

Develop Globally, Manage Centrally

Centralized control and comprehensive visibility enable companies to leverage the efficiencies of geographically distributed development without the traditional headaches.

Executive Summary

In today's fast-paced markets, software development projects are increasingly being outsourced to overseas programmers. This practice, known as geographically distributed development (GDD), saves money, time, and IT resources, while increasing business agility and expanding an organization's global reach to partners, talent, and markets. However, GDD comes with several drawbacks, including a lack of thorough project visibility, cultural and communication barriers, and an expensive collaboration tool infrastructure, usually in the form of a replicated server approach—which is well known for its latency problems, preventing true distributed collaboration.

Centralized GDD, where all the project assets are stored on a centralized server and changes are updated globally as they happen in near-real time, enables enterprises to leverage all the benefits of GDD while maintaining tight control over the entire development process, thus creating an environment conducive to agile business practices.

Borland's Application Lifecycle Management (ALM) tools approach GDD in a unique and highly efficient manner, enabling enterprises to manage all the program assets from a central server and push out updates to securely connected remote caches around the world. Effective collaboration is achieved through lightweight

remote caching agents that are "trickle charged" with changes from the central repository in near-real time, thus keeping the entire project up to date and free of conflicts while utilizing significantly less bandwidth than traditional methods.

The Borland® StarTeam® platform and Caliber® Analyst tools enable businesses to control their entire application lifecycle—from requirements definition to distributed development to iterative releases—seamlessly and efficiently. In this whitepaper you will learn how to:

- Eliminate update delays from your remote project contributors
- Avoid expensive duplicate hardware setups and associated admin costs
- Keep project assets secure while avoiding merge conflicts
- Develop trusted, centrally managed and reusable code components

Borland customers are realizing significant savings with StarTeam-driven GDD projects, to the tune of 60-80 percent over competing platforms. Borland's services include migration assistance and delivery of complete ALM solutions for today's competitive enterprises.

Introduction

Competitive enterprises are no longer limited to operating in a specific city, region, or country. The global marketplace is truly global, with smart companies leveraging resources around the world to do business internationally, with vast computing and communications infrastructures linking disparate elements of the software development operation.

The days when companies 'bought and sold local' are long gone. If development talent is more cost-efficient overseas, that's where development tasks will be routed.

The same applies to customer service, manufacturing, and even basic data entry; the most competitive bid wins, regardless of physical location.

According to research firm IDC, more than 80 percent of enterprises develop software overseas. Why? There are a number of reasons:

- Utilizing geographically distributed development (GDD) can cost far less than hiring and retaining local talent.



WEB SEMINARS

by StickyMinds.com & Better Software magazine



**BETTER
SOFTWARE**

Dear *Better Software* magazine readers,

StickyMinds.com and *Better Software* magazine staff work hard to bring you the latest and most relevant information to you in a timely manner. One way of staying committed to our promise is by hosting educational, bi-weekly Web seminars on a variety of subjects. These complimentary presentations can be viewed during a live broadcast or from our on-demand archive [here](#).



FEATURED ON-DEMAND WEB SEMINAR

If security is important for your systems, you need a comprehensive validation approach that includes cyclomatic complexity and basis-path analysis to scrutinize risky code structures using data and control flows. Speakers Paco Hope and Thomas McCabe, Jr. will discuss which activities to apply first in your lifecycle to get the most bang for your buck and ways to use static- and dynamic-path analysis to improve application security.

[REGISTER HERE](#)

We hope you enjoy our Web seminar presentations and that they prove to be a great learning experience.

Sincerely,
StickyMinds.com & *Better Software* magazine Staff



WEB SEMINARS

by StickyMinds.com & Better Software magazine



**BETTER
SOFTWARE**

Dear *Better Software* magazine readers,

StickyMinds.com and *Better Software* magazine staff work hard to bring you the latest and most relevant information to you in a timely manner. One way of staying committed to our promise is by hosting educational, bi-weekly Web seminars on a variety of subjects. These complimentary presentations can be viewed during a live broadcast or from our on-demand archive [here](#).



FEATURED ON-DEMAND WEB SEMINAR

If security is important for your systems, you need a comprehensive validation approach that includes cyclomatic complexity and basis-path analysis to scrutinize risky code structures using data and control flows. Speakers Paco Hope and Thomas McCabe, Jr. will discuss which activities to apply first in your lifecycle to get the most bang for your buck and ways to use static- and dynamic-path analysis to improve application security.

[REGISTER HERE](#)

We hope you enjoy our Web seminar presentations and that they prove to be a great learning experience.

Sincerely,
StickyMinds.com & *Better Software* magazine Staff

December 2008

\$9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to **StickyMinds.com**

**PUT ON YOUR
THINKING CAP**
A fresh look at
test improvement

**2008 SALARY SURVEY
RESULTS INSIDE!**

What's A Manager TO Do?

FINDING YOUR
PLACE ON A
SELF-ORGANIZING
TEAM

**FIND THE BUG INSIDE & WIN
AN AMAZON GIFT CARD!**

Looking for a better way to manage test cases?



Learn how to write and manage test cases more effectively. Read our TestTrack TCM best practices white paper.

Download it today at www.seapine.com/bstcmbp



Seapine

TestTrack® TCM

Software for test case planning, execution, and tracking

You can't ship with confidence if you don't have the tools in place to document, repeat, and quantify your testing effort. TestTrack TCM can help you thoroughly test your applications in less time.

In TestTrack TCM you have the tool you need to write and manage thousands of test cases, select sets of tests to run against builds, and process the pass/fail results using your development workflow.

With TestTrack TCM driving your QA process, you'll know what has been tested, what hasn't, and how much effort remains to ship a quality product. Deliver with the confidence only achieved with a well-planned testing effort.

- Ensure all steps are executed, and in the same order, for more consistent testing.
- Know instantly which test cases have been executed, what your coverage is, and how much testing remains.
- Track test case execution times to calculate how much time is required to test your applications.
- Streamline the QA > Fix > Re-test cycle by pushing test failures immediately into the defect management workflow.
- Achieve complete traceability between test cases and defects with seamless TestTrack Pro integration.

