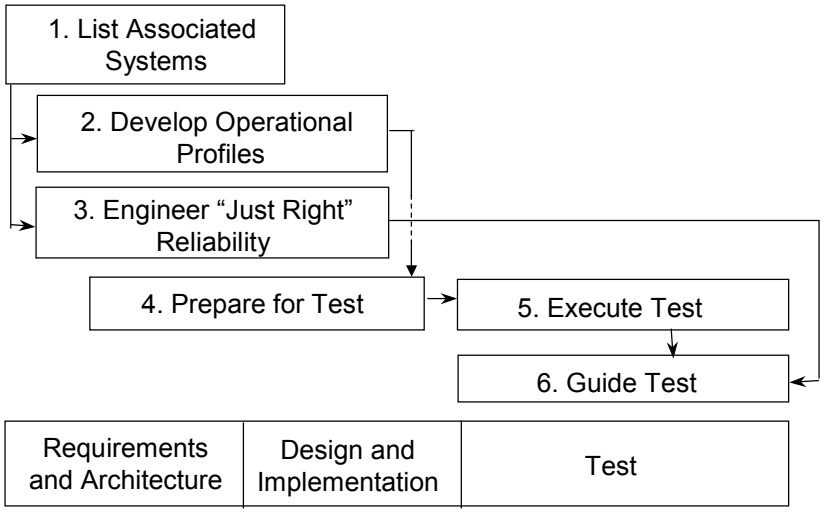





SRE - A Proven, Standard, Widespread Best Practice
3. Widespread practice
A. Users include Alcatel, AT&T, Bellcore, CNES (France), ENEA (Italy), Ericsson Telecom, Hewlett Packard, Hitachi, IBM, NASA's Jet Propulsion Laboratory, Lockheed-Martin, Lucent Technologies, Microsoft, Mitre, Nortel, Saab Military Aircraft, Tandem Computers, the US Air Force, and the US Marine Corps.
B. Over 50 published articles by users of software reliability engineering as of 1997, growing rapidly ([2], pp. 371 - 374)
Introduction - Why SRE? - Proven, Standard, Widespread Best Practice
SREV9C
11
Copyright John D. Musa 2001


SRE Is Widely Applicable
1. Technically speaking, you can apply SRE to any software-based product, beginning at start of any release cycle.
2. Economically speaking, the complete SRE process may be impractical for small components (involving perhaps less than 2 staff months of effort), unless used in a large number of products. It may still be worthwhile to implement it in abbreviated form.
3. Independent of development technology and platform
4. SRE requires no changes in architecture, design, or code - but it may suggest changes that would be beneficial.
Introduction - Why SRE? - Widely Applicable
SREV9C
12
Copyright John D. Musa 2001

Activities of SRE Process and Relation to Software Development Process



SRE Process

SREV9C 13

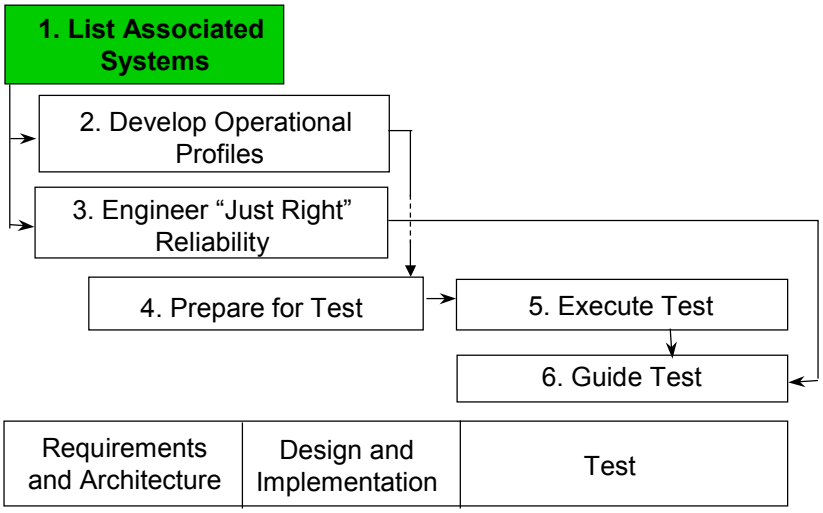
Illustration - FONE FOLLOWER (FF) - Product Description

1. Subscriber calls FF, enters planned phone numbers (forwardees) to which calls are to be forwarded vs time.
2. FF forwards incoming calls (voice or fax) from network to subscriber as per program. Incomplete voice calls go to pager (if subscriber has one) and then voice mail.

SRE Process

SREV9C 14

Activities of SRE Process and Relation to Software Development Process



List Associated Systems

SREV9C 15

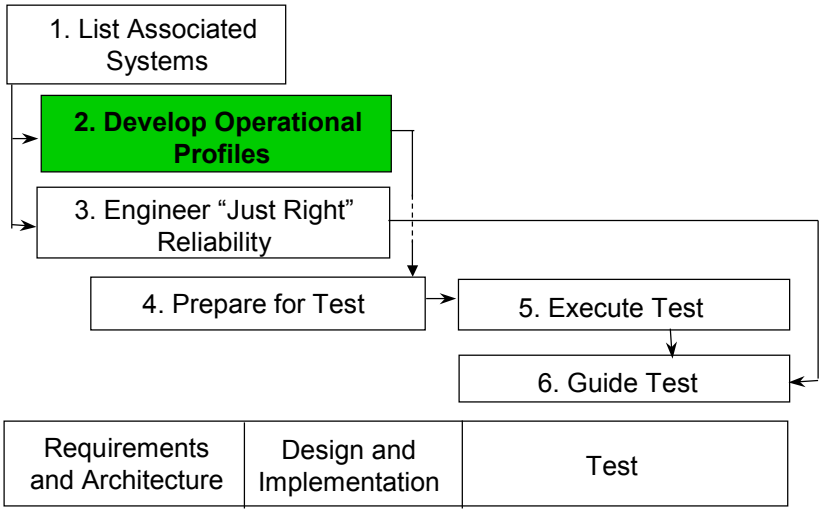
List Associated Systems of Product

1. Base product
2. Major variations of base product (for substantially different environments, platforms, or configurations)
3. Frequently used supersystems of base product or variations

List Associated Systems

SREV9C 16

Activities of SRE Process and Relation to Software Development Process



Develop Operational Profiles

SREV9C 17

Operation

Operation: major system logical task of short duration, which returns control to system when complete, and whose processing is substantially different from other operations

Illustrations - FF:

Process fax call, Phone number entry, Audit section of phone number database

Develop Operational Profiles - Concepts

SREV9C 18

Operational Profile

Operational profile (OP): complete set of operations with probabilities of occurrence

Illustration - FF:

Operation	Occur. Prob.
Process voice call, no pager, ans.	0.21
Process voice call, pager, ans.	0.19
Process fax call	0.17
Process voice call, pager, ans. on page	0.13
⋮	
	1

Develop Operational Profiles - Concepts

SREV9C 19

Develop Operational Profiles - Step by Step

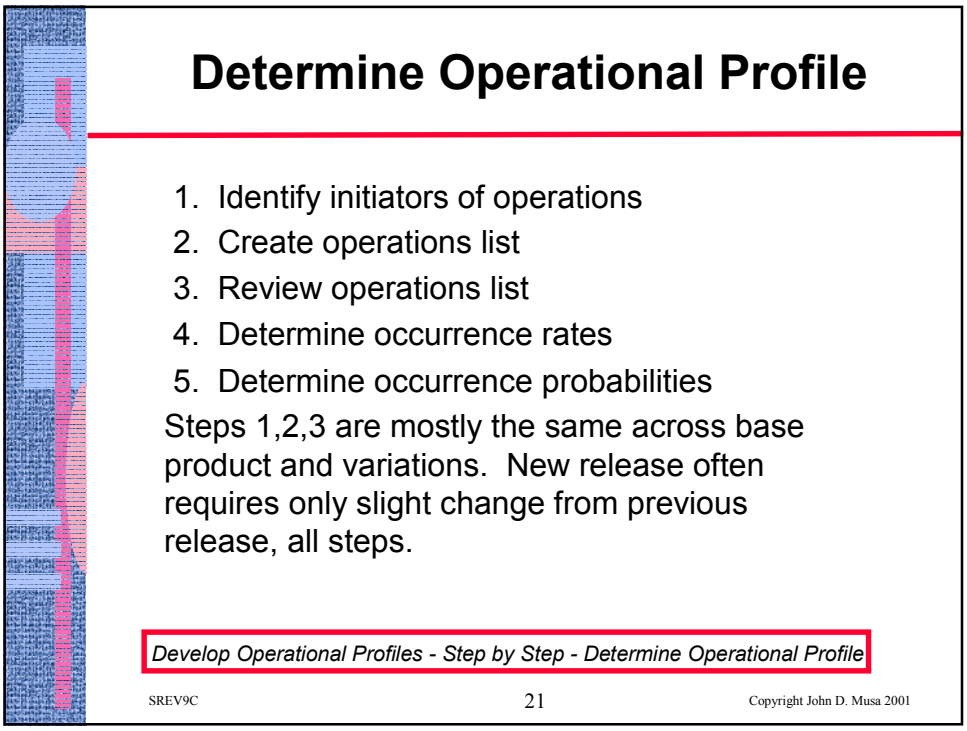
For base product and each variation:

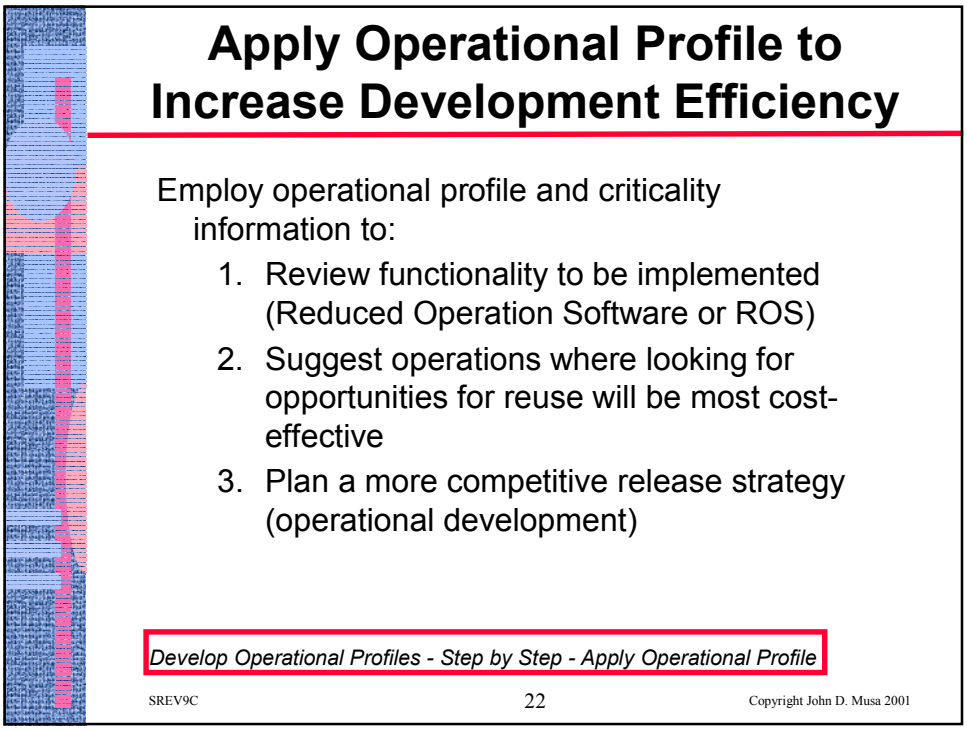
*1. Determine operational profile
*2. For any software you are developing, apply operational profile to increase development efficiency

Operational profiles of supersystems are same as those of their base product or variations.

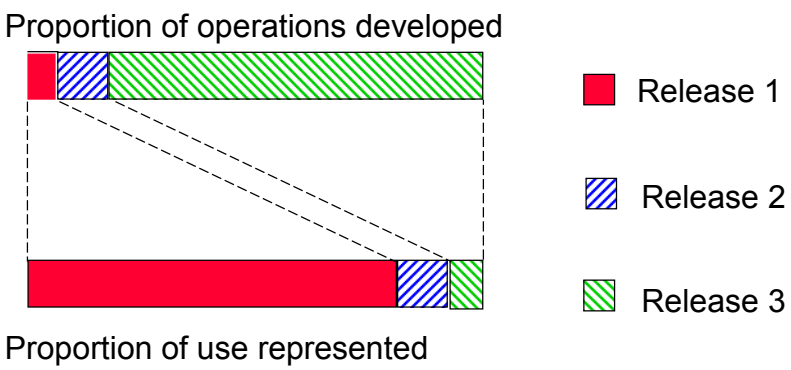
Develop Operational Profiles - Step by Step

SREV9C 20





Operational Development - Illustration



Develop Operational Profiles - Step by Step - Apply Operational Profile

SREV9C 23

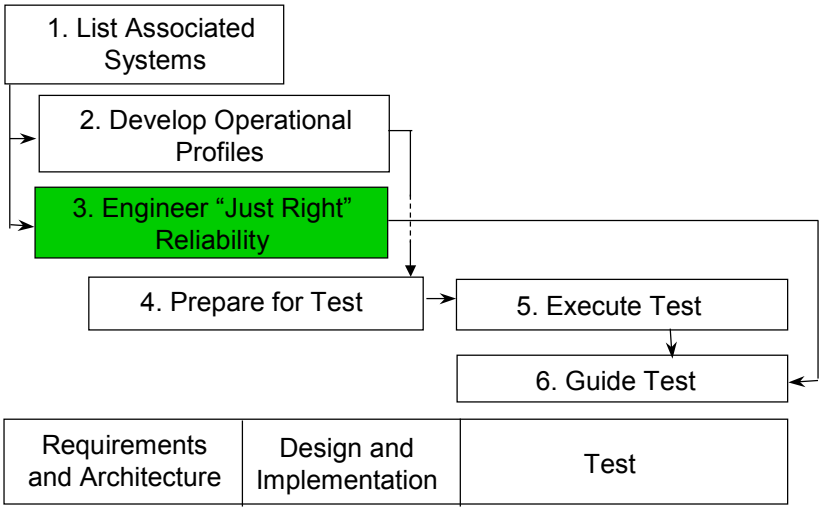
Apply Operational Profile to Increase Development Efficiency

4. Allocate resources for requirements, design, code reviews among operations to cut schedules and costs
5. Allocate system engineering, architectural design, development, and code resources among operations to cut schedules and costs (test resources will be allocated among operations in Prepare for Test, Execute Test modules)
6. Allocate development, code, and test resources among modules to cut schedules and costs ([2], pp. 118 - 119)

Develop Operational Profiles - Step by Step - Apply Operational Profile

SREV9C 24

Activities of SRE Process and Relation to Software Development Process



Engineer "Just Right" Reliability

SREV9C 25

Failure and Fault

Failure	Fault
Departure of system behavior in execution from user needs	Defect in system implementation that causes the failure when executed
User-oriented	Developer-oriented

