International Conference on Software Quality 2007
Tutorials

Monday, October 15, 2007

Preliminary program is subject to change

7:00 a.m. - 8:00 a.m. Continental Breakfast

Full Day Tutorials: 8:00 a.m. - noon

01: 7 Low Overhead Software Process Improvements

Morning Tutorials: 8:00 a.m. - noon

02: Understanding and Applying Agile Values and Principles

03: Beginner's Toolkit

04: Integrating Value - Added Audits and Collaborative Assessments for Software Process Improvements

05: Delivering Flawless Tested Software Each Agile Iteration

Noon - 1:00 p.m. Lunch (available only to full day tutorial attendees and those attendees taking both a morning and afternoon tutorial.

Afternoon Tutorials: 1:00 p.m. - 5:00 p.m.

06: Extreme Programming: Best Practices, Tradeoffs, and Variants

07: How to Define Practical Software Metrics

08: Calculating CMMI[SM]-Based ROI: How, What, When and Why?

09: Test Design Techniques

Software Division Meeting
Monday, October 15, 2007
5:30 p.m. – 6:30 p.m.

Software Division members and others interested in the Software Division are invited to attend!!!

Come learn about the Software Division and meet our division officers, committee chairs, and regional councilors.

Tutorial #01: Full Day Tutorial

7 Low Overhead Software Process Improvements - Robin Goldsmith

Summary: Software process improvement doesn't have to be synonymous with expensive formalized approaches, such as the Software Engineering Institute's Capability Maturity Model (SEI CMM). This interactive seminar workshop describes it along with a number of alternative approaches that can provide significant software productivity and quality improvements without extensive bureaucracy or organization-wide cultural change. Exercises enhance learning by allowing participants to practice applying practical techniques to realistic examples.

Abstract: To many people, software process improvement (SPI) is considered to be an expensive formalized high-overhead, paperwork- and busywork-intensive activity. Many assume SPI is synonymous with highly-publicized commercialized approaches, such as the Software Engineering Institute's various Capability Maturity Models (SEI CMM and CMMI), Six Sigma, or Information Infrastructure Technology Library (ITIL). Because of the high costs and lengthy start-up times before results begin to occur, and because up to two-thirds of such large efforts fail to produce desired improvements, many organizations either don't undertake or abandon such high-overhead initiatives. Consequently, they may be making de facto conclusions that software process improvement is effectively impossible or impractical; and therefore they believe they must remain stuck with whatever their current software process is (not) able to produce.

In fact, a number of alternative low-overhead techniques can achieve meaningful software process improvement faster, cheaper, and more reliably. This interactive seminar workshop describes seven alternative approaches that can provide significant software productivity and quality improvements without extensive bureaucracy or organization-wide cultural change. These methods fall into two categories.

(1) Recognizing that processes and process improvement existed long before any of the formalized branded approaches, we describe several traditional proven generic process improvement concepts that can be applied to one's own specific software process. Keys are recognizing the important distinction between one's REAL and Presumed Processes and realizing that the REAL process includes beliefs and customs as well as step-by-step procedures. Common difficulties implementing these approaches are identified so they can be avoided/overcome.

(2) The most common approach that managers use to improve results is imposing “good practices” that are not specifically geared to addressing one’s own process weaknesses but are presumed likely to produce better results than the practices currently used by one’s software organization. Such trial-and-error “fads du jour” are familiar to most software professionals, as is their common failure and replacement with the next grand buzzword, which by the way are commonly unrecognized elements of one’s REAL software process. Instead, we’ll explain several well-thought-out truly “good practices” that understandably and reliably can lead to markedly faster, cheaper, and better software for man/most organizations. Such practices include discovering REAL, business requirements and Proactive Testing™.

Exercises enhance learning by allowing participants to practice applying practical techniques to realistic examples.

Objectives: When people leave this presentation they will be able to:

- Recognizing real processes and distinguishing them from presumed processes.
- Avoiding common traps that lead to making only illusory improvements.
- Ways to analyze and measure processes to identify meaningful improvements.
- Placing externally-defined formalized process improvement approaches in one’s own context.
- How hiring, training, rewarding, and management style are part of the real process too.
- Powerful “good practices” that quickly and economically improve software process results.