Seapine ALM Solutions:



TestTrack Pro
Issue & Defect Management



TestTrack TCM
Test Case Planning & Tracking



Surround SCM
Configuration Management



QA Wizard Pro
Automated Functional Testing

 **Seapine Software**

Download and evaluate TestTrack TCM today at
www.seapine.com/bstcm08

Your global project team on the same page

Agile Project Management
Software from the Pioneers of Agile



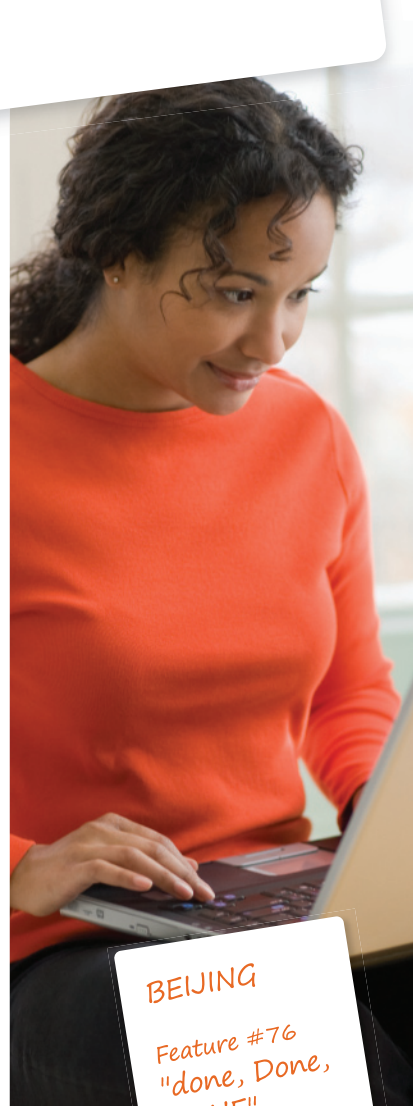
BOSTON

Card #54
"Under
Development"



BANGALORE

Bugfix #31
"In Testing"



BEIJING

Feature #76
"done, Done,
DONE"

Provide a shared work-
space for Developers, QAs,
BAs, Customers & PMs

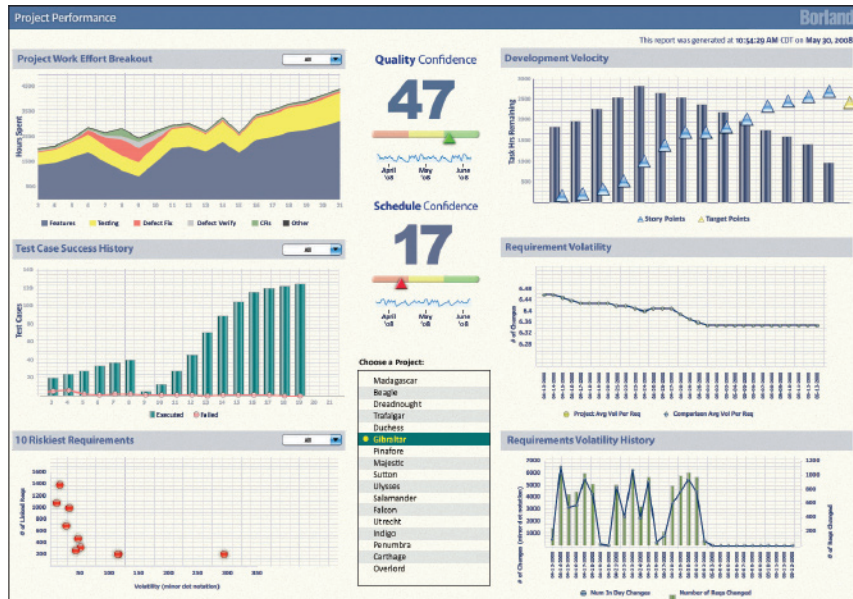
Get real-time intelligence
with burn-down charts,
velocity graphs, etc.

Capture & Visualize
all project activity

Leverage the industry's
best User Interface

Manage XP, Scrum,
Lean & Agile Hybrid
projects

Agility meets Visibility.



Solutions for the Agile Enterprise

Agile naturally brings improved communication and visibility to development teams, but as you attempt to scale in complex enterprise environments, maintaining that visibility across multiple projects, teams and geographies is challenging. During our own Agile transformation, we realized that Agile projects could be viewed from a whole new perspective.

Welcome to the Borland® Management Suite™ – delivering new possibilities for Agile teams in the enterprise.

What we learned can have a dramatic impact on the value you get from Agile:

325 developers.
5 locations worldwide.
13 Scrum teams.
12 products.
1 common vision.

Want to see what we see?
Watch Pete Morowski, Senior VP of R&D at Borland, talk about our own Agile transformation online at www.borland.com/enterprisetalk

Borland®
THE OPEN ALM COMPANY



Find the Bug
and win an **Amazon Gift Card!**
Search through the digital
edition to find the **flying bug**.
Click it to be entered for a
chance to win a **Amazon Gift Card**.

Cover Story

WHAT'S A MANAGER TO DO? 22

Self-organizing teams still need managers. But those managers need to rethink how they do their jobs and consider how much self-management the team can take on. Finding the sweet spot between hands on and hands off is the key. *by Esther Derby*

Features



Columns & Departments

In Every Issue

Mark Your Calendar **4**

Contributors **6**

eLightenment **8**

Product Announcements **44**

10 Things You Might
Not Know About ... **46**

Ad Index **48**

Better Software magazine—The print companion to StickyMinds.com brings you the hands-on, knowledge-building information you need to run smarter projects and deliver better products that win in the marketplace and positively affect the bottom line. **Subscribe today to get ten issues.**

Visit www.BetterSoftware.com
or call 800.450.7854.

SIX THINKING HATS FOR TESTERS 28

Fresh ideas can provoke great insights: Six thinking hats did just that for Julian Harty, who then applied them to software testing with great success.
by Julian Harty

THE KEY TO GOOD INTERVIEWING 34

The foundation of any successful assessment is the interview. But asking the right questions and asking those questions right makes all the difference in the quality of information you can elicit from your interviewees.
by Robert Sabourin & Lee Copeland

TECHNICALLY SPEAKING 11

Lessons Learned in Close Quarters Combat • *by Antony Marcano*

Few would think that Special Forces tactics bear any relation to software project teams. But Antony Marcano draws a surprising parallel between the dynamics of modern Special Forces “room-clearing” methods and the dynamics of modern software development teams.