Engineer "Just Right" Reliability - Concepts

SREV9C 26

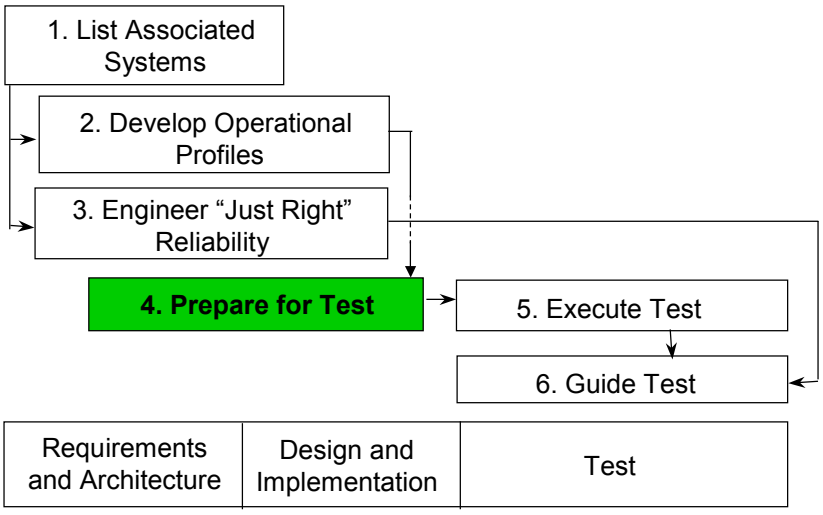
Engineer "Just Right" Reliability - Step by Step

1. Define failure consistently over life of product release and clarify with examples
2. Choose common reference measure for all failure intensities
3. Set system FIO for each associated system
4. For any software you develop:
   A. Find developed software FIO
   B. Choose software reliability strategies to meet developed software FIO and schedule objectives with lowest development cost

Engineer "Just Right" Reliability - Step by Step

SREV9C 27 Copyright John D. Musa 2001

Activities of SRE Process and Relation to Software Development Process



Prepare for Test

SREV9C 28 Copyright John D. Musa 2001

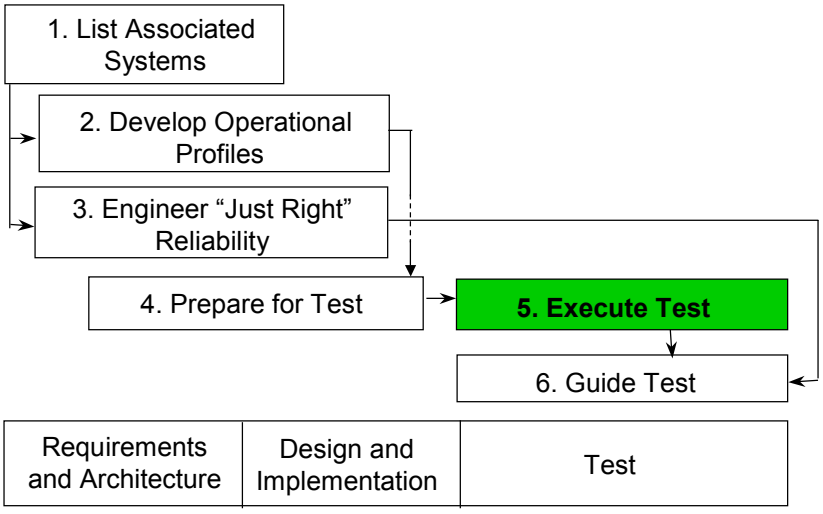
Prepare for Test - Step by Step

1. Specify test cases
   A. Allocate test cases to be developed to operations based on operational profile
      Illustration - FF:
      Allocate 17% of test cases to Process fax call operation
   B. Detail test cases within operation by selecting from possible choices with equal probability
      Illustration - FF:
      Forwardee = Local calling area
2. Specify test procedure, based on the test operational profile

Prepare for Test - Step by Step

SREV9C 29

Activities of SRE Process and Relation to Software Development Process



Execute Test

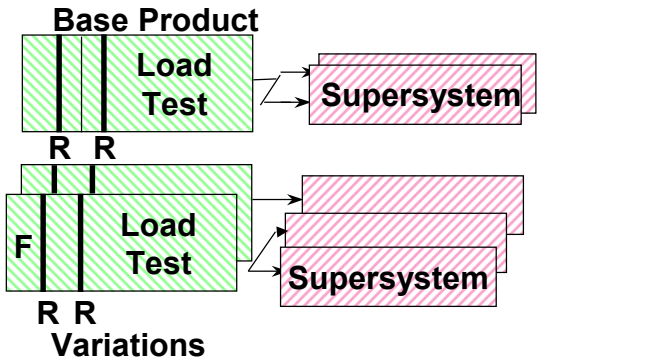
SREV9C 30

Execute Test - Step by Step

1. Determine and allocate test time among associated systems and types of test (feature, load, regression)
*2. Invoke test in accordance with operational profile
3. Identify system failures and when they occurred - use data in Guide Test

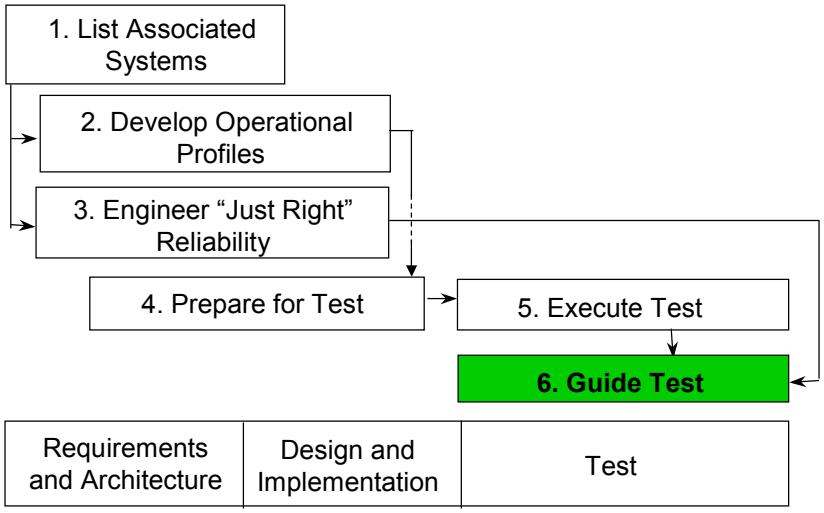
Execute Test - Step by Step

Invoke Test



Execute Test - Step by Step - Invoke Test

Activities of SRE Process and Relation to Software Development Process



Guide Test

SREV9C 33

Guide Test - Step by Step

Process system failure data gathered in test to:

*1. Certify reliability of assurable acquired components, supersystems, base product and variations that customers will acceptance test

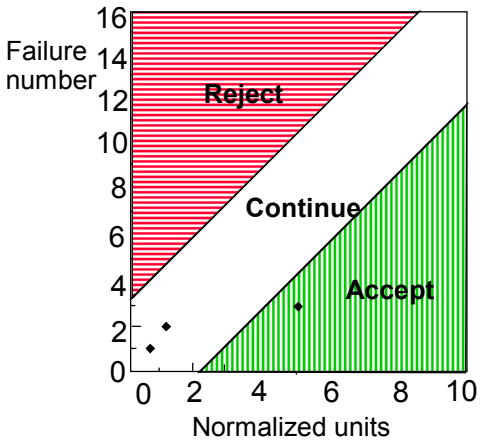
*2. Track reliability growth of developed base product and variations

3. Guide product release

Guide Test - Step by Step

SREV9C 34

Certify Reliability - Demonstration Chart



Fail. No.	Mcalls at Failure	Normalized Units
1	0.00375	0.75
2	0.00625	1.25
3	0.025	5

Failure intensity objective:
200 failures / Mcalls

Guide Test - Step by Step - Certify Reliability

SREV9C 35 Copyright John D. Musa 2001

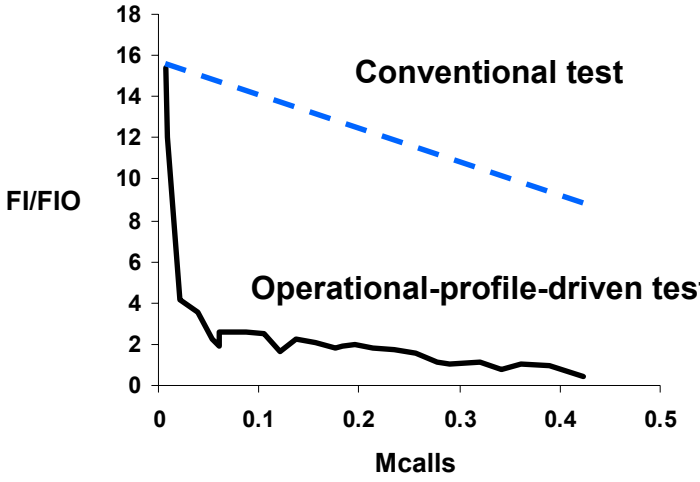
Track Reliability Growth

1. Execute CASRE software reliability estimation program to obtain FI / FIO ratio
2. Plot FI / FIO ratio against time
*3. Interpret plot

Guide Test - Step by Step - Track Reliability Growth

SREV9C 36 Copyright John D. Musa 2001

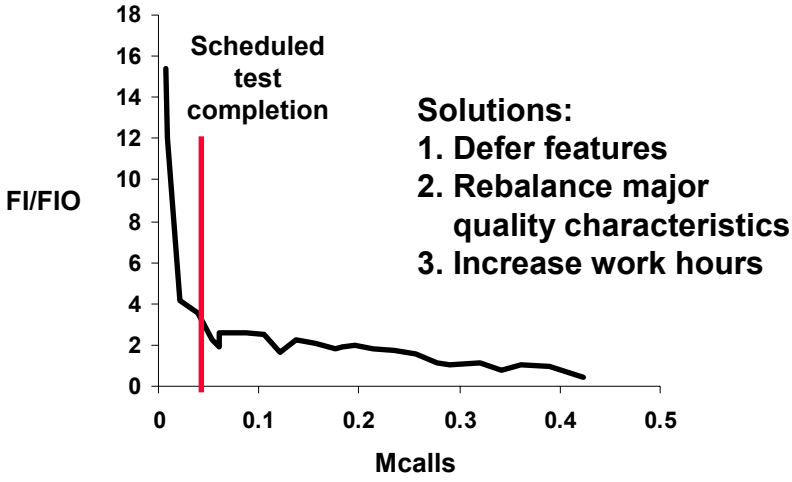
Interpret Plot : Illustration - FF



Guide Test - Step by Step - Track Reliability Growth

SREV9C 37

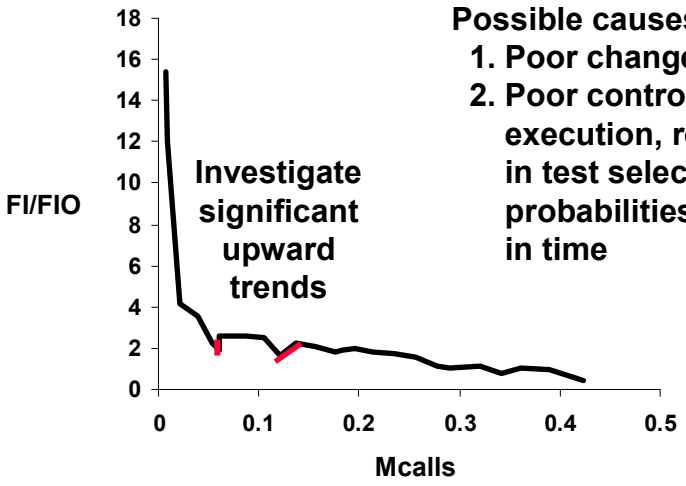
Interpret Plot : Illustration - FF



Guide Test - Step by Step - Track Reliability Growth

SREV9C 38

Interpret Plot : Illustration - FF

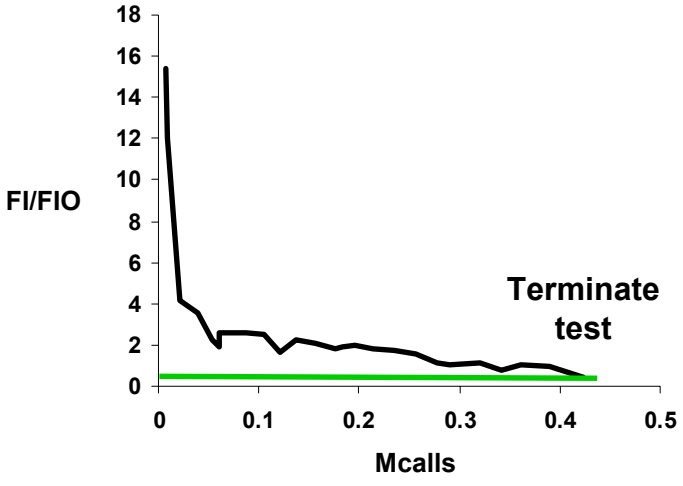


Possible causes:
1. **Poor change control**
2. **Poor control of test execution, resulting in test selection probabilities varying in time**