Bio: Robin F. Goldsmith, JD is President of Go Pro Management, Inc. consultancy, which he co-founded in 1982. He works directly with and trains business and systems professionals in improving software development/acquisition processes through more effective requirements definition, Proactive Testing™, managing outsourcing, project management, and measurement. He was International Vice President of the Association for Systems Management and Executive Editor of the Journal of Systems Management. A frequent speaker at leading conferences, he is the author of the recent Artech House book, Discovering REAL Business Requirements for Software Project Success.

Outline:

I. “REAL” VS. “PRESUMED” PROCESSES
   A. Your process, issues, improvement efforts
   B. What a process is and why we care
   C. Relation between process and results
   D. The only way to change your results
   E. Why most process improvements fail
   F. Distinguishing “real” from “presumed”
   G. When the real process is not recognized
   H. Defined and documented processes
   I. Silos, stovepipes, and smokestacks
   J. Measuring a process to its full end result
   K. Real vs. presumed testing process
   L. Overcoming unwise conventional wisdom

II. HIGH-OVERHEAD APPROACHES
   A. Process capability
   B. Capability Maturity Models
   C. Benefits and advantages
   D. Stepwise vs. continuous models
   E. Proliferations, e.g., Testing Maturity
   F. Activity vs. results
   G. Empirical analysis of actual improvement
   H. Piece of paper mentality
   I. Process imposition vs. process improvement
   J. Role of management styles and practices
   K. Six Sigma as applied to software
   L. Strengths and weaknesses
   M. Identifying appropriate measures
   N. Religious approaches vs. real improvement
   O. Management gaps

III. IF YOU DON’T KNOW WHAT YOU’RE DOING, YOU DON’T KNOW WHAT YOU’RE DOING
   A. Evolutionary vs. revolutionary improvement
   B. Key perspective to identify the real process
   C. How to measure a process
   D. Multivariate process mapping
   E. Analyzing handoffs and bottlenecks
   F. Evaluating value added
   G. Streamlining and eliminating error sources
   H. Measurement and metrics
   I. Avoiding main causes of resistance
   J. Implementing effective metrics baselines
   K. Fallacies of on-time and in-budget
   L. Non-operational “soft” components
   M. The three envelopes

N. Emperor's new clothes risk
O. Measuring and improving people processes
P. "They won't let us use this training"

IV. POWERFUL IMPROVEMENT PRACTICES
A. Making good practices work
B. Stop turning inadequacies into virtues
C. REAL, business requirements
D. System/software requirements
E. Use cases
F. Problem Pyramid™ tool
G. Effective reuse
H. Reviews and inspections
I. Meaningful early involvement
J. Proactive Testing™
K. Letting testing drive development
L. Scaling risk

Back to top

Tutorial #02: 1/2 Day a.m. Tutorial

Understanding and Applying Agile Values and Principles - Scott P. Duncan

Summary: This tutorial will present the Agile values and principles then, in workshop fashion, participants will discuss and develop ways to apply the values and principles to their software process and practices.

Abstract: While many organizations seek to reduce software development costs and cycle time as well as increase the quality of their software, there exists some suspicion regarding the claims of agile methods for doing so. Detractors have characterized agile methods as rejecting process, refusing to document, failing to plan, and, in general, encouraging a "hacker" attitude toward development. Scott believes that nothing could be further from the truth as a hallmark of agile methods, as Kent Beck has put it, is to encourage "responsible development." This tutorial presents the agile values and principles as described in the Agile Manifesto, explaining their intentions and actual values regarding processes and tools, documentation, customer relationships, and planning. It also presents the twelve agile principles associated with the Manifesto's value statements, describing the agile community's views on communication and feedback, change, work environment, technology, lifecycle models, teamwork, and other ideas important in achieving software development agility. Doing this, attendees through small group, workshop interactions then consider what sort of specific practices they see as supported by agile values and principles when applied to project organization and management, requirements development, software design, testing, implementation and product delivery. The work of the individual groups is combined to see how the agile concepts can be applied to many development organizations. For many, bringing in an entirely new development approach such as eXtreme Programming or Scrum may simply not be possible. This tutorial results in ideas for how agile values and principles can be used more incrementally to make changes to existing development processes and methods. Attendees can then pick and choose those ideas which they believe could reasonably be tried in their development environments to increase agility without upsetting their entire development apple cart.

Objectives: When people leave this presentation they will be able to: Apply the values and principles to their own development processes to make their development environment more agile.