CODE CRAFT 12

Train Wreck Spotting • *by Kevlin Henney*

An oft overlooked goal of encapsulation is to simplify usage. Without this sensibility, classes can end up with simplistic interfaces and callers can end up with method-call pile-ups.

TEST CONNECTION 16

A Map by Any Other Name • *by Michael Bolton*

A mapping is a relationship between two things; a map illustrates the relationship. In testing, a map might look like a road map, but it might look like a list, a chart, or a table. We can use any of these to help us think about test coverage.

MANAGEMENT CHRONICLES 18

Don't Fear the Repartee • *by Nance Goldstein*

Conflict reduces people's productivity and generosity toward the organization and their coworkers. These four steps can help defuse a conflict situation and improve the chances for a solution that, at the least, both parties can live with.

THE LAST WORD 47

The Abolition of Ignorance • *by Alan Page*

Good testers study testing to improve their knowledge of the areas they know about, but great testers strive to find out about areas of software testing they don't yet realize they don't know about.

StickyMinds.com We invite you to visit StickyMinds.com, the online companion to *Better Software* magazine. StickyMinds.com covers the same pertinent topics as the magazine, putting the power of information at the click of your mouse. Weekly columns, headline-making bugs, hundreds of technical papers, an online tools guide, discussion boards, and so much more make StickyMinds.com your site for 24/7 brainfood to help you build better software.

MARK YOUR CALENDAR

TRAINING WEEKS

www.sqetraining.com/Public

Testing

March 23–27, 2009

Boston, MA

April 20–24, 2009

San Diego, CA

Agile Software Development

March 30–April 3, 2009

Washington, DC

April 20–24, 2009

San Francisco, CA

SOFTWARE TESTING CERTIFICATION

www.sqetraining.com/certification

February 17–19, 2009

Houston, TX and Toronto, ON

February 24–26, 2009

Mountain View, CA and Atlanta, GA

March 3–5, 2009

Los Angeles Area and Tampa, FL

CONFERENCES

STAREAST 2009

Software Testing Analysis & Review

www.sqe.com/stareast

May 4–8, 2009

The Rosen Centre Hotel

Orlando, FL

Better Software Conference & EXPO 2009

www.sqe.com/bettersoftwareconf

June 8–11, 2009

The Venetian

Las Vegas, NV

BETTER SOFTWARE

Publisher

Wayne Middleton

Vice President of Publishing

Holly N. Bourquin

Editor-in-Chief

Heather Shanholtzer

Editorial

Managing Technical Editor

Lee Copeland

Technical Editors

Chuck Allison

Jonathan Kohl

Antony Marciano

Editor, StickyMinds.com

Francesca Matteu

Managing Editor, Multimedia

Joseph McAllister

Production Coordinator

Cheryl M. Burke

Design

Creative Director

Catherine J. Clinger

Advertising

Senior Advertising Sales Manager

Shae Young

Advertising Sales Manager

Joe Anderson

Production Coordinator

April Evans

Circulation and Marketing

Marketing Coordinator

Sidney White

Circulation Coordinator

Jamie Green-Gago

A PUBLICATION OF SOFTWARE QUALITY ENGINEERING



CONTACT US

Editors: editors@bettersoftware.com

Subscriber Services: info@bettersoftware.com

Phone: 904.278.0524, 888.268.8770

Fax: 904.278.4380

Address:

Better Software magazine

Software Quality Engineering, Inc.

330 Corporate Way, Suite 300

Orange Park, FL 32073



SOFTWARE TESTING

ANALYSIS & REVIEW

The Greatest Software Testing Conference on Earth

**STAR
EAST**

99.7% OF 2008 ATTENDEES RECOMMEND STAREAST TO OTHERS IN THE INDUSTRY



- 32 In-depth pre-conference tutorials
- 42 Concurrent sessions
- 5 Keynote presentations from top testing experts
- Networking opportunities to meet your peers
- Largest Testing EXPO event
- Friday Testing & Quality Leadership Summit

www.sqe.com/stareast

REGISTER EARLY AND SAVE UP TO \$300!





RICHARD BENDER has more than forty years of experience in software with a primary focus on quality assurance and testing. He has consulted internationally to large and small corporations, government agencies, and the military. He has been involved in establishing industry standards for software quality, serving as the Technical Lead for the International Y2K Test Certification Standards, and assisting the U.S. Food and Drug Administration in defining their Software Quality Guidelines. For his breakthroughs on code-based testing, he was one of the first programmers ever awarded IBM's Outstanding Invention Award. rbender@BenderRBT.com.



MICHAEL BOLTON lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries. He is co-author, with James Bach, of *Rapid Software Testing* and a regular contributor to *Better Software* magazine. Contact Michael at mb@developsense.com.



LEE COPELAND has more than thirty years of experience in the field of software development and testing. He has worked as a programmer, development director, process improvement leader, and consultant. Based on his experience, Lee has developed and taught a number of training courses focusing on software testing and development issues. Lee is the managing technical editor for *Better Software* magazine, a regular columnist for StickyMinds.com, and the author of *A Practitioner's Guide to Software Test Design*. Contact Lee at lcopeland@sqe.com.



ESTHER DERBY is a regular StickyMinds.com and *Better Software* magazine contributor and has presented at STAR-EAST and the Better Software Conference & EXPO. Recognized as one of the world's leaders in retrospective facilitation, Esther often receives requests to work with struggling teams. Esther is one of the founders of the AYE Conference. She is the co-author of *Agile Retrospectives: Making Good Teams Great*. You can read more of Esther's articles on the wonderful world of software at www.estherderby.com and on her Weblog at www.estherderby.com/weblog/blogger.html. Her email is derby@estherderby.com.



NANCE GOLDSTEIN enables individuals and organizations to deliver better results through better communication, collaboration, and conflict-resolution competencies. She's an experienced trainer, speaker, researcher, and consultant in high-tech industries and healthcare. Contact Nance at Nance.goldstein@post.harvard.edu.