Guide Test - Step by Step - Track Reliability Growth

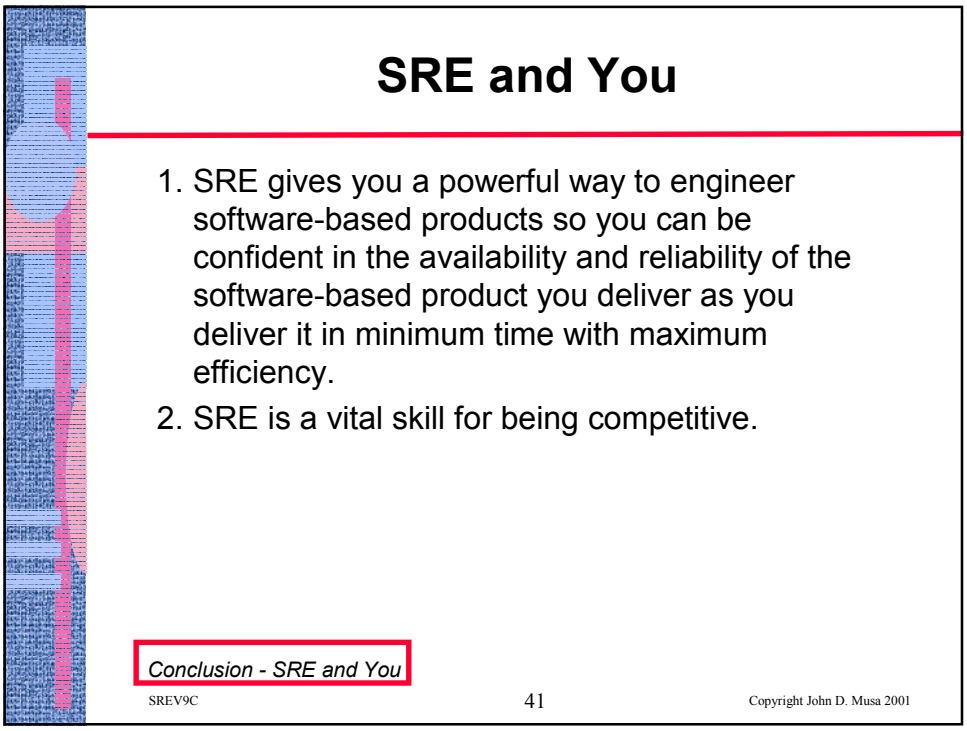
SREV9C 39

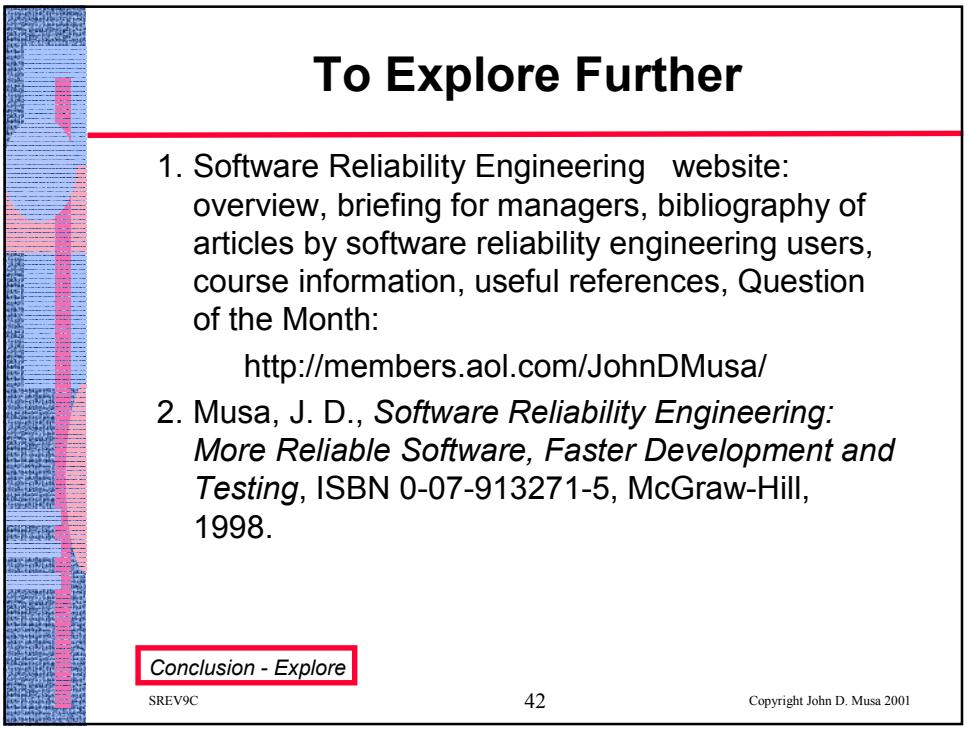
Interpret Plot : Illustration - FF

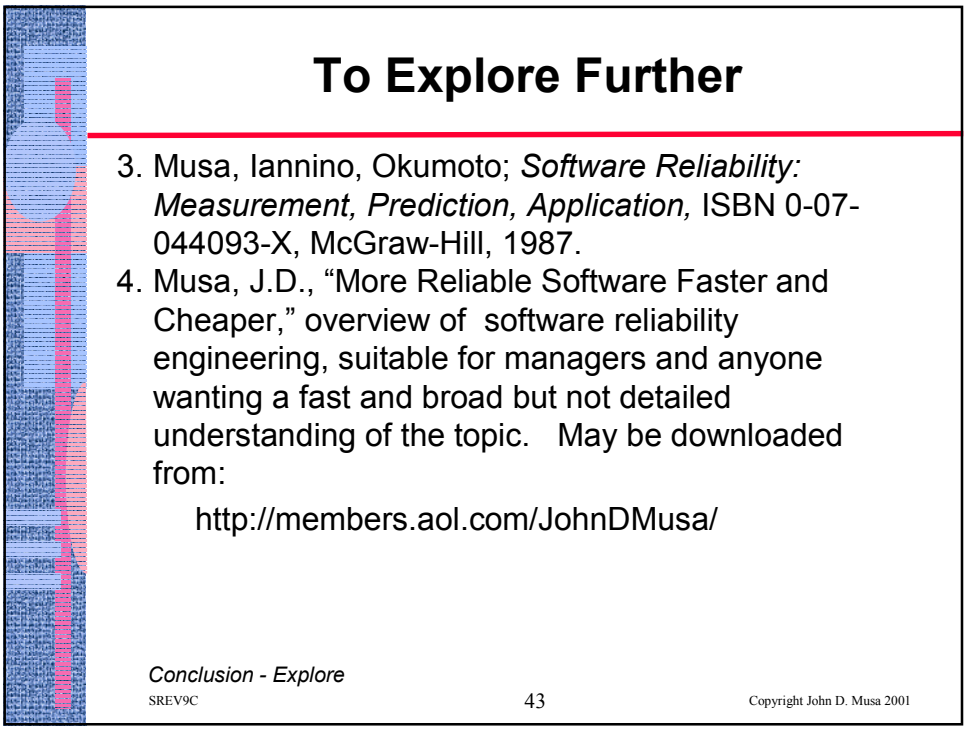


Guide Test - Step by Step - Track Reliability Growth

SREV9C 40





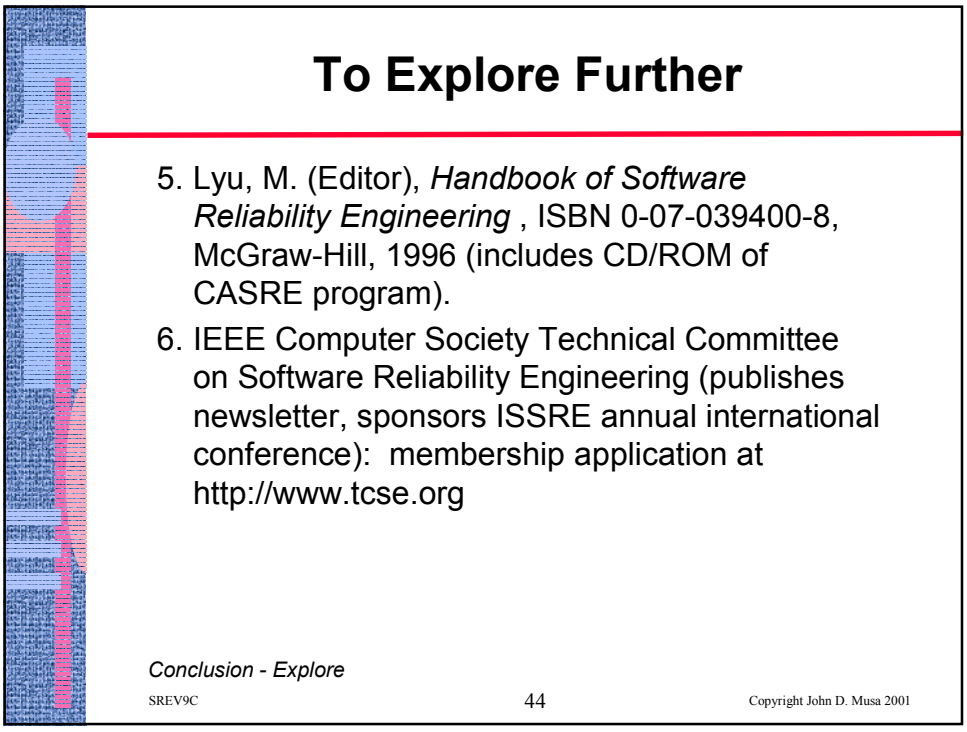


To Explore Further

3. Musa, Iannino, Okumoto; *Software Reliability: Measurement, Prediction, Application,* ISBN 0-07-044093-X, McGraw-Hill, 1987.
4. Musa, J.D., "More Reliable Software Faster and Cheaper," overview of software reliability engineering, suitable for managers and anyone wanting a fast and broad but not detailed understanding of the topic. May be downloaded from:

   http://members.aol.com/JohnDMusa/

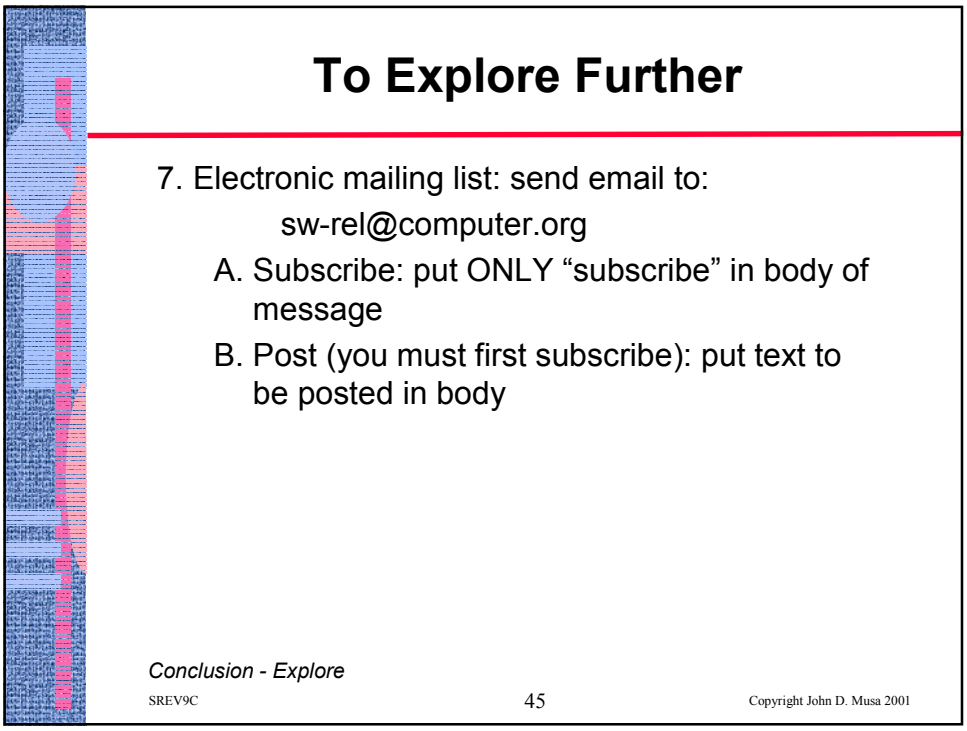
Conclusion - Explore
SREV9C 43



To Explore Further

5. Lyu, M. (Editor), *Handbook of Software Reliability Engineering* , ISBN 0-07-039400-8, McGraw-Hill, 1996 (includes CD/ROM of CASRE program).
6. IEEE Computer Society Technical Committee on Software Reliability Engineering (publishes newsletter, sponsors ISSRE annual international conference): membership application at http://www.tcse.org

Conclusion - Explore
SREV9C 44



QW2001 Standby Paper SB Internet

Dr. Kobad Bugwadia & Mr. Kris Mohan
(Intel Corporation)

Quality Issues, Requirements & Challenges for Multimedia Streaming on the Internet

Key Points

- Multimedia Streaming - User Expectations & Requirements
- Streaming Quality Benchmarks
- New Challenges/approaches for improving Streaming Quality

Presentation Abstract

The total number of broadband subscribers is expected to grow to 28 million US homes by 2004, according to a new Gartner Dataquest study (Nov. 2000). However the study also found that the quality of the content available via broadband will eventually determine its long-term success.

Thus, the value of broadband will not be in the access itself, which will be commoditized over time, but in the delivery of multimedia, voice and other useful and timesaving applications and services that these connections enable. The study concluded that among the most compelling broadband applications will be TV-based Quality streaming video and audio services.

About the Author

Kobad A. Bugwadia has M.S. and Ph.D. degrees in Electrical Engineering from Rutgers University, NJ. He is currently employed at Intel Corporation where his focus is on Internet Technology - Quality & Reliability. He received the IEEE Best Paper Award for his paper published in the 1996 IEEE Transactions on Consumer Electronics and has currently two outstanding U.S. and International Patents. He is a member of the IEEE and the Tau Beta Phi Honor Society.

Kris Mohan is Corporate Manager for Internet Technology Reliability at Intel. He has over 20 years of industry experience in the areas of networking and semiconductor technology. He is also an adjunct professor of Networking at Univ. of Santa Clara and San Jose State Univ. where he teaches graduate level courses in advance computer networking. He has published several papers and chaired several industry conferences.

Quality Issues, Requirements & Challenges for Multimedia Streaming on the Internet

Kobad Bugwadia & Kris Mohan
Intel Corporation

Streaming Technology – A Compelling Application for Broadband Internet !

- Total number of Broadband subscribers is expected to grow to 28 million US homes by 2004 (Gartner Dataquest study – Nov. 2000)
- However Quality/ Delivery of applications & services available via Broadband will eventually determine its long-term success
- Much of the Bandwidth/Broadband growth will be driven by TV-based Quality Streaming audio/video services
- Higher Bandwidth improves user experience, increasing its need/use

Streaming Technique & Overview

- Streaming refers to the continuous transmission of digital audio or video over the internet
- A small portion (a few seconds) of the file gets downloaded at the client side to a buffer before playback
- While this portion is decompressed and begins playing subsequent portions of the multimedia file are downloaded
- At no point in time is the complete multimedia file available to the client

Streaming Media Software

- Streaming Media Capture & Encoding Software
  - Enables transform of audio/visual media into files appropriate for streaming
  - Real Producer G2, Windows Media Encoder/ On-Demand Producer, QuickTime Pro
- Streaming Media Server Software
  - Large file sizes requiring real-time transmissions (Media Servers)
  - Real Networks, Microsoft, and Apple provide on-demand and live streaming Server Software
- Streaming Media Player Software
  - Real Player, Windows Media Player, QuickTime Player

Expectations and Requirements (Consumer Space)

Requirements:

- Ability for user to upload and stream home movies/videos to friends and family

Expectations:

- Simplicity
- Ease of use (Nice and easy to follow GUI)
- Technical details transparent to the user
- Low to No cost to user (already free services available)

Expectations and Requirements (Business/Enterprise Space)

Requirements:

- To provide an added advantage for increasing sales/value of e-businesses -- used for product demonstrations & presentations

Expectations:

- Higher quality of video and audio
- Easy integration within a clients existing web-site / applications
- More control features/options/choices
- Scalability as the number of users increases
- Robust & Secure service

Streaming Quality Measurement/Benchmark

- A Brand new Rating System introduced by Keynote Systems and supported by group of streaming industry leaders (Oct. 2000)
- Measures/ quantifies streaming quality and performance as experienced by end-users – Improving Quality through Measurement !
- Comprises of the :
  * Streaming Scale
  * Streaming Index

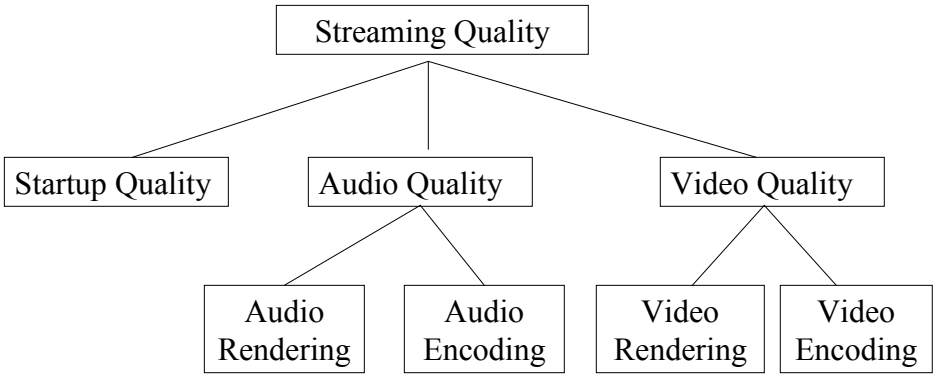
Streaming Quality Scale

- Assesses quality of audio and video streams on the internet
- A scale from one to ten with 10 representing DVD Quality (as seen on TV)
- DVD Quality – 720 x 480 frame size at 29.97 fps
- Logically combines the quality factors affecting streaming in different variations and degrees
- Current highest possible score about 6.0 – ideal delivery of a 300 kbps video stream at 20 fps

Streaming Quality Score

- Numerical score on the Streaming Scale of 1 to 10
- Score incorporates playback quality of both audio & video portion of the stream and is weighted based on the type of encoding used on the original content
- Logically combines the relevant encoding and delivery factors that affect stream quality
- Includes : connect time, redirect time, initial buffer time, video frame rate, late, lost and dropped packets and BW utilization