Bio: Scott Duncan has been involved in all facets of internal and commercial software development with business and government organizations since 1972. Since 1992 he has been an internal/external consultant helping software organizations achieve international standard registration and various national software quality capability assessment goals. He is a member of the IEEE-CS, Standards Chair for ASQ's Software Division, and the Division's representative to the US SC7 TAG and to the IEEE S2ESC Executive Committee and Management Board. He is also Working Group Chair of IEEE 90003 (adoption) and of IEEE 1648 on agile methods.

Outline:

I. Agile Methods
A. Related Methods/Techniques
1 What a Methodology Should Be
B. Methodology "Triangles"
1 Agile Manifesto
C. Individuals & Interactions
1 People & Teams
D. Working Software
1 But What About "Quality"
E. Customer Collaboration
1 Customer Satisfaction
F. Responding to Change
1 Enabling Change
2 Planning

II. Principles Behind the Manifesto
III. Agile May Not Be For You If...
IV. Another Way to Think About This
A. Values & (Condensed) Principles
B. Remember Two Things

Back to top

Tutorial #03: 1/2 Day a.m. Tutorial

Beginner's Toolkit - Taz Daughtrey

Summary: Many individuals attend this conference as their introduction to the software quality profession. This tutorial is designed to introduce "new professionals" to the resources available in the rest of the conference … and beyond… by providing a framework for considering the software quality discipline. It will feature personal contact with Software Division volunteer leadership and an overview of the wide range of resources available.

Abstract: This session will offer an orientation to the content and expectations of software quality professionals. Its framework will be the Body of Knowledge established for the ASQ Certified Software Quality Engineer. As available, key Software Division leaders and program personnel will meet with those new to the field. Samples will be provided of SOFTWARE QUALITY PROFESSIONAL journal and Quality Press books. Details will be provided of training courses, local and regional meetings and conferences, and online resources. Each individual will develop a short-range plan (for this conference) and a long-range plan for afterwards. If possible, a mentor or more-experienced contact person will be identified for each participant.

Objectives: When people leave this presentation they will be able to:

- Take fuller advantage of conference activities
- Relate to fellow professionals who at different stages in their own careers
- Be aware of the full range of developmental resources and activities available
- Plan realistically for their own personal growth

Bio: Taz Daughtrey serves as Executive Director of the World Congress for Software Quality. He has taught in the Computer Science Department at James Madison University for the past six years, after a lengthy industry career. A Fellow of the American Society for Quality, Taz was the Founding Editor of the Society's peer-reviewed journal *Software Quality Professional* and has helped edit two volumes of *Fundamental Concepts for the Software Quality Engineer* for Quality Press. He has taught and consulted on a wide range of software assurance and management topics throughout North America, Europe, and Japan.

Outline:

I. Introduction to Profession: Body of Knowledge + Code of Conduct
II. Resources at this Conference
III. Resources Beyond this Conference:
IV. publications (journal + magazines + books), training, meetings, conferences, online, networking
V. Planning for Professional Development

Back to top

Tutorial #04: 1/2 Day a.m. Tutorial

Integrating Value-Added Audits and Collaborative Assessments for Software Process Improvements - Bill Deibler

Summary: While internal and external assessments, called by a variety of names (appraisals, assessments, profiles, evaluations, audits) are elements of many standards and software quality models (such as ISO 9000-3, CMM[SM], or CMMI[SM]), they are often viewed as unpleasant have-to's. In fact, when properly implemented, they can be a valuable tool for software process improvement.

A collaborative based assessment is an approach that integrates auditing with process improvement and employee communications strategies. In a value-added structure, auditors are selected for their software engineering expertise and trained to go beyond compliance. An intended side-effect of this peer-based collaborative approach is the advancement of best practices across organizational and project boundaries. This concept is also embraced in the CMMI's SCAMPI appraisal method.

Value-added auditors seek objective evidence to trigger change that can prevent problems from occurring and that can improve process effectiveness and efficiency. They represent management's commitment to quality.

Audience: This tutorial is for anyone who is implementing or managing an assessment program to support models and standards such as SEI's CMM/CMMI, ISO 9001/90003, and ISO 15504 (SPICE) or any other software process improvement effort.

This is a unique, interactive tutorial, with frequent opportunities for questions and discussion.

You will have frequent opportunities to ask questions and to identify and resolve implementation issues associated with appraisals in software development environments.