JULIAN HARTY currently works at Google as a software tester on some really cool and innovative projects, which hopefully many of you will find useful and fun. With more than twenty years of experience in computing, Julian is passionate about making software work properly. He is an author and an international speaker. Contact Julian at jharty@google.com.



KEVLIN HENNEY is an independent consultant and trainer based in the UK. He provides consultancy and training in programming techniques, software architecture, and development process. He is co-author of two recent books on patterns, *A Pattern Language for Distributed Computing* and *On Patterns and Pattern Languages*.



ANTONY MARCANO has thirteen years of experience in software development and testing across numerous sectors. Since 2000, much of Antony's work has been on Extreme Programming projects. Now, as a practitioner, mentor, coach, and consultant, he helps teams realize the benefits associated with agile development. A regular speaker at peer workshops and conferences, Antony's views have been quoted in numerous publications including *Corporate Insurance & Risk* magazine and on VNUNet.com. Antony is creator and curator of [testing Reflections.com](http://testingReflections.com), one of the most influential agile testing sites on the Internet, and is a technical editor for *Better Software* magazine.



DONALD ALAN PAGE is test architect for Microsoft's Engineering Excellence Group, where he works with product teams across the company to identify and promote best practices in testing. Contact Alan at alanpa@windows.microsoft.com.



ROBERT SABOURIN, P. ENG., has more than twenty-five years of management experience leading teams of software development professionals. A well-respected member of the software engineering community, Robert has managed, trained, mentored, and coached hundreds of top professionals in the field. He frequently speaks at conferences and writes on software engineering, SQA, testing, management, and internationalization. Robert is the author of *I am a Bug!*, the popular software testing children's book; an adjunct professor of software engineering at McGill University; and the principle consultant (and president/janitor) of AmiBug.Com, Inc. Contact Robert at rsabourin@amibug.com.

EDITOR'S PICK

Abusing Quality

As this article is being finalized, I'm going back and forth between editing and keeping tabs on StickyMinds.com's latest launch. In this release, we're opening traffic to our new podcast landing page. You have to check out the new pages; we worked hard on them and feel so proud of our newest addition.

Aside from the emotions that are associated with being a proud parent (Look at my baby; isn't it so cute?), I have learned a priceless lesson. I learned to let go of imperfections. Let me explain: When designing and testing these new pages, I saw many little imperfections that I felt, if changed, would make the site sparkle like a flawless diamond. But when you're working in two- to three-week iterations, you have no time to polish to crystal clarity. We spent our gestation period making sure the pages would function well. We didn't and couldn't worry about cosmetics. Over time, we'll have many moments to perfect the podcast pages.

What matters most to us is quality. And let me be very clear about this: Quality to us means that the site functions exactly as it should. It means that every podcast is titled correctly, the link takes you to the correct page, all the pictures of the interviewees are correct and up to date, the correct MP3 queues up when you open the page, and the abstracts accurately describe the podcast you're listening to. Quality to the end-user should mean functionality, but to some, quality is that sparkling diamond. You could spend as long as you'd like polishing your software until it's "perfect." But by the time you release it, someone else might have beaten you to the market with a cubic zirconium.

Since development on our podcast pages was limited to three weeks and since I didn't want to become a bottleneck to production, I had to let go of a few "sparkly" features so we'd meet our launch date. At first it was difficult to let go of these items, but turning our definition of quality into a mantra helped tremendously.

We've all heard people use the term "quality" as some great characteristic to shoot for. There are commercials out there with the tagline, "Quality is our highest priority." But what does this mean? Bring that tagline into software development. What does it mean for testers, developers, project managers, and coders? In your daily activities, is it reasonable to strive to create the highest quality software? If you're in the medical industry developing pacemakers, quality is a huge concern. But then again, you could debate that you're striving to create 100 percent reliable software. What this means is that quality is a subjective word that, when it's not clearly defined, is easily abused.

In most cases, we don't have time to obsess over the quality of software. (We know that testing every single facet of software under development is impossible.) So when someone comes to you and says, "We've got to focus on quality," I hope you take a step back and think for a moment about what is being asked of you.

Many experts with whom I've worked creating content for StickyMinds.com are leery of the word quality. (They're the same people who also say that best practices don't exist.) One of those experts is Elisabeth Hendrickson who will admit that she hates the word, and she's got reason enough. Her argument is that this buzzword is constantly being used out of context. People use it subjectively and rhetorically. In a profession where logic reigns supreme, this word has been—and continues to be—used to elicit emotions to get people to work more on software rather than launching it once its functions are operable enough to benefit the customer. Abusing the word quality can be a serious offense, especially when it prevents you from meeting your deadlines.

Find out what Elisabeth says about this subject and what's more important than quality.
www.StickyMinds.com/editorspick10-10.



Francesca Matteu
Editor, StickyMinds.com
fmatteu@sqe.com

Quotables

"I can speak to the women's stalls, and I will say they're not any better! Plus, even if I don't have luggage, I probably at least have a purse, though there may be a hook on the back of the door if it hasn't broken off (and I refuse to place my purse on the floor). You didn't mention bulky winter coats, which add another challenge—either to remove the coat or attempt to proceed with the coat still on, and again, I don't want my coat touching that floor, and some hooks are too wimpy for some coats. When testing, we need to remember the many kinds of users with different needs and goals, and even the users we may want to be 'unfriendly' to. (How low is the wall between stalls?)"

STICKYMINDS.COM MEMBER JO WAHLE COMMENTING ON DALE PERRY'S ARTICLE, THINK LIKE A TESTER
www.stickyminds.com/quotables10-10a

"Well said, Mary, using both oars is as important as using them in appropriate, synchronized manner. The article is one of the classic approaches for development as well as testing. Thanks for such an insight in a good, logical sequence. In fact I was so engrossed while reading this article that the end was like a jerk to bring me out of it."

STICKYMINDS.COM MEMBER JAIDEEP KHANDUJA COMMENTING ON MARY GORMAN'S COLUMN BUSINESS RULES AND DATA REQUIREMENTS
www.stickyminds.com/quotables10-10b

"I expect that the reason so many of the testing professionals who read this kind of thing are the among the happiest is because they are among the most motivated in their field, so they genuinely feel they are in control of their careers, which is why they are reading these articles in the first place."