Keynote Streaming Metrics



Streaming Quality Measurement Metrics

- The "Encoding" metric measures the quality of the original, compressed stream before transmission
  - What you have created e.g. a postage stamp size video at 5 fps
  - Encoding is prior to Server
- The "Rendering" metric measures the degradation due to transmission
  - What you actually receive/see
  - Rendering is Server + Network + Client
- Startup time includes network connection, redirection, and initial buffering

Present Streaming Media Quality

(Streaming Quality vs. Available Bandwidth)

Barely Acceptable (28k & 56k modem users – 83% US home internet *)
- Approx 160 x 120 resolution
- 5 to 8 fps

Generally Considered Acceptable (56k & Broadband)
- 160 x 120 or 320 x 240 resolution (VHS Quality)
- 8 to 13 fps

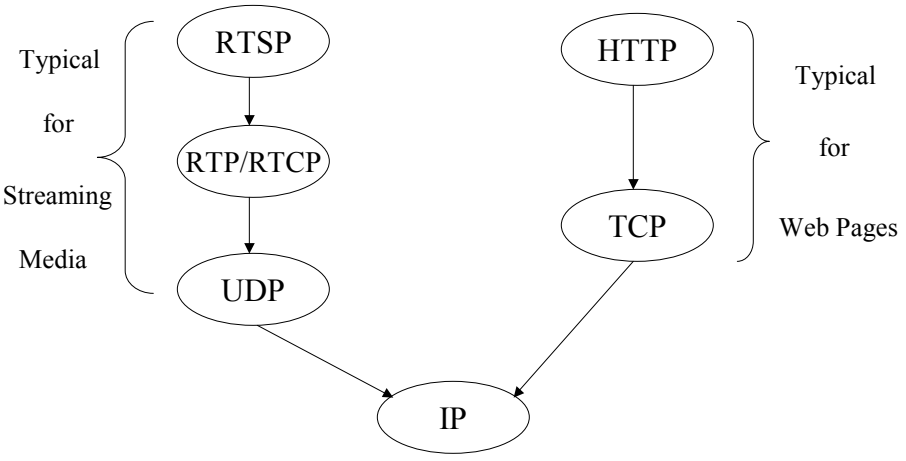
High Quality (Broadband users Only – 12% of US home internet connections *)
- 100 kbps or greater
- 320 x 240 or 720 x 480 resolution (DVD Quality)
- 15 to 20 fps

Streaming Quality Goal (Full Broadcast DVD Quality)
- 720 x 480 resolution (DVD Quality)
- 29.97 fps (Full Broadcast Quality)

Factors Affecting Streaming Quality/ User Experience

- CODEC Strategies/ Converting Content into Streaming Media
  - Compressing a file/multimedia content
  - Makes audio/video files smaller
  - Decompressed in real time during play-back
  - Most Commonly used : H.263 and MPEG-4
  - Exploit both Spatial and Temporal redundancies – DCT, MC Prediction
- Transmission Strategies
  - Serving & Delivering the Media
  - Multimedia Protocols - RTSP/RTP/UDP
  - Distributed Networking – Content Distribution Network

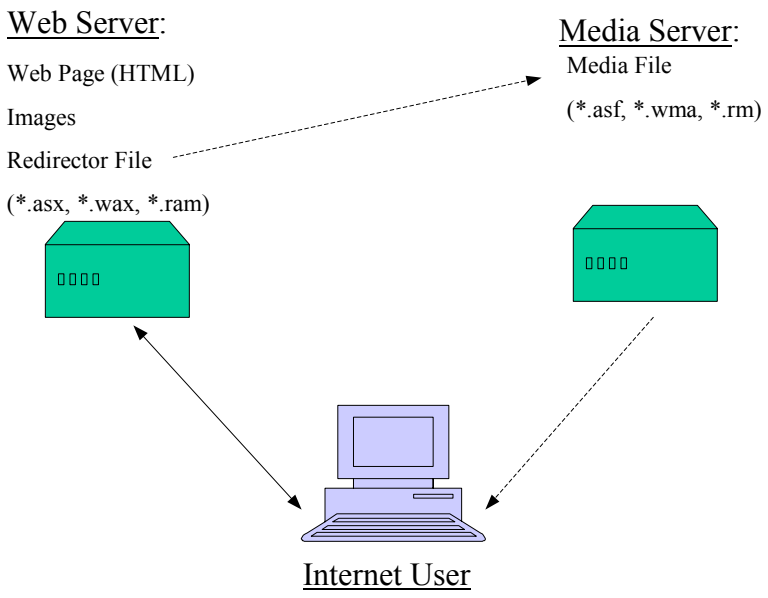
Transmission of Streaming Media



Transmission of Streaming Media

- Just as HTTP transports HTML – RTSP is generally used to transport multimedia data
- It is architecturally located above RTP & RTCP
- RTSP accomplishes data transfer using RTP
- RTP itself can use Transmission Control Protocol (TCP) or User Datagram Protocol (UDP)
  - UDP is lossy, but continues in case of lost packets
  - TCP does not allow lost packets, slower to start and more likely to rebuffer

Streaming-Media Delivery Process



Serving and Delivering the Media

- The delivery process involves 2 servers
  - One servers the HTML pages (Web Server) – via HTTP
  - Another serves the streaming media (Media Server) – via RTSP/RTP/UDP
- Client sends request for streaming media to HTML server
- Web server redirects request to media server via redirector or pointer file – placed in the same folder as HTML page
- Media server streams the media files to client

Current State of Streaming Technology

- Processor & BW Performance continue to grow - - - -
- However, 28.8/56 Kbps modems makes up 83% of all home internet connections (Dec. 2000) *
- BW cost & limited availability to home require technology solution & optimization
- Currently due to BW limitations, industry avoids using a very high encoding quality to prevent resulting in poor rendering quality
- Improvements in quality, ease-of-use, and accessibility must continue if streaming consumption is to become as commonplace as broadcast or cable television

* Nielsen//NetRatings

Limitations/Causes for Degradation in Streaming Quality

- Lossy Compression/Encoding
- Overloaded Servers
- Problems in Internet Caching & in transmission to last mile
- Problems within the last mile
- Decoder Software inefficiencies
- End user Computer too slow to decode properly

Challenges for Achieving High Quality Streaming

- Intelligent Multimedia Content Delivery
  - Speed Content Delivery
- Intelligent Content Management
  - Ease Content Management
- High Performance Compression Algorithms
- Achieve Optimal Cooperation between Client and Server – best possible performance given the network BW at a given time

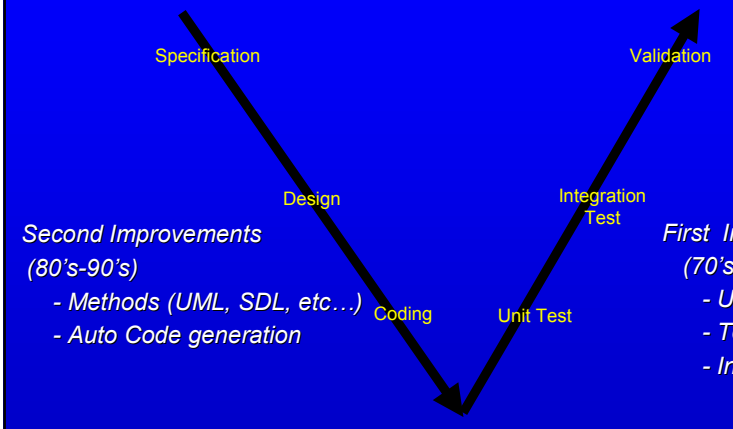
PolySpace TECHNOLOGIES

Bug Detection Tools for Productivity

Based on Abstract Interpretation

PolySpace TECHNOLOGIES

SW Development Process

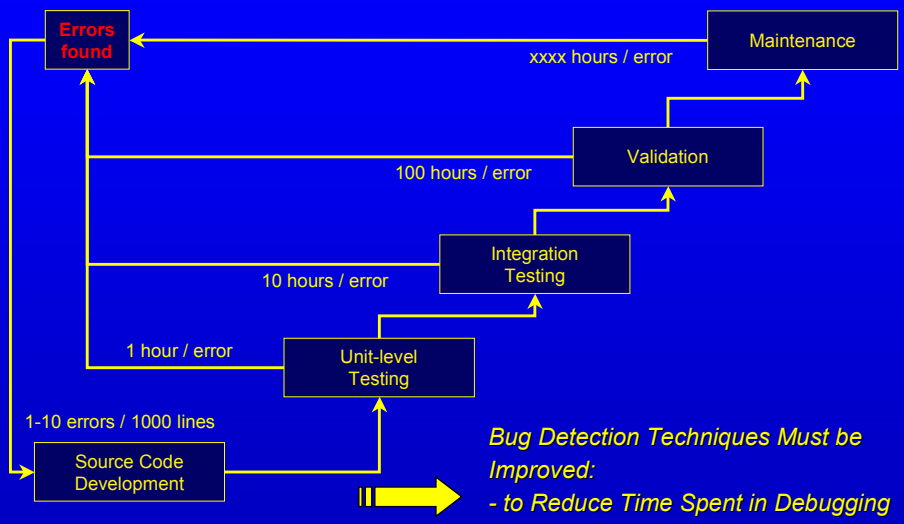


Second Improvements (80's-90's)
- *Methods (UML, SDL, etc…)*
- *Auto Code generation*

First Improvements (70's-80's)
- *Unit Testing*
- *Test Coverage*
- *Instrumentation*

Next Improvements will Come From the Testing/Validation Phases

Error Detection and Correction Costs



Bug Detection Techniques Must be Improved:
- to Reduce Time Spent in Debugging
- to Improve Functional Testing Reliability

What is a Run-Time Error?

They are well defined software errors that cause non determinisms, incorrect results or processor halt

- ***De-Referencing through Null, Out-of-Bound Pointers***
- ***Out-of-Bounds Array Access***
- ***Read Access to Non-Initialized Data***
- ***Access Conflict on Shared Data (Multi-Tasks Applications)***
- ***Invalid Arithmetic Operations:***
  Division by Zero, Square Root of a Negative Number
- ***Illegal Type Conversion***
- ***Overflow / Underflow on Integers and Floating Point Numbers***
- ***Unreachable (dead) Code***

Costs Analysis

what it costs to find errors

Run-time Errors are costly to identify
but are easy to fix when located

- *Run-time Errors* are costly to identify and located:
  - hard to detect (ex: non-initialized variable)
  - hard to reproduce (ex: dynamic task interaction)
  - hard to trace (ex: processor halt)

About 10 hours per error (up to 100+ hours) spent under debugger to track the origin of an error

Costs Analysis

what it costs not finding an error

The later an error is discovered, the more it costs to fix.

- *Error found in Validation*
  - Development team not available

  Consequences: Deployment is delayed, budget is over
- *Error found after Deployment*
  - Products do not perform as expected (loss of service)
  - Products need to be recalled (loss of mission)
  - Products cause damage

Consequences: Impacts on corporate image and business as a whole

Existing Techniques

Techniques	Limits
- Simulation - Dynamic Testing	Very expensive - Tests cases to write - Require embedded environment models - Intrusive: source code modification - Not exhaustive: Doesn't catch all errors
- Instrumented Code - Post-mortem Analysis Tools	
- Test Coverage	
Syntactical Analyzers	- Weak support for errors detection
Quality Metrics Coding Rules Verification	
Formal Methods	- Very Intrusive, not applicable to existing projects

PolySpace Verifier

- Static, Non Intrusive: Only the source code
  - No Execution of Your Application
  - No Change to Your Development Process
- Automatic: Gives the exact location where run-time errors will occur
  - No Test Cases to Write
  - Only CPU Time, No Manpower
- Exhaustive: Identifies and checks all potential sources of run-time errors

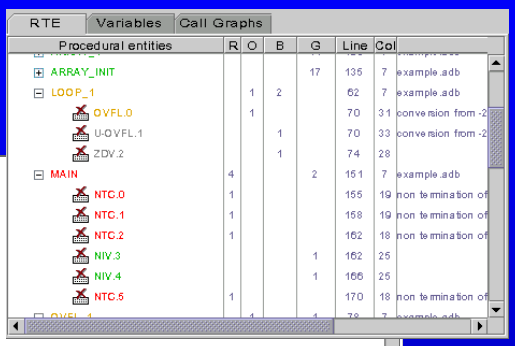
PolySpace Solution

PolySpace Viewer

```
-- arithmetic exception
procedure arith_2 is
v : long_float;
p : long_float;
y : long_float;
u : float := random.random;
begin
    arith_1 (u, v);
    p := v - 0.75;
    pragma inspection_point(v,p);
    y := sqrt ( p);
end arith_2;

-- unreachable or dead code by linear constraint
procedure unr is
   x : integer := random.random;
   y : integer := random.random;
begin
       if (x > y) then
              x := x - y;
              if (x < 0) then
                     x := x + 1;
              end if;
       end if;
end unr;

Scale  : constant Tab_Float(1..10) :=
(  0.0000000E+00 ,   2.0000000E+05 ,   4.0000000E+05 ,   6.0000000E+05 ,
   1.0000000E+06 ,   1.2000000E+06 ,   1.4000000E+06 ,   1.6000000E+06 ,
-- floating arrays
procedure array_1 is
   Tmp : Float := 0.0;
   Sum : Float := 0.0;
   K : Integer;
begin
   for I in 1 .. 9 loop
      Tmp := Tmp + (1.0 / (Scale (I+1) - Scale (I)));
      for J in 1 .. I loop
         K := I-J+1;
         pragma inspection_point (K);  -- to display variation domain
         Sum := Sum + Scale(J) - Scale (K);
      end loop;
   end loop;
end array_1;
```


PolySpace Results

RTE View

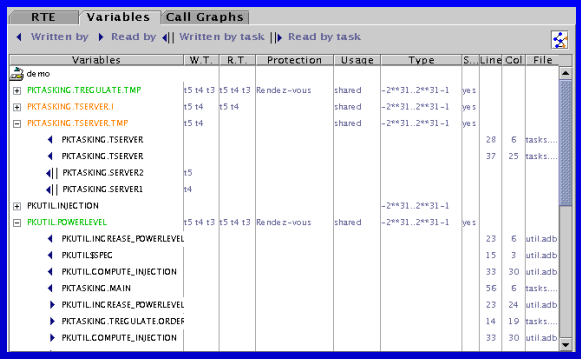


Source Code View

```
        procedure array_init is
           K : Integer := 1;
           z : float :=10.0 ;
           t1, t2, t3 : tab_int(1..10);
        begin
           for I in 1..10 loop
              t1(I) := I;
              t2(I) := I+1;
              K := K+1;
              for J in 1..I loop
                 t3(I) := t1(J) + t2(J);
              end loop;
              z := z/float(t3(i));
           end loop;
           if (random.random > 0) then
              z := z/float(t3(K));        -- out of bound array access
           end if;
        end array_init;

        -- random procedure is used to simulate complexity
        procedure mainRTE is
           v : integer;
        begin
```