Background: While internal and external assessments, called by a variety of names (appraisals, assessments, profiles, evaluations, audits) are elements of many standards and software quality models (such as ISO 9000-3, CMM, or CMMI), they are often viewed as unpleasant have-to's. In fact, when properly implemented, they can be a valuable tool for software process improvement.

Assessors verify that processes are followed; they examine plans, reports, and other documents and observe work activity to uncover discrepancies and issue nonconformance reports. The effectiveness of processes is audited by reviewing down-stream defect reports and customer complaints to ensure that they fall within specified parameters.

A collaborative based assessment is an approach that integrates auditing with process improvement and employee communications strategies. In a value-added structure, auditors are selected for their software engineering expertise and trained to go beyond compliance. An intended side-effect of this peer-based collaborative approach is the advancement of best practices across organizational and project boundaries. This concept is also embraced in the CMMI's SCAMPI appraisal method.

Value-added auditors seek objective evidence to trigger change that can prevent problems from occurring and that can improve process effectiveness and efficiency. They represent management's commitment to quality.

Value-added auditing focuses on strategies and techniques that meet the requirements of compliance auditing and goes further to support continuous improvement.

This tutorial focuses on avoiding the problems and exploiting the opportunities associated with new and existing audit programs:

- Gaining acceptance in established organizations and in organizations growing from start-up to established
- Determining where compliance ends and suggestions begin
- Overcoming organizational barriers to process improvement

Objectives: The tutorial focuses on tools and techniques to:

- Secure the maximum business benefit for your organization from the investment in audits
- Support organizational objectives for quality and growth
- Establish the value of audits in an environment that values independence and creativity
- Ensure that quality remains or becomes everyone's job - not the responsibility of the quality department
- Gain willing organization-wide support for audits
- Move beyond compliance audits in support of continuous improvement
- Improve the effectiveness and efficiency of existing audit programs

Bio: William (Bill) J. Deibler II has an MSc. in Computer Science and over 25 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, hardware and systems engineering environments. Bill is an SEI Authorized CBA IPI Lead Assessor and SCAMPI Lead Appraiser for CMMI. He has led over 20 SCAMPI CMMI assessments.

Bill is a founding partner of SSQC. Since 1990, SSQC has specialized in supporting organizations in the definition and implementation of Software, Hardware, and Systems Engineering Practices, Software Quality Assurance and Testing, Business Process Reengineering, ISO 9000 Registration and CMM/CMMI implementation.

Outline:

I. Introduction to Value-Added Audits
   A. Increasing demands for audits, assessments, and compliance
   B. Creating a win-win environment
   C. Myths and misunderstandings about audits

II. Audit Terms of Reference
   A. Compliance audits
   B. Value-added audits
   C. Process based versus spot audits
   D. Risk Assessments

III. Audit Process Overview
   A. Audit requirements from the CMMI, ISO 9001/90003, TickIT
   B. The audit process: manage, prepare, audit, report, follow-up
   C. Making audits work
      1 Organization-wide roles and responsibilities - ensuring collaboration and cooperation
      2 Audits as an agent for cultural change

IV. The Auditor's Mission

V. The Auditor's Role
   A. Prepare
      1 Audit definition
      2 Documentation review
      3 Scheduling and notification - minimizing disruption
      4 Checklist, flowchart, and matrix - allaying fear
   B. Audit
      1 The opening meeting - setting the tone
      2 Conducting interviews
      3 Asking questions
      4 Reducing anxiety
      5 Dealing with problem interviews
      6 Taking notes
   C. Draft report
      1 A standard format
      2 Presenting a true and fair view
      3 Root cause analysis
      4 Strengths and weakness

5 The position statement
6 Major and minor findings

D. After the audit
1 Reviewing Corrective Action Plans - spotting deficiencies
2 Follow-up and closure
3 Management reporting

Back to top

Tutorial #05: 1/2 Day a.m. Tutorial

Delivering Flawless Tested Software Each Agile Iteration - Alex Pukinskis

Summary: This tutorial introduces core principles and practices for agile acceptance testing. We'll talk about automating acceptance tests and also how an effective agile team nails down requirements without detailed written specs. Exercises and real world examples will ensure participants can go back to their teams and move closer to having a "system that always runs" in each short iteration of their agile project.