DAVID FRITZ COMMENTING ON LINDA HAYES'S COLUMN STATE OF THE INDUSTRY
www.stickyminds.com/quotables10-10c



The list below features some of the more popular columns that StickyMinds.com members have been reading lately.

11 Critical Chain Scheduling for Software Projects

By Clarke Ching

www.stickyminds.com/10-10Top1

21 The Test Manager's Vade Mecum

By Fiona Charles

www.stickyminds.com/10-10top2

31 Got You Covered

By Michael Bolton

www.stickyminds.com/10-10Top3

41 Test Managers—Start Managing!

By Dion Johnson

www.stickyminds.com/10-10Top4

51 Making Retrospective Changes Stick

By Esther Derby

www.stickyminds.com/10-10Top5

LET'S TALK ABOUT IT

Members of StickyMinds.com use the Discussion Boards to seek out answers to troubling questions, hash out problems, and tap into the minds of other Discussion Board residents. The following are real questions posted to the StickyMinds.com Discussion Boards. Got an answer? Log on to StickyMinds.com and share your ideas today.

FROM THE MANAGE PEOPLE & PROJECTS DISCUSSION THREAD:

"Does anyone have any data/benchmarks around project staffing benchmarks by discipline? I am interested in what the percentage of total resource time should be by discipline. Example: IT, 60%; testing, 15%; business analyst, 15%; project management, 10%. I'm interested in knowing where to turn for benchmarks."

Posted by: pfg

www.stickyminds.com/MPP

FROM THE TEST & EVALUATION THREAD:

"What are the standard methodologies regarding release criteria when most of the new feature tests are complete but you have a high defect count in legacy code? This is kind of tricky since all new features are functional-test complete, while in regression you start seeing more defects. I would like to understand the practices adopted during such circumstances to help make a decision either to go ahead with the release or to fix the legacy code."

Posted by: Sudhi

www.stickyminds.com/TE

FROM THE PROCESS IMPROVEMENT THREAD:

"We are in the process of attempting to better define our QA roles so that they don't blur together too much from the internal aspect, to make it easier to recruit more accurately, and to better represent ourselves to clients."

I'd like to welcome definitions and preferred qualifications/certifications for the following roles:

- Senior Test Manager / Test Director
- Test Manager
- Test Lead
- Test Coordinator
- Senior Test Analyst
- Test Analyst
- Tester

If you have good definitions for other terms, such as "Test Admin" or "Test Engineer," please feel free to define those too."

Posted by: taneate

www.stickyminds.com/PI

If you have insight or advice to share, log on to the StickyMinds.com Discussion Boards or email us at editors@bettersoftware.com. Select responses will be published in an upcoming issue of *Better Software* magazine.

Books Guide

The STICKYMINDS.COM BOOKS GUIDE is one of the most popular areas on our Web site. With more than 800 books—including many that have been reviewed by thought leaders, industry experts, and your peers—the STICKYMINDS.COM BOOKS GUIDE is your first stop for finding a good read. Not sure what you're looking for? Browse books by topic, including:

- Project & Team Management
- Test & Evaluation
- Requirements
- Design & Architecture
- Development & Deployment
- Reviews
- Process Improvement
- Measurement & Reporting
- Security
- Defect Tracking
- Configuration Management

DECEMBER WEEKLY COLUMNS

We're celebrating the holidays every Monday with the gift of knowledge by bringing you columns from respected experts in the field—Michele Sliger, Linda Hayes, Ellen Gottesdiener, and Bryan Sullivan. We're sure you'll find their expert commentary beneficial and timely. Been on vacation and missed a column? Check out the column archive any time to catch up on quality reading.

BOOK REVIEW

Successful Test Management: An Integral Approach

By: Iris Pinkster, Bob van de Burgt, Dennis Janssen, and Erik van Veenendaal

Reviewed By: Noreen Dertinger kayak27@gmail.com

Successful Test Management: An Integral Approach describes the activities of a test manager based on a test management model developed by LogicaCMG. The activities described include setting the conditions for a test project, test strategies, estimating, and planning. The target audience for this book includes test managers, test coordinators, and team leaders. The authors suggest less experienced members of the book's audience read the book end to end while more experienced readers can refer to the areas that are of most interest.

The methods described in this book apply mainly to software development. In the opening chapter, the authors describe different types of testing, namely unstructured testing, testing with design documents, requirements-based testing, and risk-based testing. The recommended approach described in this volume is risk- and requirements-based testing that focuses on the risks to the business, not just the risks to the software under development. The priorities are based on product risks. Tests that cover the highest product risk are given preferential treatment. Requirements are linked to the appropriate product risks, and the test team creates tests accordingly. In this process, a thorough risk assessment must be carried out. The identification of risks should include the following stakeholders: clients, customers, product experts, the project team, the project leader, a marketing manager, and the developers.

The remaining chapters deal with how to carry out the project in terms of feasibility, managing the test project, risk analysis, estimating the boundaries of the project (time and money for the testing), planning, the testing organization (roles and people required to carry out the testing), time management, and reporting.

Supporting appendices supplement the material in the book. These include a detailed summary of Prince2, (which is already used extensively by the UK government as well as in the private sector and internationally) and test management, a risk checklist, a template for risk- and requirement-based testing, quality attributes according to ISO9126, a template for the test plan, the goal metric question applied to testing, a checklist of project risks, and various report templates.

The authors encourage readers to substitute their own test methods into the outlined test management methodologies. This book offers valuable information to almost anyone in the testing organization. For example, this book explains how to estimate time and resource requirements. I was most interested in the section about estimating and did find a good, high-level overview of the topic in this book. The materials and approaches contained within will be of greatest interest to those working with the Logica methodologies. Since the authors have indicated that the method can be adapted to one's own situation, it may also be of value to others.

Visit www.StickyMinds.com/bythebook10-10 to post your comments on this book.

Lessons Learned in Close Quarters Combat

by Antony Marcano

I'm at the door, heart pounding so loudly I'm sure everyone can hear it. Four silhouettes are flat to the wall. I'm point man, stuck to the edge of the door we're about to breach. The team lead whispers through a covert radio mike, "Alpha 1 in position." Silence ... tense anticipation ... then, after a thirty-second eternity, my earpiece crackles as control responds, "Standby ... Standby ... Strike! Strike! Strike!"