Access Conflicts to Shared Data

Global Data Dictionary



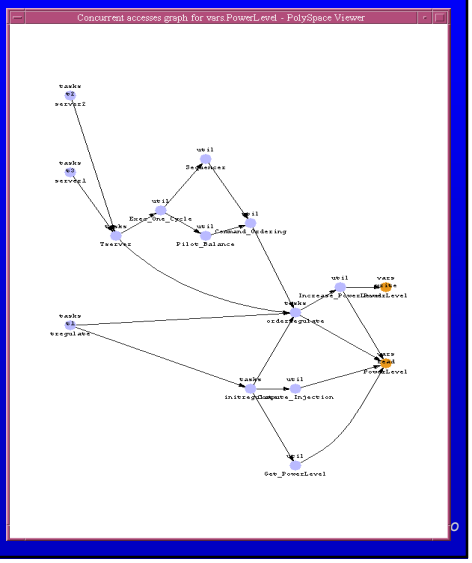
- Data description
  - Type
  - Usage Pattern (shared, modified constant, read but not written)
  - Read and Write Accesses by Tasks and Functions
  - Protection Mechanisms

Access Conflicts to Shared Data

Benefits:

- *Understand how the global data are used*
- *See possible access conflicts*
- *Design a solution easily*

One Printable Concurrent Access Graph per shared data

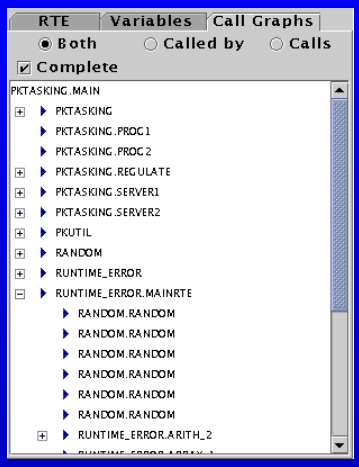


Call Tree

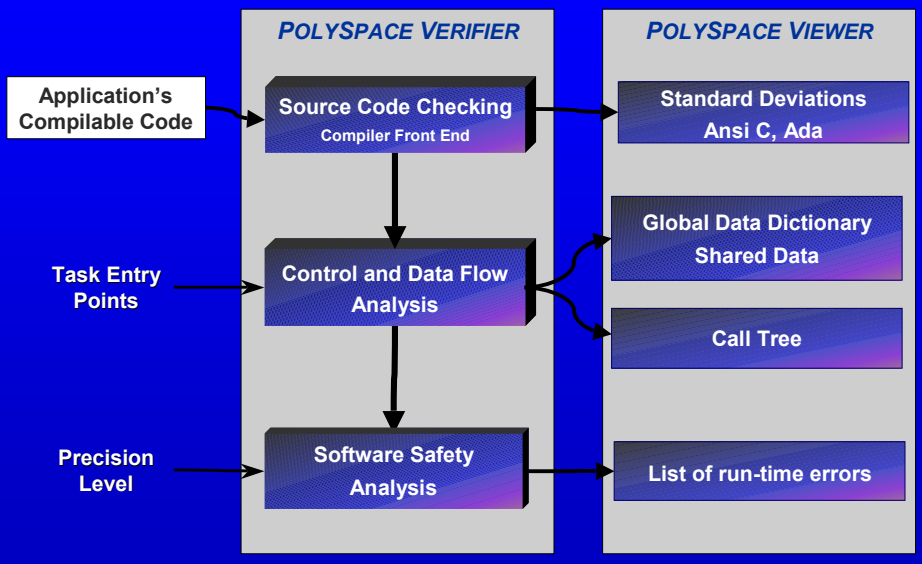
Dynamic Navigation through the Application Call Tree:

- *Called Calling Functions*
- *Propagation to the Source Code*
- *Easy check of the Call Architecture*

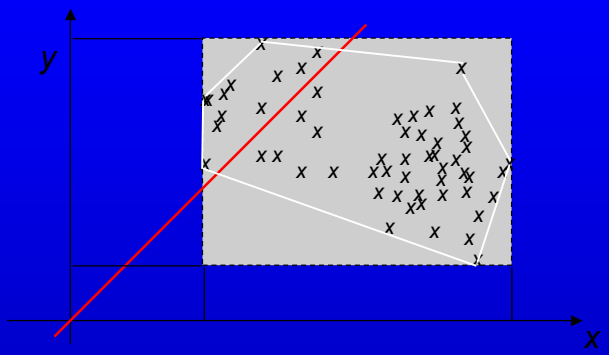
Exportable Local/Global Call Tree



Methodology



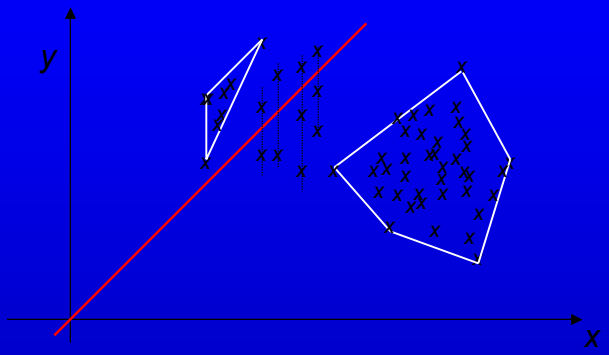
Abstract Interpretation



Expression: `a = x /(x-y);` (condition: $x \neq y$)

Low precision algorithms return a Potential Error

Abstract Interpretation



Expression: `a = x /(x-y);` (condition: $x \neq y$)

High precision algorithms return Always Correct

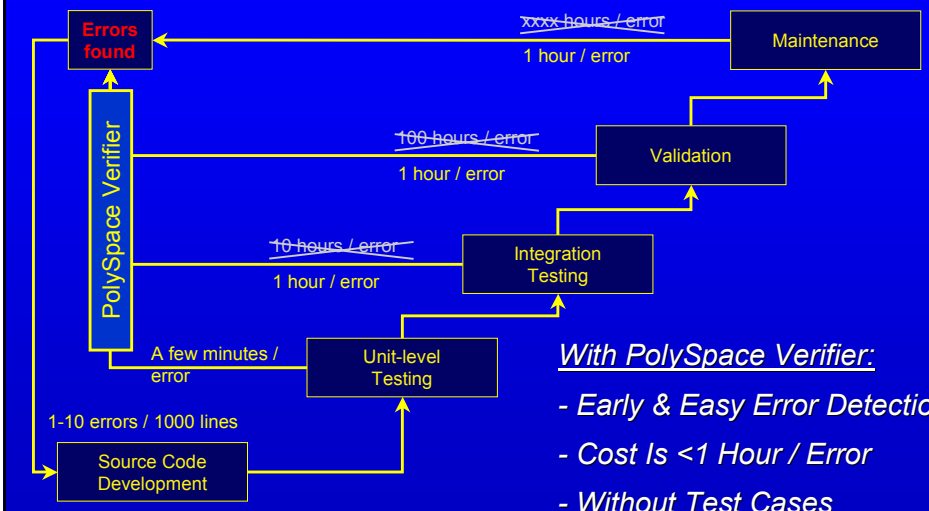
Methodology

- Clean your source code before testing !
- Module Level:
  - Find red and gray errors quickly.
  - No need for test cases or execution.
  - Find the errors in a single run

Methodology

- Clean the source code before testing !
- Integration/Validation Level:
  - Find the red and gray errors quickly
  - Depending on criticality, check some warnings using :
    - Code review (focused on relevant operations only)
    - Testing
  - 90 % to 99 % of the code is proven (green, red, gray)

Productivity Gains



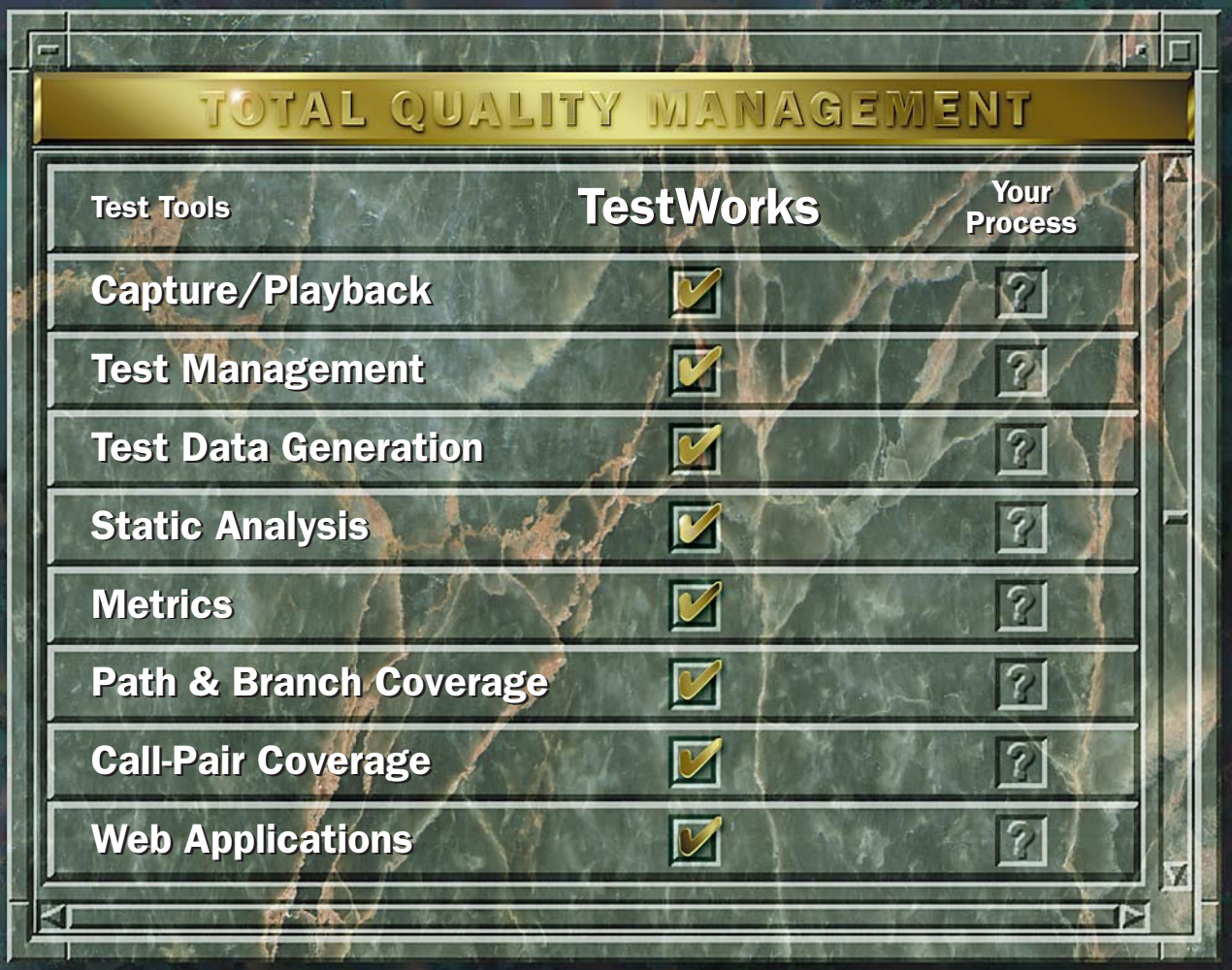
With PolySpace Verifier:

- *Early & Easy Error Detection*
- *Cost Is <1 Hour / Error*
- *Without Test Cases*

Benefits

- *Fast, Early & Effortless Identification of run-time errors*
  - *Without Test Cases*
  - *Without Execution of Applications*
  - *Without Modification to Your Development Process*
  - *Without Instrumentation of Code*
- *Reduce bug detection costs by a factor of 10!*

When PolySpace finds an error you save 10 hours.
Without PolySpace it will cost this amount.



All the tools you need for rock-solid code

Software Research has created a suite of fully integrated and automated testing tools to help you ensure the quality of your software products. TestWorks offers an end-to-end solution that covers all aspects of your process life cycle: test design and development, test data generation, test execution and evaluation, reporting and test management, code comprehension, coverage analysis, metrics and maintenance. Perfect Tools for the Quality Architect.

TestWorks improves the quality of your products, whether in an embedded or distributed client/server system, in GUI desktop or web applications. Its open architecture empowers your work across the major UNIX and all Windows platforms and OSs, in C, C++, Java, Ada, and Fortran.

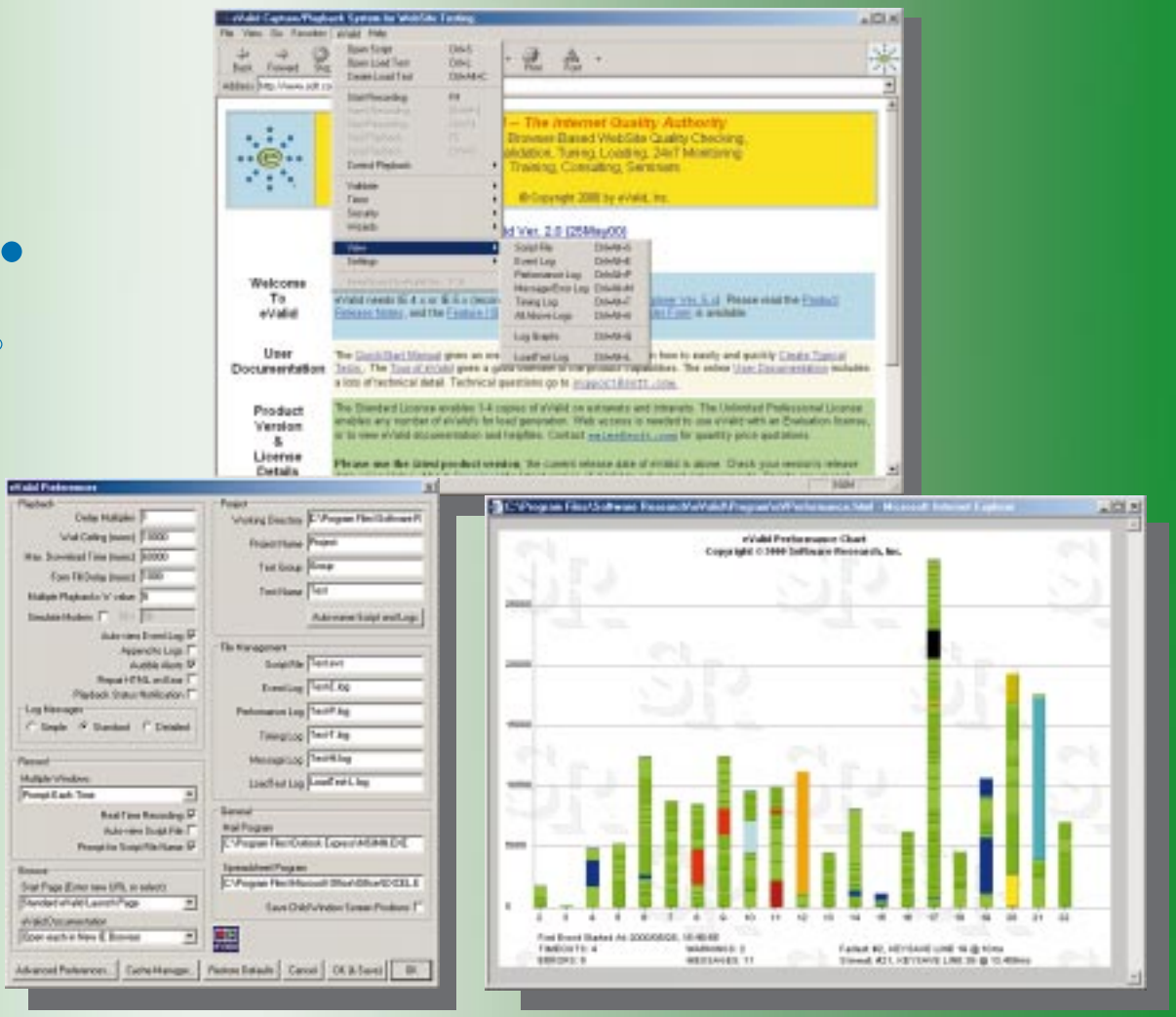
You need to deliver high-quality software on time, under budget?