Abstract: Is your team exploring Agile development, but still struggling with bugs and poor customer satisfaction? Many teams leap feet-first into agile development only to discover that it's not always easy to write good software without detailed requirements. Sometimes it seems like testers are running an iteration behind the rest of the team and are constantly playing catch-up. Even if your Test-Driven Development process is leading to great code quality, you may still be struggling with tracking down the details behind your customer's needs.

This tutorial introduces the agile acceptance testing process. Automating acceptance testing is a big part of the picture, but even more important is agreeing as a team on key attributes of the system being built. This tutorial heavily emphasizes building consensus so that by the end of the Iteration Planning Meeting, the whole team is on the same page about each feature.

We'll discuss what makes a good user story, how and when to elaborate your stories with detailed acceptance criteria, and how to use Fit/FitNesse to turn those criteria into automated acceptance tests. We'll work through exercises and specific examples, so that participants leave knowing what they need to get started with agile acceptance testing.

This tutorial has been delivered twice – at the Agile 2006 Conference in Minneapolis, and at SQuAD 2007 conference in Minneapolis

Objectives: When people leave this presentation they will be able to:

- Understand the core concepts of Agile testing and how it differs from traditional testing
- Understand the difference between acceptance tests and developer tests
- Understand how stories, acceptance criteria, and automated tests work together as a replacement for traditional requirements
- See the key problems with their current approaches to requirements definition and acceptance.
- Understand the role FitNesse can play as part of a comprehensive testing strategy.

Bio: Alex Pukinskis has helped thirty software teams improve quality and reduce time-to-market using Agile principles and practices. As an agile coach with Rally Software and ThoughtWorks, Alex helped guide small and large teams in ISVs and enterprise IT organizations through the transition from traditional software development to lean and agile. With a background including both development and management, Alex has deep experience with entire software lifecycle. Alex holds a B.A. from the University of Connecticut. He currently works with Rally as a Product Manager.

Outline:

I. Agile Promises Around Quality and Testing
II. When Agile Iterations Go Wrong
III. Core Principles of Agile Testing
IV. Acceptance Testing vs. Developer Test-Driven Development
V. Incrementally Improving Agile Testing
VI. Good User Stories
VII. From Stories to Criteria
VIII. Exercise: Writing Acceptance Criteria (30 minutes)
IX. From Criteria to Tests
X. A Working Example in FitNesse
XI. Q&A (depending on number of participants)

Back to top

Tutorial #06: 1/2 Day p.m. Tutorial

Extreme Programming: Best Practices, Tradeoffs, and Variants - Mark C. Paulk

Summary: Extreme Programming (XP) is the best known and most widely used of the agile methods that are at the forefront of software methodology discussions today. One objective of this tutorial is to discuss the antecedents of the XP variants and the alternatives that have been developed in other contexts. The tutorial will discuss both possible tailorings and reasons an XP practice might be either poorly implemented or not implemented at all.

Abstract: Extreme Programming (XP) is the best known and most widely used of the agile methods that are at the forefront of software methodology discussions today. The practices that comprise XP have been clearly stated for some years, and a variety of case studies have been published discussing the implementation and tailoring of those practices. The XP practices have

been practiced in some form for decades – the "extreme" modifier is Beck's indication that XP is a set of good practices carried to an extreme implementation. One objective of this tutorial is to discuss the antecedents of the XP variants and the alternatives that have been developed in other contexts. Pair programming, for example, can be considered an extreme implementation of peer reviews. Walkthroughs and inspections are variant forms of peer review. When implementing XP, most organizations tailor the XP practices. In some cases, alternatives have been adopted; in some cases, an XP practice has been abandoned. The tutorial will discuss both possible tailorings and reasons an XP practice might be either poorly implemented or not implemented at all

Objectives: When people leave this presentation they will be able to:

- Discuss XP practices in the context of more traditional software engineering methods.
- Discuss the tradeoffs to consider in adopting and tailoring XP.

Bio: Mark is a Senior Systems Scientist at the IT Services Qualification Center at Carnegie Mellon University, where he works on best practices for IT-enabled services. From 1987 to 2002, Mark was with the Software Engineering Institute at Carnegie Mellon, where he led the work on the Capability Maturity Model for Software. Mark's research interests revolve around high maturity practices, statistical process control, and agile methods. Mark received his PhD in industrial engineering from the University of Pittsburgh. He is a Senior Member of the IEEE, a Senior Member of the ASQ, and an ASQ Certified Software Quality Engineer.