The method of entry (MOE) man, on the opposite edge of the door frame, kicks in the door; behind me, the team lead throws in a stun grenade. BANG! As point man, I enter first—followed by number two. Five action-packed seconds and three targets later, we call out, "ROOM CLEAR!"

Instantly, we move to the next room. The MOE man is nearest to it, so he immediately becomes point man. The man on rear cover follows, taking up the MOE position becoming the MOE man. I fall into the number two position behind the point man, becoming team lead. The former team lead drops in behind me to provide rear cover. Each of us instantly switches roles, and the next door is breached without hesitation.

This might sound like a scene from an action movie, but, no, it was a close quarters combat (CQC) training session I attended with my fellow Dark Angels—a top UK Airsoft team. Airsoft is a skirmishing sport similar to paintball but with realistic, imitation firearms capable of firing 6mm plastic projectiles at rates in excess of 1,000 rounds per minute.

In this training session, we were practicing the latest CQC techniques taught to Special Forces. In CQC situations, Special Forces don't have time to shuffle around to get team members into the position that an individual's job title might dictate. Lives are at stake (or points in Airsoft). The team must be able to adapt instantly; there can be no waste in the process.

Indeed, there are specialists. For ex-

ample, our MOE specialist might tackle the especially tricky entries or advise the team on entry tactics, but we all are competent in MOE. Each of us is capable of dynamically switching roles, quickly adapting to changing circumstances. This is a perfect example of a *truly* cross-functional team of generalizing specialists [1].

This is almost the opposite of your typical software organization, in which each person has a job title that fixes his role—business analyst, developer, tester. These job titles make complete sense in phased-development approaches (e.g., waterfall) where the work is divided up as if it were a production line; business analysts pass the outcome of business analysis to developers who pass the result of development on to testers and so on. These job titles make less sense when using agile approaches that integrate these activities so tightly that they are all but inseparable.

More significantly, however, the balance of skills needed during each iteration (or timebox) fluctuates depending on the nature of the features being implemented. One change may involve more refactoring (changing internal and not external behavior) and be protected by pre-existing, automated tests. Another change may involve limited coding and much more exploratory testing. There are endless variations on those themes.

Like the Airsoft team's going from one room to the next, software teams must seamlessly go from one iteration to the next. It's simply too wasteful for progress to be halted waiting for a fixed-role specialist to finish his previous task. Everyone on the team must constantly adapt so that we continue to fulfill our shared responsibility of *frequently* delivering working software.

An increasing number of organizations seem to recognize this, creating

“It's simply too wasteful for progress to be halted waiting for a fixed-role specialist to finish his previous task.”

a demand for multi-skilled people. Interestingly, however, history seems to be repeating itself! Until the mid 1990s, high- and low-level software design was performed by systems analysts and then coded by programmers. Perhaps due to the demands of rapid application development,

the roles later combined and the multi-skilled analyst-programmer emerged. Subsequently, this became the norm and analyst-programmers were thereafter known only as “developers.”

Today the title of developer-tester (or tester-developer) is emerging in response to the flexibility demanded by agile teams—kind of an analyst-programmer-tester. As the uptake of agile methods grows, this demand is only going to rise! If history repeats itself, the developer-tester may, too, become the norm—negating the need for the “tester” suffix that differentiates them. Yes, the “software tester” job title, one of the last remaining titles derived from phased-development methods of old, could suffer the same fate as Ye Olde Systems Analyst.

This wouldn't mean that software testing as a discipline will disappear altogether—just that many of the testers and developers of today will need to leave their comfort zones to become the developers of tomorrow.

So, hold on to your job titles for dear life as the world around you evolves, or broaden your horizons and embrace the future of flexible roles. I've chosen the latter. What you do is up to you! **{end}**

REFERENCES:

[1] Ambler, Scott. “Generalizing Specialists: Improving Your IT Career Skills.” *Agile Modeling*, 2006. www.agilemodeling.com/essays/generalizingSpecialists.htm

Train Wreck Spotting

by Kevlin Henney

The story so far: In my November 2008 Code Craft column, “Encapsulation and Vampires,” I clarified some of implications of encapsulation that are typically overlooked when programmers reduce the concept of encapsulation to mean nothing more than using the `private` keyword on data. A more robust definition can be found in the *New Oxford Dictionary of English*:

encapsulate

- Enclose (something) in or as if in a capsule.
- Express the essential feature of (someone or something) succinctly.
- Enclose (a message or signal) in a set of codes that allows use by or transfer through different computer systems or networks.
- Provide an interface for (a piece of software or hardware) to allow or simplify access for the user.

The emphasis here is on enclosure, essence, and simplified usage. In code, the idea of data privacy follows from this, but it is a consequence rather than the main driver or the sole definition of encapsulation. This view of encapsulation shifts the emphasis from being supplier-centric, where the supplier is the author of the class, to consumer-centric, where the consumer is a user of the class. The good news is that, in the long run, caring for the consumer also makes life easier for the supplier.

```
class recently_used_list
{
public:
    void insert(const std::string & most_recent)
    {
        std::vector<std::string>::iterator found =
            std::find(
                elements.begin(), elements.end(),
                most_recent);
        if(found != elements.end())
            elements.erase(found);
        elements.insert(
            elements.begin(), most_recent);
    }
    ... // state-changing operations, such as clear
    const std::vector<std::string> & list() const
    {
        return elements;
    }
private:
    std::vector<std::string> elements;
};
```

Listing 1



ISTOCKPHOTO

Barely Encapsulated

The C++ code in listing 1 is where we left off last time. It shows a class that represents a recently used list—a set of strings held in order of insertion. Although the data members are declared private, the code is at best barely encapsulated (*barely*, adverb: [1] only just; [2] nakedly).

Some might consider the code in listing 1 to be a strawman and dismiss it out of hand. It represents, however, coding habits that are more common than not. This approach to class design publishes incidental assumptions, reduces options, and leads to train wrecks. While it can be tricky to recognize your own assumptions (they’re assumptions, after all), and it’s not always obvious when your options are being reduced, it *is* easy to spot train wrecks.