We have TestWorks! Call the company that created the field .

SR Software Research

1 (800) 942-SOFT
email: info@soft.com
http://www.soft.com

Quality Testing of WebSites



- Testing
- Tuning
- Loading
- Content Validation

eValid

Your e-Business Partner

Free Download

www.e-valid.com

1-800-942-SOFT

ExtraView™

The Top Performance Tracking Engine

The Top Three Reasons People are Turning To ExtraView:

#1 Scalable, Custom Workflow Within Moments

#2 Truly Platform Independent: All Features Web-Based

#3 Simple Configuration Eliminates Programming Costs

ExtraView is a highly scalable Web and Wireless-based tracking system used to collect, route, and resolve product issues, test failures and enhancement requests. Used by quality teams, customer support, project managers, and engineers, ExtraView is easily customized for unique workflow and product improvement processes. Administrators may add unlimited user-defined fields and workflow rules with just a few clicks--no expensive programming is necessary. Automatic Email notification is triggered by important events that you deem necessary. Each Email includes a direct link to edit or view the current case. ExtraView detects the security and access level of each user and displays only the appropriate data. ExtraView supports multiple file attachments for accurate case descriptions while summary and statistical reports may be exported to HTML, text, Microsoft Excel and Microsoft Word formats.

All ExtraView features are accessible through Web browsers on all platforms. There is absolutely no dependency on legacy client/sever code. ExtraView is also available through a wireless Palm Pilot interface and through an ASP option to eliminate hardware maintenance costs.

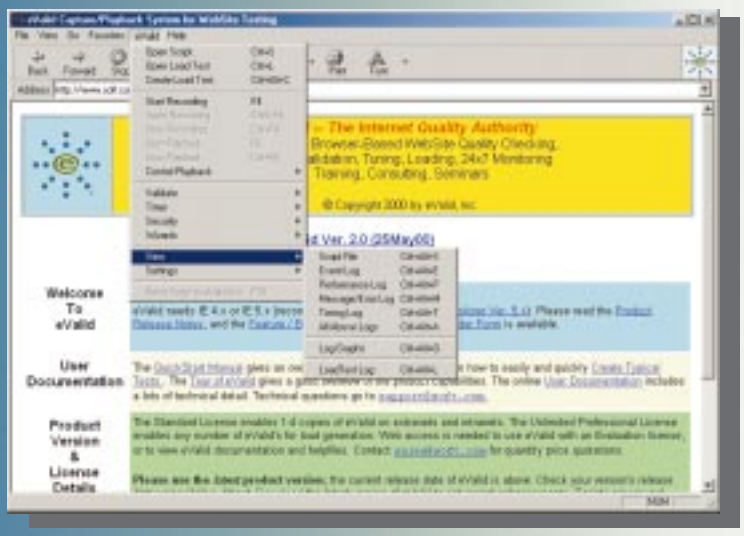
sesame technology

Visit Sesame Technology-
Booth **#109** at the Software &
Internet Quality Week 2001
or at **www.sesame.com**

WIN!

a Palm Pilot VIIx

Quality Testing of WebSites



Make Your WebSite 100% reliable – automatically

- Recording
- Playback
- Script Creation
- Content Validation
- Performance Tuning
- Load Testing

eValid

Your e-Business Partner

www.e-valid.com

- **Standard Monitoring:** eValid monitoring, based on the eValid test engine, runs standard tests on your site. eValid's 24x7 website performance monitoring provides for email and/or pager/beeper alert service, plus customer access on our WebSite to historic testing and monitoring data. *Be the first to know whenever your site is misbehaving.*
- **Custom Monitoring:** Use eValid test services to contract us to run tests you have recorded and proved out yourself using the standard eValid test engine. Custom eValid test executions run on standard intervals, in varying time zones, and are all 24x7. *Make sure your **own** tests run successfully all the time.*
- **WebSite Testing, Qualification, Verification, Loading:** eValid consulting services include WebSite testing, test suite development, WebSite qualification, e-commerce verification, and WebSite loading and capacity checking exercises. All work is based on application of the eValid test engine plus other non-released WebSite analysis facilities.
- **WebSite Quality Consulting & Seminars:** eValid website quality experts can work along side your web developers to make sure your site meets the highest reliability, quality, performance, and capacity standards. eValid seminars and workshops are aimed at bring your own team up to speed.

eValid -- Your E-Business Partner

eValid -- offering products and custom services -- is your one stop solution provider for WebSite quality. eValid is your true e-business partner.

eValid, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Phone [+1] 415.550.3020
FAX [+1] 415.550.3030
evalid@soft.com.

OCLC is a nonprofit, membership, library computer service and research organization dedicated to the public purposes of furthering access to the world's information and reducing information costs. OCLC software products and systems link more than 40,000 libraries in over 75 countries around the globe via the Internet and the World-Wide-Web.

OCLC pioneered test automation for online systems over 20 years ago and depends on automation as a strategic advantage in the control of costs and quality. OCLC's test automation tool for World-Wide-Web, WebART(tm), was introduced in the early 1990's when OCLC began deploying systems on the web.

Today, WEBART(tm)is a nuts-and-bolts test-automation tool that provides inexpensive solutions for creating, executing, and evaluating automated tests for World Wide Web, internet, and e-commerce applications. It's capabilities include automated link verification, load testing, functional testing, and regression testing. Evaluation copies are available from the WebART web site at http://www.oclc.org/webart/.

PolySpace Technologies provides the industry's first software testing products that automatically detect 100% of run-time errors in embedded applications. PolySpace's next-generation software tools have been designed to efficiently find and identify errors, enabling software developers to drastically reduce time-to-market of software applications and save significant time, money and software testing resources. By applying "abstract interpretation" techniques, PolySpace has developed extremely efficient and easy to use software error detection products that offers three major benefits:

Exhaustively: 100% run-time errors detection at compilation time.
Productivity: No test cases to write; no debugging time.
Ease of use: No code instrumentation; no change to development process.

For more information visit www.polyspace.com.

Princeton Softech

Address: 111 Campus Drive, Princeton, NJ 08540
Phone: 800.457.7060
Fax: 609.627.7799
Email: info@princetonsoftech.com
Website: princetonsoftech.com

Product/ Services Description

Princeton Softech's Relational Tools™ for DB2 and Servers help companies streamline application testing and increase productivity. Move, edit and compare referentially intact subsets of related data with 100% accuracy to improve application quality. Princeton Softech's Archive for DB2™ streamlines application databases to improve performance, availability and reduce costs.

QualityLogic Inc.

5401 Tech Circle

Moorpark, CA 93021

Phone: 800-436-6292 ext. 122

Fax: 805-531-9045

Email: info@qualitylogic.com

WWW: http://www.qualitylogic.com

QualityLogic is a full-service software Quality Assurance company with a comprehensive menu of offerings to engage with your company at any organizational level. We provide enterprise-wide consulting and quality management; we can establish a QA department or process within your company; and we can provide product-specific testing and QA services. We are flexible in our approach to your quality issues - we design a customized solution for each client's situation. QualityLogic can help you move from reacting to product defects to proactively preventing defects by design. Our services improve the efficiency and effectiveness of quality processes and the execution of quality activities. QualityLogic can reduce the opportunity costs associated with managing product quality. From product test to quality process design, we make quality happen.

Reasoning’s InstantQA is an automated software inspection service that combines the use of proprietary technology and hands-on software quality experts to help you get your software out on time, on budget, and without fatal defects. InstantQA locates five classes of crash-causing and data-corrupting defects in C and C++ code before code integration.

The InstantQA process comprises four steps, performed at Reasoning's secure inspection centers, within a five-day period. First, Reasoning conducts a thorough inventory of the code submitted, including all program and include files, to ensure full coverage. Then, our technology automatically analyzes the source code for defects representing situations that could cause a program to fail, produce incorrect or unexpected results or makes the program difficult to maintain. Next, our software quality experts manually assess the defects and remove any false positives. Finally, you receive a comprehensive report with executive and inventory summaries, metrics to assist you in benchmarking and improving your software quality, and detailed information on the location and specific problem that could be caused by each defect.

By finding defects at the source code level, InstantQA complements traditional testing technologies and practices by ensuring that code sent for testing is of the highest possible quality.

Sesame Technology presents ExtraView, a highly scalable wireless and Web-based problem tracking system used to route and resolve problem reports, defects, and product change requests. Used by customer support, quality assurance, field service, and engineering groups, ExtraView may be easily customized to individual workflow and quality processes. Geographically dispersed teams and management may access all ExtraView features such as auto-assignment of problems, dynamic email notification, and powerful query and reporting capabilities. ExtraView Remote is a wireless service automation solution that provides up-to-the-second access to ExtraView's centralized trouble ticket database. Sesame Technology is proud to be a sponsor of the 14th Annual Software Quality Week.

Software Development Technologies

125 South Market, Suite 700 San Jose, CA 95113 sdt@sdtcorp.com
Ph 408-297-1911 Fx 408-297-1993 www.sdtcorp.com

GLOBAL PRESENCE. Our mission at SDT is to assist organizations throughout the world with improving their software quality through Software Test Automation Products: UTP, an integrated test design and automation solution and ReviewPro for automated Technical Reviews and Inspections; Training and Consulting: software testing, software test automation, test planning and design; and Software Testing Outsourcing Services.

TEST DESIGN AND AUTOMATION SOLUTION. Unified TestPro™ (UTP) is a 3rd generation test automation solution to help your team create more cost-effective and maintainable test suites using fewer technical testers.

REVIEW AND INSPECTION SOLUTION. SDT offers a fully integrated, automated solution to Technical Reviews and Inspections. The SDT Technical Review and Inspection Process – TRIP™ - provides the educational and consulting services needed to develop or improve your inspection process, while ReviewPro®, SDT's web-based application, automates data and metrics collection and reporting, allowing easy communication and collaboration across business units and geographies.

TECHNOLOGY ASSESSMENTS. SDT scrutinizes your organization's current development and test practices vis-à-vis industry-acknowledged best practices. SDT recommends a step-by-step plan to most effectively address the goals and actions for optimal improvement.

TEST CURRICULUM LICENSING. SDT is continually evolving its comprehensive public and on-site training courses in the areas of Software Testing, Unit Testing, Testing for Managers, Testing Tools and Technical Review and Inspection.

OUTSOURCING TEST SERVICES. SDT provides a complete software test solution, from test design through implementation, both manual and automated, and will manage and run your test service and train your staff, if desired, in SDT's state-of-the-practice test techniques.

Sesame Technology presents ExtraView, a highly scalable wireless and Web-based problem tracking system used to route and resolve problem reports, defects, and product change requests. Used by customer support, quality assurance, field service, and engineering groups, ExtraView may be easily customized to individual workflow and quality processes. Geographically dispersed teams and management may access all ExtraView features such as auto-assignment of problems, dynamic email notification, and powerful query and reporting capabilities. ExtraView Remote is a wireless service automation solution that provides up-to-the-second access to ExtraView's centralized trouble ticket database. Sesame Technology is proud to be a sponsor of the 14th Annual Software Quality Week.

TestWorks

Software Research, Inc.'s TestWorks, an integrated suite of software test tools, is the broadest test tool suite available. TestWorks tools help automate and streamline the software development and testing process with product lines that work independently or as an integrated toolsuite. TestWorks is the only tool suite that offers Regression Testing, Test Suite Management, and Test Coverage support for Web **and** Windows **and** UNIX Platforms.

TestWorks For Windows

TestWorks for Windows, an integrated suite of automated testing tools, is the broadest suite of tools available to test applications running under MS/Windows (Win3.1), MS/Windows NT or MS/Windows '95/'98 (Win32).

Testworks for Windows has two main bundles of tools:

- TestWorks/Regression
  - CAPBAK/MSW
  - SMARTS/MSW
- TestWorks/Coverage
  - TCAT C/C++
  - TCAT for Java/Windows

TestWorks For UNIX

TestWorks for UNIX is designed to work independently or as an integrated tool suite to provide an efficient, automated testing environment for most UNIX-based platforms.

TestWorks for UNIX consists of three product lines:

- TestWorks/Regression
  - CAPBAK
  - SMARTS
  - EXDIFF
- TestWorks/Advisor
  - METRIC
  - TDGEN
  - STATIC
- TestWorks/Coverage
  - TCAT C/C++
  - TCAT for Java/UNIX
  - TCAT/S-TCAT Ada/f77

Downloading Products

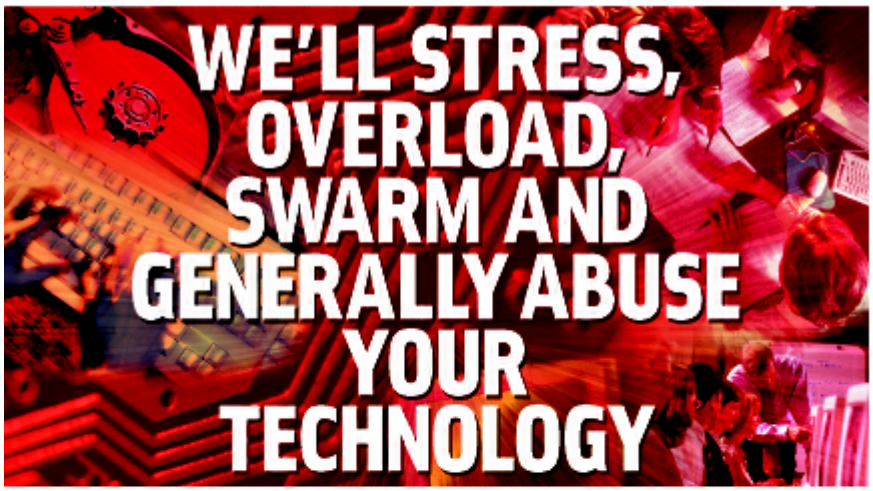
DOWNLOAD PRODUCTS **Download Datasheets**

License Key Request **QuickStart Manuals**

User Manuals

http://www.soft.com/TestWorks

Software Research, Inc., 901 Minnesota Street, San Francisco CA 94107 USA
PHONE: [+1] (415) 550-3020 FAX: [+1] (415) 550-3030
Comments & Suggestions: suggest@soft.com



[AND YOU'LL COME BACK FOR MORE]

You can drain your budget on **expensive test equipment and hard-to-find testing staff**. Or you can let the experts at eTesting Labs **give your products the workouts they need to survive** in today's highly competitive marketplace–while you focus on your core business.

Microsoft, AltaVista, Cisco, Intel, Apple, and over a hundred and fifty other **top companies have turned to eTesting Labs** for mission-critical testing, including stress/load, Web site, performance, QA, compatibility, usability, and custom tests. Find out why.

Going it alone is yesterday's solution. **Embrace the future.** Make eTesting Labs your partner for all your testing needs.

Let us attack your testing needs

Contact us now [you can thank us later]

- CALL: TOLL-FREE AT 877.619.9259
- EMAIL: ETESTING_LABS_INFO@ZIFFDAVIS.COM

TESTING LABS
www.etestinglabs.com

We test e.everything [so you don't have to]

RATIONAL SOFTWARE

Rational Software, the e-development company, helps organizations develop and deploy software for the Internet through a combination of tools, services and software engineering best practices. Rational's e-development solution helps organizations overcome the e-software paradox by accelerating time to market while improving quality. In 1999, International Data Corporation recognized Rational as the leader in multiple segments of the software development life cycle management market.
www.rational.com

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Web: www.rational.com

ZD LABS

ZD Labs (www.zdlabs.com) leads the industry in Internet and technology testing. Building on Ziff-Davis Publishing's history of leadership in product reviews and benchmark development, ZD Labs brings independent testing, research, development, and analysis directly to publications, Web sites, vendors, and large IT organizations everywhere.
www.zdlabs.com

Vanteon e-Quality Solutions

Vanteon e-Quality and Test solutions are as creative as the software you write, as innovative as the web site your business depends on.