Outline:

I. Introduction
II. Overview of XP Practices
III. Management Practices in XP
IV. Design Practices in XP
V. Code and Test Practices in XP
VI. Relationships and Dependencies
VII. Tradeoffs and Tailorings
VIII. Concluding Thoughts

Back to top

Tutorial #07: 1/2 Day p.m. Tutorial

How to Define Practical Software Metrics - Tim Olson

Summary: Most organizations struggle with metrics. Some metrics are easy to collect but are not very useful. Other metrics are too expensive to collect. Some organizations collect too many metrics, and then don't use them effectively. What is a good metric? What are the vital few metrics? This tutorial will describe "what is a good metric", and provide a baseline of the vital few software engineering metrics. An ASQ award winning Measurement Framework will also be described. The Measurement Framework is based upon the popular Goal/Question/Metric (G/Q/M) paradigm, the Juran Quality Trilogy, and the initial core measures recommended by the Software Engineering Institute (SEI). The G/Q/M Paradigm is applied to the goals of planning, control, and improvement and based on powerful metrics that have a proven track record. In order to illustrate the power of the Measurement Framework, real examples from industry are used. Finally, the Measurement Framework helps to ensure that all metrics are collected (e.g., on a form) and stored (e.g., in a database). There will be hands on exercises, as well as time for questions and answers.

Objectives: When people leave this presentation they will be able to:

- Discuss common problems with metrics.
- Present measurement principles, "what is a good metric", and the vital few metrics (e.g., metric dashboard).
- Describe an ASQ award winning Measurement Framework.
- Provide real examples and success stories from industry.
- Describe some lessons learned of "what works" and "what doesn't work" when using metrics.
- Provide time for questions and answers.

Bio: Timothy G. Olson is President of Quality Improvement Consultants, Inc (QIC). While performing quality consulting, Tim Olson has helped organizations measurably improve quality and productivity, save millions of dollars in costs of poor quality, and has helped numerous organizations reach higher SEI maturity levels. Tim Olson has been formally trained in Crosby, Deming, Juran, ISO, CMM®, and CMMI℠ quality approaches. He is also a Juran Institute Associate. Tim Olson was a lead-author of a Software Quality Course for the University of Minnesota, and he is currently a senior member of ASQ and a member of IEEE and NDIA.

Outline:

I. Introduction
II. Exercise 1
III. Measurement Problems Overview
IV. Measurement Overview (What is a "Good Metric")
V. Break
VI. Exercise 2
VII. Process Measurement Framework Overview
VIII. Real Examples
IX. Some Lessons Learned
X. Questions and Answers

Back to top

Tutorial #08: 1/2 Day p.m. Tutorial

Calculating CMMI[SM]-Based ROI: How, What, When and Why? - Rolf W. Reitzig

Summary: Demonstrating the value of CMMI-based improvement efforts is recognized as being a difficult proposition, particularly for less mature organizations with little or no measurement capabilities in place. With resources in scarce supply, CMMI efforts increasingly must be able to show superior value compared to other programs in the organization's portfolio. In particular, Return on Investment (ROI) must be demonstrated to the organization's leadership, many of whom don't know (or need to know) many details about the model or its implementation. They simply care that the effort brings financial value.

This workshop will provide guidance to attendees on the following topics:

- How? How do organizations calculate ROI? How can they do it better?
- What? What information is needed in order to calculate ROI? What information is inappropriate?
- When? When should these ROI calculations be performed? When can you compare ROI results across organizations?
- Why? Why should an organization attempt to calculate the ROI of CMMI-based efforts? Is ROI the right way to show financial value?

Objectives: When people leave this presentation they will be able to understand:

- Software and systems engineering economics
- Costs of Poor Quality
- How poor quality costs affect productivity
- How productivity gains can be realized and monetized
- Typical costs & benefits for CMMI efforts
- The various equations that encompass "Return on Investment"

Bio: Rolf W. Reitzig is the president and founder of cognence, inc. Mr. Reitzig has more than 18 years experience across multiple software engineering disciplines and has helped dozens of Fortune 500 companies improve quality, productivity and project results. cognence is a Software Engineering Institute (SEI) CMMI partner and Mr. Reitzig is a Resident Affiliate assisting the SEI in communicating the return on investment of CMMI efforts to the software development community. Mr. Reitzig speaks regularly at international conferences, seminars and user groups sponsored by the SEI as well as software engineering automation tool providers like IBM Rational and Serena. He also coaches senior managers, helping them understand the economics of process improvement and its implications on organizational change. Mr. Reitzig holds a Bachelor's degree in Computer Science and an MBA in Finance from the University of Colorado.