Chain of Irresponsibility

The *train wreck* description is sometimes applied to a method-call or scope-qualification pileup in code, where chained calls traverse an object hierarchy and scopes are drilled into. For example, if you want to query anything useful about the recently used list defined in listing 1, you have to write code such as that shown in listing 2 (train wrecks are in italics).

```

recently_used_list mru;
...
bool is_empty = mru.list().empty();
...
std::string most_recent = mru.list().front();
...
std::vector<std::string>::const_iterator position;
for(position = mru.list().begin();
    position != mru.list().end();
    ++position)
    ...

```

Listing 2

This is quite a modest example, involving only one extra carriage, but in such a simple class it's enough to illustrate the basic problem: To perform almost any query on a recently used list first requires calling the `list` function. The call chain weakens rather than strengthens the intention of the code, which is why the result is deemed a train wreck rather than an exposition of elegant and skilled object navigation. Examples of train wrecks from production code can involve enough carriages to rival transcontinental freight trains.

I have heard the style justified as being open, flexible, and easy. You don't have to anticipate all of the operations in advance, and it saves having to write a lot of forwarding operations. The only problem with this claim is that it is (1) based on vague hand waving, (2) confused in terms of its priorities, and (3) wrong.

First off, what happened to simplifying access for the user? If this style is justified as easy, for whom is it easy? Not writing forwarding functions may be a shortcut for the supplier of the

```

recently_used_list mru;
...
bool is_empty = mru.empty();
...
std::string most_recent = mru.front();
...
recently_used_list::iterator position;
for(position = mru.begin();
    position != mru.end();
    ++position)
    ...

```

Listing 3

class, but it is not convenient for its consumers. For such a simple class, we would expect (and should be entitled) to write the query code in listing 3 rather than the code in listing 2.

Publish and Be Damned

Now, let's see what happens when we make some representation changes and expose a few of the assumptions that have been published into the dependent code.

An alternative representation would be to use a `deque`. Like `vector`, a `deque` container supports random access and, there-

```

class recently_used_list
{
public:
    void insert(const std::string & most_recent)
    {
        std::deque<std::string>::iterator found =
            std::find(
                elements.begin(), elements.end(),
                most_recent);
        if(found != elements.end())
            elements.erase(found);
        elements.push_front(most_recent);
    }
    ...
    const std::deque<std::string> & list() const
    {
        return elements;
    }
private:
    std::deque<std::string> elements;
};

```

Listing 4

fore, efficient indexing. Unlike a `vector`, a `deque` also supports efficient insertion at both ends, which allows the implementation of `insert` to be written slightly more economically, as shown in listing 4 (changes from listing 1 in italics).

What are the consequences of the changes in listing 4? Many queries will be unaffected, but any usage code mentioning `vector`, such as iterator declarations or anywhere `vector` is declared for copying or passing, will no longer compile. What should have been a modest and reasonable change has turned out to be a build breaker.

```

class recently_used_list
{
public:
    void insert(const std::string & most_recent)
    {
        elements.remove(most_recent);
        elements.push_front(most_recent);
    }
    ...
    const std::list<std::string> & list() const
    {
        return elements;
    }
private:
    std::list<std::string> elements;
};

```

Listing 5

The C++ standard library offers another sequence container that is also a reasonable candidate for representation: `list`. The code in listing 5 (changes from listing 1 in italics) illus-

trates some of the brevity possible using a `list`.

Better, right? Not if you're the consumer. In addition to all the breakages we would have had switching to `deque`, there's a whole lot more. A `list` is a linked list and does not support constant-time, random-access operations such as subscripting. Both `vector` and `deque` support operator `[]`, but `list` does not. Had indexing been encapsulated by providing the appropriate operation as a member function on a `recently_used_list`, the issue would never have arisen.

Not only is it reasonable to switch between `vector`, `deque`, and `list`, but it also should be reasonable to migrate this particular class to Embedded C++ [1]. Although `string` is supported under EC++, `vector` is not, so we would have to employ ordinary arrays. This would break almost every single line of code written against the interface style used so far. As I've already mentioned, encapsulation is more than just using `private` on your data declarations.

The Semantic Spider Web

All this compilation breakage is enough to make you head straight back to `vector`, which serves to highlight the flaw in the justification of this style: There is no flexibility because the code is now locked in. The supplier cannot change the underlying type for fear of breaking all the consumer code and the consumer is stuck with clunky code. As Joshua Bloch has noted, "Public APIs, like diamonds, are forever." [2]

But the reasonable options you might choose to exercise as an implementer are not yet exhausted. Although `push_front` is a constant-time operation supported by both `deque` and

more as listing 6 (changes from listing 1 in italics).

Superficially this is a very modest change. But appearances can be deceptive. Although the data type used has not been changed, the way it is used has been. The code compiles as before, but the semantics are different: The `list` is now back to front. Once more this will break consumer code—but now at run time instead of compile time. If you have unit tests run as part of the build, you'll catch this. If you don't, you may either be convinced to start writing unit tests or vow only ever to write new code and never to touch existing code again. Both options have career implications, albeit quite different ones.

Managing the Supply Chain

Although there are other improvements that could be applied to the code, such as introducing `typedefs` to avoid long type names and extracting a private function to perform the element removal, these are left as an exercise for the reader.

The conclusions drawn concerning train wrecks support what is known as the Law of Demeter [3]. It is not really a law in the physical or legal sense, but a recommendation: Talk to yourself (`this`), talk to your associates (passed arguments and data members), and talk to your offspring (any objects created in the current scope), but don't talk to strangers. As the metaphor of consumer and supplier has been used in this column, there's another way of looking at this: Don't expose your whole supply chain.