For more than fifteen years, Vanteon has developed and refined the industry's most effective QA and testing methodologies, processes, documentation and tools to meet the business, market and quality requirements of our clients. As business and technology platforms constantly change, Vanteon continuously evolves its methodologies to keep pace with the rapid deployment of new web technologies, and hardware and software designs. At Vanteon, we not only implement QA and testing best practices for our internal product development, we create them for our clients as well.

Through its diverse experience, Vanteon has built a repository of knowledge accessible to companies looking for customized Quality Assurance and Testing solutions. Vanteon specializes in QA process consulting, test planning, test documentation development and test execution, including strategies and scripts for Automated Desktop (capture/playback) and Automated Web Testing (functional, load, stress, performance, benchmark). Compatibility testing is performed in our Configuration Test Labs. Vanteon solutions span eBusiness, wireless peripherals, desktop/consumer, networking, client/server, embedded and print/imaging. Our professionals are experts in a range of technologies, including Windows, Mac, UNIX, LINUX, Windows CE, VxWorks, QNX and PSOS.

Vanteon is the premier national provider of comprehensive engineering solutions that generate revenue for clients ranging from the Fortune 500 to hot.com start-ups. In our seven centers of engineering excellence, Vanteon has assembled one of the industry's most proficient integrated teams of engineering and consulting professionals in e-business, quality assurance, commercial software development, and hardware and embedded systems. For more information, call 1.800.266.5046 or visit www.vanteon.com.

Ziff Davis Labs (www.zdlabs.com) is the independent, for-hire testing service of Ziff Davis Publishing. At our facilities in Silicon Valley and North Carolina's Research Triangle area, we perform two distinct but related kinds of work:

- providing top-quality independent, for-hire **testing** of Internet and technology products, and
- developing and distributing for Ziff Davis Publishing its industry-standard **benchmark software**

We provide the trusted, independent testing services you need to move at top speed in the Internet economy. Whether you want us to test your Web site, your desktop or server system, or any other Internet or technology product, we have the expertise, experience, and equipment to do the job.

We live and work at the heart of the Internet economy, testing for companies large and small, established and up-and-coming. We'll put your product--and, if you wish, your competitor's--through its paces. Our experts will give you an objective assessment and warn you about any bugs or potential problems. We'll examine your product from a fresh perspective, go as deeply into the details of its operation as necessary, and put our findings in a meaningful, action-oriented context.

If you'd like to keep our findings to yourself, we'll provide strict confidentiality. If you want to publicize the results, our marketing team can help you leverage our testing and analysis efforts to your best competitive advantage. When your customers see that Ziff Davis Labs has tested your Internet or technology product, they will know they can trust the results.

COMPUWARE CORPORATION

With trailing 12-month revenues of more than $2 billion, Compuware is a world leader in the practical implementation of enterprise and e-commerce solutions. Compuware productivity solutions help 14,000 of the world's largest corporations more efficiently maintain and enhance their most critical business applications. Providing immediate and measurable return on information technology investments, Compuware products and services improve quality, lower costs and increase the speed at which systems can be developed, implemented and supported. Compuware employs more than 15,000 information technology professionals worldwide. For more information about Compuware, please contact the corporate offices at (800)521-9353. You may also visit Compuware on the World Wide Web at www.compuware.com.

COMPUWARE CORPORATION
31440 Northwestern Hwy
Farmington Hills, MI 48334

Vanteon QA Solutions
QA with a high IQ.

QA and Test solutions as creative as the software you write, and as innovative as the web site your business depends on.

QA/Test Services
- QA Consulting
- Web Performance Testing
- System (Black Box) Testing
- Unit (White Box) Testing
- OS/Browser/Platform Compatibility Testing
- Automation
- Wireless/Handheld Testing
- Web Rapid Quality Evaluation

For
- eBusiness
- Wireless / Handheld
- Desktop/Consumer
- Networking
- Client/Server
- Embedded
- Print/Imaging

Technologies
- Windows
- Mac
- UNIX
- LINUX
- Windows CE
- VxWorks
- QNX
- PSOS

Discover the significant advantages of partnering with Vanteon, who has the depth of experience, resources and collaborative spirit to work closely with you to create innovative and highly effective QA and Test solutions.

For more information about Vanteon, please visit BOOTH #413.

www.vanteon.com
equality@vanteon.com
800.266.5046

VANTEON™

101 Sun's P2P strategy, p. 10

Are patterns dead?, p. 72

Application Development Trends®

DEVELOPMENT SOLUTIONS FOR CORPORATE SOFTWARE MANAGERS

WWW.ADTMAG.COM

Our seventh annual competition, p. 40

Data warehousing lets its hair down

When it comes to new tools for data warehousing, the dynamism of Web commerce has become as much a problem as a solution. Here's how to succeed in the quest to unlock the value hidden deep inside this new clickstream of data. Page 19

APRIL 2001 VOLUME 8, NUMBER 4
A 101communications Publication

QUALITY

SD Times
SOFTWARE DEVELOPMENT
The Industry Newspaper for Software Development Managers

APRIL 15, 2001
ISSUE NO. 028

A BZ MEDIA PUBLICATION $7.95
www.sdtimes.com

EXPRESSO FRAMEWORK 3.0 CLEANS UP JAVA PACKAGES

BY DOUGLAS FINLAY

Jcorporate Ltd.'s Expresso Framework version 3.0 open-source framework for building Java-based Web applications integrates popular external programs while it reorganizes the methods by which Java packages are organized, to enable better access to the code.

Repackaging enables better access to code, according to Jcorporate's Nash.

"We began to understand through earlier versions what a good structural framework should look like, especially when bundling Java code," said Michael Nash, Jcorporate's lead developer. He said that because of the way Java bundles its code, developers were unable to optimize it for more efficient use. He said version 3.0 would enable Java developers to more efficiently utilize Java code because it would be packaged differently to ensure better accessibility by separating core classes from services and extensions.

In addition to reorganizing Java packages, version 3.0 integrates a number of external programs "to take advantage of those open-source projects that would help build the framework," Nash continued. Among the new additions are the Apa-

▸ continued on page 9

Store It On The Web, Says I-Drive

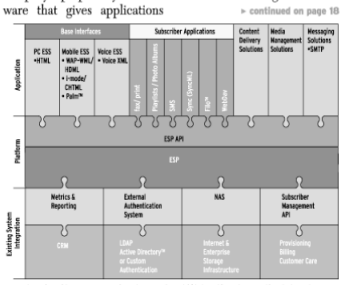
New APIs let Internet devices access remote storage services

BY EDWARD J. CORREIA

Storing data without a local hard drive can now be done with ESP. That's the claim of I-Drive.com Inc., which has released its Enhanced Storage Platform, a set of APIs for the company's proprietary middleware that gives applications running on Internet appliances, WAP phones and other resource-constrained devices the ability to store data on Web-connected storage media and exchange files peer-to-peer.

The API builds on I-Drive's Enhanced Storage Solution

▸ continued on page 10



By releasing its APIs, I-Drive hopes to shift hosting to vertical developers.

Embedded Devices Join the Neighborhood

Visuality NQ brings CIFS to Linux, VxWorks, Windows CE

BY EDWARD J. CORREIA

It's a beautiful day in the neighborhood. At least, that's a sentiment embedded developers might share when they start using Network Quick (NQ), a new file-sharing system from Visuality Systems Ltd. that lets devices running Linux, VxWorks or Windows CE be seen and configured from the Network Neighborhood or Network Places browsers on Windows computers.

Sam Widerman, Visuality's CEO, described one obvious application involving an NQ-enabled printer residing on a small office network. Users could click on the printer in their Network Neighborhood window, run a setup utility stored in the printer and install the necessary drivers onto their own local machine, gaining immediate access to the printer with virtually no involvement from IT staff.

Drivers could be stored in and shared by a network device, says Visuality's Widerman.

According to Igor Lerner, Visuality's research and development manager, NQ gives embedded devices the ability to communicate using Microsoft's Common Internet File System protocol, or CIFS, which is in use on every Windows machine and on Linux/Unix machines running Samba, an open-source CIFS stack.

Lerner suggested remote-device monitoring as another possible application. "We offer the platform for doing that. You will implement your own management software, but it will utilize the abilities of our product for the transport purposes without introducing any management protocols like SNMP, which is not easy to implement." Security is maintained through user-level permissions, VPNs and other means, he added.

After some preconfiguration, NQ installs on the client device as a CIFS server capable of sharing local files or directories specified prior to build. Depending on the processor and operating system, it occupies between 150KB and 250KB on the target device. However, an NQ device cannot read remote files, and can there-

▸ continued on page 27

MICROSOFT BREWIN' UP A (HAIL)STORM

Web services strategy based on standards

BY ALAN ZEICHICK

The hints have been available for months, leaked in speeches, white papers and press releases. But in late March, Microsoft Corp. finally unveiled key portions of its Web services strategy, based on the SOAP and XML standards, Microsoft's .NET Framework, Microsoft's Passport online user-authentication service, and .NET-enabled client devices whose browsers can send and receive SOAP messages.

Now marketed under the name HailStorm, the new strategy is designed to provide developers with a common set of tools, deployment servers and Internet-based authentica-

▸ continued on page 33

Software Business is The Magazine for Software Executives! Every issue of Software Business Magazine includes the following editorial features: . CEO Strategies . Electronic Distribution . ASP Reports . Financial Reports . Business Automation . Software Replication . Product Development . Packaging/Fulfillment . Support/Customer Service . International Opportunities . And More...!

Amphora Quality Technologies (AQT) is a Software Testing and Quality Assurance services company. AQT helps IT companies and corporations worldwide to design competitive products of high quality and reliability.

Amphora Quality Technologies was founded by a group of experienced professionals and managers seasoned within the software industry. Extensive work experience in top-level positions with major Russian software manufacturers provided AQT's top managers with wide-ranging expertise in and knowledge of Software Quality Assurance, as well as in modeling, design and development of major information systems.

Structurally, AQT consists of four divisions:

- Web Lab – a laboratory specializing in quality of Web Site and Internet applications;
- Functionality Lab – a laboratory for software functionality testing, software test results and source code analysis center;
- Performance Lab – a laboratory for the analysis of software performance characteristics and reliability;
- Research department – the company research center;

Amphora Quality Technologies a subsidiary company of Amphora LLC (Northern California, USA) is based in Moscow, Russia, and has offices in the USA.

About Atesto Technologies

Atesto is creating a new global standard for web testing. Atesto's suite of fully automated, online test services enable e-businesses to reduce costs, achieve faster time-to-market, and ensure quality in end user experience. With its web-based set of solutions, Atesto meets the testing and monitoring needs in the entire application lifecycle. Atesto works with leading companies in the area of web infrastructure services, system integrators, consulting firms, solution partners and technology providers to create a consolidated base of services for web-enabled businesses.

Currently, Atesto offers Atesto Load Test, Atesto Response Watch and Atesto Functional Test services via the company web site. Applications that live on the web should be tested on the web. For more information visit www.atesto.com or call 1-866-300-TEST.

NTS-XXCAL

NTS-XXCAL is the oldest and largest testing laboratory of it's kind, having tested Software & Hardware since 1982, Environmental testing since 1961. Lab locations are in Los Angeles, London and Japan. NTS IS AN "NRTL" CERTIFIED (NATIONALLY RECOGNIZED TEST LAB) - A US GOVERNMENT RECOGNIZED LAB. NTS-XXCAL tests Software, Hardware & Peripherals. Decrease your time-to-market! We help you validate the integrity of your product by covering your target market thus leaving your valuable development resources free to work on your new products. This makes our testing service extremely cost effective!
www.ntsxxcal.com

Butler Technology Solutions

As a division of Butler International, a recognized leader in the technology and technical services industry, Butler Technology Solutions offers a complete and broad range of information technology expertise, technology staffing, project management, and technology solutions that are delivered through Quality Assurance, Customer Relationship Management, Business Intelligence and Enterprise Applications practices. These practices are supported by our staff augmentation and project management capabilities. Unlike our competition we were the first technology services firm to achieve ISO 9000 certification, conduct annual employee and customer satisfaction surveys and report the results to the public. Butler Technology Solutions has offices in New York City, Washington, DC., Raleigh, Atlanta, Chicago and Milpitas. To learn more about Butler Technology Solutions, please visit www.butler.com.

Compuware quality assurance solutions automate the multiple, complex steps of thorough application testing and provide comprehensive, repeatable and predictable results in less time. With Compuware tools, you can test every step in the application process for mainframe, distributed and web platforms. From defining requirements, creating test scripts, executing functional, web, regression, integration and performance tests and managing defect resolution, Compuware products and services improve application quality and performance.

For example: QARun gives programmers the automation capabilities they need to create and execute test scripts, verify testsand analyze test results. QALoad performs repeatable load testing and helps determine the ultimate performance and potential limits of your system. Reconcile, Compuware's new requirements management tool, provides a method to synchronize planning, development and testing activities, giving you the ability to accurately assess the project's status. QADirector, a powerful, extensible test management solution for full lifecycle testing of distributed large-scale applications, supports a framework for managing the entire testing process-from design to execution to analysis. Compuware NuMega products help quickly detect, diagnose and repair errors; analyze performance bottlenecks; and locate untested code in applications built in C++, Java, Visual Basic, Active Server Pages, or HTML. DBPartner enables developers to optimize database performance by capturing, tuning, and debugging SQL statements and PL/SQL calls. Developers can also utilize TestPartner for unit testing and sharing test scripts with testing teams. For organizations that lack the time, infrastructure or resources to perform web site and application testing, Point Forward offers a remote web testing and monitoring solution that gives you confidence in the reliability, integrity, scalability and performance of your web site.

For more information, visit us on the web at www.compuware.com.

eTesting Labs Inc. (www.etestinglabs.com), a Ziff Davis Media company, leads the industry in Internet and technology testing. In June 2000, ZD Labs changed its name to eTesting Labs to better reflect the breadth of its testing services. Building on Ziff Davis Media's history of leadership in product reviews and benchmark development, eTesting Labs brings independent testing, research, development, and analysis directly to publications, Web sites, vendors, and IT organizations everywhere.

eValid ™ -- The Internet Quality Authority ™
Client-Side Browser-Based WebSite Quality Checking,
Testing, Validation, Tuning, Loading, 24x7 Monitoring
Training, Consulting, Seminars

About eValid -- The Internet Quality Authority

eValid enhances your e-business success by assuring that your WebSite is trouble-free, reliable, speedy, and available 24x7. In a Web-paced world your WebSite is your key asset. eValid checks, protects and insures.

eValid Products

eValid's Test Enabled Web Browser™ is a test engine that provides you with browser based 100% client side quality checking, dynamic testing, content validation, page performance tuning, and webserver loading and capacity analysis.

This new cutting-edge technology, is 100% client side based, and is completely object-oriented. eValid offers a unified approach to WebSite testing that is unique in its simplicity, power, efficiency, effectiveness, and superior ease of use.

By focusing entirely on the users' view of WebSite quality, eValid results are accurate, complete, repeatable, and highly effective -- all as experienced by your users. The eValid test engine is available in several product configurations.