Outline: Introduction and Logistics

I. Why?
II. When?
III. What?
IV. How?
V. Wrap-Up

Back to top

Tutorial #09: 1/2 Day p.m. Tutorial

Test Design Techniques - Louise Tamres

Summary: So you've received, yet again, another requirements document that's confusing, incomplete, and ambiguous. And somehow, you have to translate this mess into software designs and test cases. It's not all that difficult if you know how to model requirements. Louise Tamres describes several test design techniques that pinpoint the key features and their intended behavior. In fact, once you've applied these techniques, test cases practically write themselves. These same techniques can also assist developers to identify the key functionality to implement. Now that you've defined numerous tests, you'll apply prioritization strategies to select a meaningful subset of tests.

Abstract: When receiving requirements, many perceive a magical step that automatically yields test cases. This could be true if requirements are complete, unambiguous, and presented in an all-encompassing format. In most cases, imperfect specifications result when they are written in the English language. Requirements Analysis identifies issues and addresses inconsistencies, and when this task is omitted, the recipient of the deficient requirements is the one who must translate the information into something useful. Test Design Techniques are one way of analyzing and modeling requirements. The resulting models provide a straightforward means by which to identifying missing information and improve comprehension.

In the principle that a picture is worth a thousand words, a good model focuses on content and helps the consumer ask the right questions. Common examples include Decision Tables, State Transition Diagrams, and Use Cases. We'll also cover how to apply Classification Trees, Pairwise Testing, General Tables, and Spreadsheets. In addition, we'll explore how the resulting models assist with estimating schedules, documenting tests, prioritizing tests, and recoding test outcome.

Applying the test design techniques might list out a large number of tests – much more than can be executed under tight schedule constraints. The tester must then reduce the number of tests by selecting a meaningful subset while still maintaining reasonable coverage. We'll discuss several methods for selecting and prioritizing such tests.

These Test Design Methods, which apply at all levels – from unit testing through system testing – can improve requirements, software designs, and test documentation. These same techniques can also assist developers in identifying key functionality to implement. Next time, why not incorporate the development of these models as part of the actual requirements definition.

Objectives: When people leave this presentation they will be able to:

- Perform requirements analysis.
- Identify ambiguities in requirements.
- Translate requirements into test cases.
- Produce models that better communicate requirements as compared to using text.
- Understand how to apply some of the more common test design techniques:
  - Decision Table
  - Use Case
  - State Transition Diagram
  - Classification Tree
  - Test Outline
  - Pairwise Testing
- Provide estimates for scheduling the testing tasks.
- Prioritize tests.

Bio: Louise Tamres has over 24 years experience in software engineering, specializing in software testing and software process improvement. Louise Tamres has established software quality initiatives at many companies, including GE Medical Systems, General Motors, Nortel, Electro Scientific Industries, as well as assisting start-up companies with their software quality needs. An enthusiastic speaker, she has taught many courses in software quality principles and methods. She is the founding member of Ann Arbor Software Quality Professionals, a group that meets regularly in southeastern Michigan. Her frequent role in mentoring fledgling testers led to the development of her book Introducing Software Testing published by Addison-Wesley. As part of the ASQ, Louise Tamres is a Certified Software Quality Engineer, a member of the editorial board for Software Quality Professional, and the new Councilor for Region 10. Ms. Tamres received her BS and MS degrees from the University of Michigan.

Outline:

I. Requirements Modeling
   A. What is test design and why do we care?
   B. Test design progression
II. Common modeling methods
   A. Decision Table
   B. Use Case
   C. State Transition Diagram
   D. Classification Tree
   E. Test Outline
   F. Pairwise Testing
III. Applying information from models
   A. Estimating schedules
   B. Documenting test cases
   C. Test documentation shortcuts
   D. Recoding test execution results
   E. Traceability matrix
IV. Test prioritization strategies
   A. Priority groupings
   B. Risk analysis
   C. Risk matrix
   D. Interviewing techniques
V. Final Thoughts
   A. Which methods to use and when
   B. GUI vs. embedded software
   C. Data driven vs. model driven tests
   D. Lessons learned

Back to top