- **Testing:** eValid test scripts can exercise the key parts of your site, confirm links, check content, and simulate users' activities. *Make sure your customers get the right message!* More...
- **Validation:** eValid can confirm selected content, validate document properties, images and applets. *Have confidence that you are delivering correct information!* More...
- **Tuning:** eValid timing capabilities let you identify slow-loading pages so you can "tune up" your site for optimum performance. *Keep customers from clicking away!* More...
- **Loading:** eValid load testing scenarios can simulate 100's or 1000's of users. *Can your WebSite handle the traffic when a serious crunch comes?* More...

eValid Services

eValid website quality services are all based on the eValid test engine, and are are supported through training, consulting, and technical seminars.

- **Standard Monitoring:** eValid monitoring, based on the eValid test engine, runs standard tests on your site. eValid's 24x7 website performance monitoring provides for email and/or pager/beeper alert service, plus customer access on our WebSite to historic testing and monitoring data. *Be the first to know whenever your site is misbehaving.* More...
- **Custom Monitoring:** Use eValid test services to contract us to run tests you have recorded and proved out yourself using the standard eValid test engine. Custom eValid test executions run on standard intervals, in varying time zones, and are all 24x7. *Make sure your **own** tests run successfully all the time.* More...
- **WebSite Testing, Qualification, Verification, Loading:** eValid consulting services include WebSite testing, test suite development, WebSite qualification, e-commerce verification, and WebSite loading and capacity checking exercises. All work is based on application of the eValid test engine plus other non-released WebSite analysis facilities. More...
- **WebSite Quality Consulting & Seminars:** eValid website quality experts can work along side your web developers to make sure your site meets the highest reliability, quality, performance, and capacity standards. eValid seminars and workshops are aimed at bringing your own team up to speed. More...

eValid -- Your E-Business Partner

eValid -- offering products and custom services -- is your one stop solution provider for WebSite quality. eValid is your true e-business partner. For more information, visit us on the web at **http://www.e-valid.com**.

eValid, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Phone [+1] 415.550.3020
FAX [+1] 415.550.3030
info@soft.com.

IEEE Computer Society
Booth # 305

The IEEE Computer Society strives to be the leading provider of technical information and services to the world's computing professionals. The IEEE Computer Society offers its nearly 100,000 members a comprehensive program of publications, meetings, and technical and educational activities, fostering an active exchange of information, ideas, and innovation. No other professional or commercial organization comes close to matching the Computer Society in terms of the quality, quantity, or diversity of its publications. Headquartered in Washington, DC, the society serves its members from offices in Los Alamitos, CA, Tokyo, and Brussels. The society is the largest technical society within the Institute of Electrical and Electronics Engineers (IEEE). See our CS Store at http://computer.org.

****

Marian Anderson
Advertising Coordinator
IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, California 90720-1314
USA
phone: 714-821-8380
fax: 714-821-4010

****

Company Name: LogiGear Corporation

Description:
LogiGear Corporation is a full-service software-testing firm that provides testing expertise and resources to software development organizations. LogiGear offers several value-added services including outsourced application testing, e-mobile testing, web load/performance testing, localization testing, and automated software testing for B2B, B2C, as well as for corporate internal applications.

Founded in 1994, LogiGear has built a reputation on partnering with software development organizations to help make the most of outsourcing and staff training solutions. Specializing in testing of e-applications including e-commerce sites, portals, database-access applications and web sites, LogiGear also has extensive testing expertise in e-mobile products such as wireless communication products and hand-held devices, as well as traditional end-user applications such as productivity and edutainment software titles. In addition to our outsourced testing services, LogiGear trains software-testing professionals through the offering of its Practical Software Testing Training Series. LogiGear brings its series to organizations through its OnSite Training Program, or offers them publicly in its California locations.

To complement our services and training program, we have a dedicated development team to research, design, and produce support tools and utilities to help improve testing efficiency. The first product created with this in mind is TRACKGEAR™, a Web-based defect tracking solution. Built for the Web from the ground up, TRACKGEAR is designed for everyone on your company's product development team. Project management, marketing, support, QA, testers, and developers will come to rely on TRACKGEAR as their primary communication tool. Technical groups will appreciate the power of TRACKGEAR while non-technical staff will enjoy its ease of use.

LogiGear is a privately funded corporation headquartered in Foster City, California.
www.logigear.com

McCabe & Associates, is leading provider of products and services that help IT organizations build better software, from strategic back office applications to cutting edge web technologies. The McCabe IQ products provide development organizations with effective tools for managing change, version control, quality analysis, comprehension, testing, and reengineering of business critical software applications. McCabe IQ take advantage of McCabe & Associates' innovative methodologies, visualization techniques, and metrics to help analyze, test, and reengineer software in a variety of languages on a wide array of platforms. McCabe IQ enables organizations to manage software changes, their effects on the testing and quality of applications, increase test effectiveness, and expedite and automate the testing process to promote and assure the delivery of high quality software systems.

Microsoft's vision is to empower people through great software - any time, any place and on any device. As the worldwide leader in software for personal and business computing, Microsoft strives to produce innovative products and services that meet our customers' evolving needs.

With more than 20 years of momentum behind us, a growing international presence, and one of the most talented workforces on the planet, Microsoft's position in the software industry has never been better. Opportunities for bright, talented individuals are abundant and promising.

As a tester at Microsoft you have your hands in every aspect of product development and the product life cycle from design to shipping. If you want to have an impact, if you care about easy-to-install, easy-to-use products, then testing is where you want to be. If you know you could make our products better and want to save millions of customers worldwide from the bug you just found, testing at Microsoft is the career for you.

When you get large-scale engineering projects, testing is a big part of what goes on. We have as many people who test our products inside Microsoft as we do people who build the products. We sometimes joke that we only have the people there to build the products so they can create the bugs so the testers have something to do. It's really just a big testing organization with those other guys tacked on.

Company Name: **Software SETT Corporation**
Description:

Since 1987, Software SETT Corporation has provided software quality assurance services throughout the US and abroad. Our responsive team of highly trained professionals includes SQA Engineers, Sr. SQA Engineers, Project Managers and Consultants. Software SETT also offers SQA and Project Management courses for on-site or classroom environments tailored to fit the needs, budgets and schedules of its clients.

We have an excellent reputation and take pride in repeat business and/or referrals. Our clients include startups as well as Fortune 500 firms. Our expertise includes but is not limited to projects in the following areas:

. Web-based applications and E-commerce
. Embedded Systems and Wireless Device
. Medical Device and Quality Process Audits
. Client/Server applications and Data Warehouse
. Telecommunications and DSL Technology
. Graphics, Multimedia, and Training Tools

In Fall 2000, Software SETT Corporation began hosting kSETT™, an integrated test management tool. This tool ensures that test efforts can start quickly and efficiently by offering a variety of relevant functionality and content. This cost effective and time saving tool is just another example of Software SETT's dedication to SQA preparedness.

We can be your QA outsourcing solution, perform on a project-by-project basis or provide you temporary staffing solutions to bridge your own SQA department needs. For more information about our services, please contact us at (408) 395-9376 or visit our website at www.softsett.com.

We would welcome the opportunity to be of assistance to you!

Software SETT Corporation
233 Oak Meadow Drive
Los Gatos, CA 95032
Phone: (408) 395-9376
Fax: (408) 354-6477
Email: INFO@softsett.com
Web site: www.softsett.com

Software Quality Engineering (SQE)assists software professionals and organizations throughout the world with improving their software testing and quality engineering practices. The company's hands-on experience and training expertise help companies -- large and small -- to improve testing practices, gain measurable control over software projects, and ultimately deliver better software.

Founded in 1986, SQE has grown to become one of the world's largest suppliers of testing and quality assurance training -- offering a comprehensive array of testing seminars as well as a number of development seminars. SQE's seminars cover software testing topics in the areas of software development, test management, Web development, Web testing, project management, test management and other key areas important to building and delivering quality software.

SQE organizes several international conferences, including STAREAST and STARWEST (Software Testing Analysis & Review), Software Test Automation, SM (Software Management), and ASM (Applications of Software Measurement). In addition to these resources, SQE publishes *Software Testing & Quality Engineering* magazine, a commercial publication focused on the needs of the testing and quality assurance industry, STQe-Letter - bring software testing and quality engineering info to your in-box, and StickyMinds.com - online resource for software testers, developers, managers, and quality engineers.

Founded in 1996, TeamShare, Inc. delivers Web-based collaborative software solutions. The company's product suite, powered by the TeamTrack workflow engine, enhances process management, speeds resolution, and encourages collaboration within and across enterprises. tTrack is the company's defect and development management solution. tSupport, the company's support center solution, incorporates customer participation. Together, the tightly integrated TeamTrack suite provides enterprise business process management and enables collaborative product development with business customers and partners. The products are highly configurable; their web architecture makes them simple to implement and maintain, and keeps ownership costs low.

TeamShare's customers & partners include Dell, Hewlett-Packard, Excite@Home, KPMG CitiGroup, 3-Com, and ADP. TeamShare has been named to Computerworld's 'Top 100 Emerging Companies' and the 'SoftLetter 100' lists. For more information, contact TeamShare, Inc. by phone at 888-TEAMSHARE (832-6742), via e-mail at inquiries@teamshare.com, or on the Web at http://www.teamshare.com.

TechExcel

DevTrack

The Integrated Web and Client/Server Solution for Defect and Project Tracking

DevTrack is the premiere defect- and project-tracking tool for software development teams, helping to ensure that development projects finish on time and on budget. DevTrack comprehensively tracks and manages all defects, change/feature requests, and all other development issues. DevTrack also provides powerful workflow and process automation features, robust searching and reporting, and comprehensive point-and-click customization. Intuitive and powerful, DevTrack provides a scalable out-of-the-box solution, at a great value.

DevTrack Features Include:

- Comprehensive tracking and management of all defects, feature/change requests, and other development issues, with the industry's most intuitive and powerful interface.
- DevTrack Web provides nearly 100 percent of DevTrack's powerful features, with an extremely intuitive LAN-like interface.
- Definable workflow and skills-based routing allow the defect-resolution process to be fully defined and controlled, ensuring high-quality products.
- Automatic e-mail notification for any QA or development team member when selected events occur, based on definable criteria.
- Extensive customization includes user-defined field labels, field types, drop-down menu options, master/detail relationships, and custom reports.
- The industry's best integration with Microsoft's Visual SourceSafe version control software.
- Full ODBC-compliance ensures easy scalability from one to many thousands of users.

TesCom is the world's leading software testing company offering performance, quality assurance, usability and e-business testing for systems and software. As a provider of professional services for software quality assurance, TesCom has established a position at the forefront of E-Commerce application testing, working with its clients to provide unique, cost-effective testing solutions in a customized and real-world fashion.

Since 1990, TesCom has provided a comprehensive set of testing services for its many clients around the world. TesCom has a global presence with offices in the USA, UK, Israel, France, Germany, Greece, Singapore and Australia. With over 700 testing specialists operating worldwide, TesCom has developed methodologies to provide first class solutions to companies committed to producing high quality systems within tight budgets and delivery deadlines. TesCom has extensive testing experience across all platforms, applications, networks and operating systems. The notion of risk mitigation is a key objective in all TesCom's testing engagements, since TesCom recognizes that proactive, thorough and well-planned testing can prevent highly visible system failures. For more information, please visit www.tescom-intl.com <http://www.tescom-intl.com> or via phone at +1.678.250.1100 (US).

TestQuest, Inc.
18976 Lake Drive East
Chanhassen, MN 55317
Phone: 800.756.1877
Fax: 952.936.2187
info@testquest.com <mailto:info@testquest.com>
www.testquest.com <http://www.testquest.com>

TestQuestTM Pro is the only non-intrusive automated test solution that provides comprehensive support for a wide range of electronic devices including embedded systems, computer systems, handheld devices, cell phones and Interactive TV set-top boxes. Simulating the presence of a "virtual" user, TestQuest Pro executes pre-defined streams of actions, and compares the output to valid states to determine whether the test was successful. A scripting facility provides a foundation for consistent, reliable and repeatable testing. The benefits of using TestQuest Pro include:

* Reduced test cycle time: Complete test cycles faster - customers report up to 90% time savings compared to manual methods. TestQuest Pro can run tests 24 hours a day, 7 days a week, dramatically expanding the time available for testing. This means dramatic savings in the time it takes to get products tested and into the market.

* Reduced test cost: Customers report rapid ROI and dramatic cost savings by eliminating the need to dedicate staff to testing or outsource testing to 3rd parties.

* Improved product quality: Build sophisticated test scripts that thoroughly exercise your products and reliably uncover defects. With TestQuest, repeating an advanced regression test is as easy as running a script.

VANTEON

140 Tobey Village Office Park
Pittsford, NY 14534

Phone: 800-266-5046
617-332-0202
Fax: 617-332-1121

Email: equality@vanteon.com
Web: www.vanteon.com

Vanteon Quality Assurance & Test Services

Vanteon's Quality Assurance & Test Services are as creative as the software you write, and as innovative as the web site your business depends on.

For more than fifteen years, Vanteon has developed and refined the industry's most effective QA and testing methodologies, processes, documentation and tools to meet the business, market and quality requirements of our clients. As business and technology platforms constantly change, Vanteon continuously evolves its methodologies to keep pace with the rapid deployment of new web technologies, and hardware and software designs. At Vanteon, we not only implement QA and testing best practices for our internal product development, we create them for our clients as well.

Through its diverse experience, Vanteon has built a repository of knowledge accessible to companies looking for customized Quality Assurance and Testing solutions. Vanteon specializes in QA process consulting, test planning, test documentation development and test execution, including strategies and scripts for Automated Desktop (capture/playback) and Automated Web Testing (functional, load, stress, performance, benchmark). Compatibility testing is performed in our Configuration Test Labs. Vanteon solutions span eBusiness, wireless/handheld, desktop/consumer, networking, client/server, embedded and print/imaging.

Vanteon is the premier national provider of comprehensive engineering solutions that generate revenue for clients ranging from the Fortune 500 to hot.com start-ups. In our seven centers of engineering excellence, Vanteon has assembled one of the industry's most proficient integrated teams of engineering and consulting professionals in eBusiness, Wireless, quality assurance, commercial software development, and hardware and embedded systems. For more information, call 1.800.266.5046 or visit www.vanteon.com.

VeriTest is the premium provider of testing services that enable technology companies to release *proven* enterprise-scale applications on a worldwide basis. With datacenter-equipped labs in North America and Europe, VeriTest delivers test consulting, test plan development, and test execution services through cost-effective, global processes.

Our experience, partnerships, technology and scalable resources take the uncertainty out of product quality. Since 1987, VeriTest's career test engineers and project managers have executed complex test missions under the most demanding schedules. Resources include large-scale test teams, sites on three continents, and technology alliances with industry leaders such as Microsoft, Compaq, Unisys and Rational.

With VeriTest services from Lionbridge, customers now have a single global partner for outsourcing specialized technical services associated with global product releases, including software localization, quality assurance testing, and application certification.

VeriTest services:

eTesting: VeriTest has the infrastructure to tackle the tough web and enterprise application scalability projects.

Globalization Testing and Release Engineering: VeriTest can ensure that your localized products are ready for global deployment.

System Testing : VeriTest provides complete integration, functionality, compatibility and migration testing of IT products.

Certification Testing: Leading technology companies such as Microsoft, Novell, and IBM have entrusted VeriTest with the design and implementation of standards-based testing.