In the previous column it became clear why the example class should support its own modifier operations rather than invite other code in to mess with its private parts. The considerations in this column reinforce that conclusion and further demonstrate why—to be considered reasonably encapsulated—the class must also provide its own query operations rather than rely on public handouts of private data. **{end}**

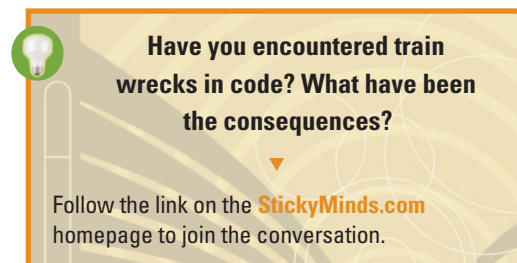
REFERENCES:

- [1] Embedded C++: www.caravan.net/ec2plus/
- [2] Joshua Bloch, "Bumper Sticker API Design": www.infoq.com/articles/API-Design-Joshua-Bloch
- [3] Law of Demeter: www.ccs.neu.edu/home/lieber/LoD.html

```
class recently_used_list
{
public:
    void insert(const std::string & most_recent)
    {
        std::vector<std::string>::iterator found =
            std::find(
                elements.begin(), elements.end(),
                most_recent);
        if(found != elements.end())
            elements.erase(found);
        elements.push_back(most_recent);
    }
    ...
    const std::vector<std::string> & list() const
    {
        return elements;
    }
private:
    std::vector<std::string> elements;
};
```

Listing 6

`list`, it is not supported by `vector`. What would be a convenient and equivalently efficient operation on a `vector`? Try `push_back`, which we can now use to rewrite listing 1 once



BETTER SOFTWARE

CONFERENCE & EXPO

JUNE 8-11, 2009
LAS VEGAS, NEVADA
The Venetian

KEYNOTES • TUTORIALS • WORKSHOPS • CLASSES

Agile Development ↑

Project Management ↑

People & Teams ↑

Testing & QA ↑

Requirements ↑

Process & Metrics ↑

Design & Architecture ↑

*Over 98% of 2008
Attendees Recommend Better
Software Conference & EXPO
to Others in the Industry*



www.sqe.com/bsce

A Map by Any Other Name

by Michael Bolton

Through most of the '90s, I worked for Quarterdeck, a company that made memory management software for PCs. Memory management tools were important in those days because programs had been developed for the MS-DOS operating system, which was in turn developed for a processor that provided only one megabyte of address space. When more powerful processors appeared on the scene, they provided access to vastly more memory, but DOS programs couldn't get at extra memory without some fairly sophisticated trickery. One of the approaches was based on *mapping*—making physical memory from far outside DOS's address space appear in a window that was inside DOS's address space.

For a long time, I found the concept difficult to understand. I knew something about mapping in cartography, and I knew that a mathematical function is sometimes referred to as a mapping, but I'd never heard the word used to describe making something appear somewhere else. Things got easier to understand when I considered mapping more generally and realized that maps are links to an idea and a representation—literally, a “re-presentation” of the idea. So a mapping, in a general sense, can take the form of charts, graphs, drawings, or diagrams but might also appear as tables or lists.

When we talk about test coverage, we might be talking about covering a map that represents the product or some aspect of it—a functional diagram or a process workflow. But we might also decide to cover a map—or more accurately a mapping—that presents testing ideas in a non-graphical form. Here are a few examples:

Use a requirements document as your map. Many test groups translate requirements documents from one form (“The input field shall accept up to twenty characters”) into another (“Verify that the input field accepts up to twenty



ISTOCKPHOTO

characters”). This is not only a waste of time but also, potentially, a directive toward weak testing. Instead of rewriting the document, try testing directly from it. Identify and prioritize statements in the document, using them to trigger test ideas. Checkmark and annotate requirements statements as they're tested, or describe tests in some kind of summary form, pointing to data tables or output files rather than reproducing them. Use the annotated document to guide reviews of testing sessions between a tester and a test manager or project lead. Combine the discussion and the annotated requirements document to check whether test coverage is satisfactory.

Requirements documents and requirements tools are intended to capture and specify someone's intentions for some aspect of the product. These specifications often focus on functional attributes and sometimes pay less attention to the parafunctional (some say non-functional) aspects. While it's likely that much has been learned or changed since the requirements were first identified, in my experience it's somewhat less likely that the document or tool has been updated to reflect the new information. Requirements-based test planning may guide the tester toward a heavily

confirmatory approach, rather than a more investigative one. So don't let your test coverage stop here; diversify and use other approaches, too.

Create a map directly from the user interface. While interacting with the product, develop a mind map or a list in hierarchical outline form of the menu and submenu options, dialogs, wizards, buttons, context menus, and other interface options that the product appears to provide. Annotate or detail your map with descriptions of how you tested each item. This approach details coverage for the options that are apparently available to the end-user, but it may be weak in terms of low-level functionality, data integrity, long-term reliability, and so on. It also fails to account for functions that may be available or necessary but not immediately visible. Diversifying your coverage ideas will mitigate the risk of missing something important in the UI. When I miss something using this approach, I ask myself whether I need to explore more methodically or whether features are buried where end-users might not find them, either.

Map the risks. Use review and brainstorming to identify important plausible risks in the product and optionally list specific test ideas. Then perform tests

designed to expose the problems that you've anticipated, checking off existing risks or tests, while keeping your eyes and mind open to new risk ideas. This approach can help drive coverage toward problems that matter. Our own hypotheses about risk are valuable, but we're all likely to be limited and constrained by our biases, so work with other people and use bug taxonomies or cheat sheets (see below) to generate fresh ideas. Ongoing testing may suggest that we're well-defended against certain risks and highly exposed to others, so revisit, review, and revise the risk list frequently to identify what's been covered and what hasn't.

Completely cover some defined corner of the domain space. Doug Hoffman's story of the integer square root function on the MASP processor is a case in point. He considered a number of coverage models to reduce the number of tests to run, and then it occurred to him: Why not try all 4,294,967,296 possible integer inputs? Using automation, he prepared a test that covered the entire input domain for that particular function in a few minutes. This wasn't complete test coverage for the whole processor, nor even for all of the possible risks for that function (like stress or flow or performance problems), but he did cover the entire map of its input values.

Map operational models, use cases, or tasks. Use cases or business process workflows can be useful in identifying places where we need to test. Whether you're provided with a list or develop one yourself, you can devise tests to cover the list. On the other hand, it's important to question use cases. I've seen a lot that are very tidy and heavily idealized, but I've never seen one that describes how people actually work in practice. Things are rarely as messy in a use case as they are in the real world.

Use a set of heuristic guidewords or test ideas. James Bach's Heuristic Test Strategy Model, Elisabeth Hendrickson's Test Heuristics Cheat Sheet, and Michael Hunter's You Are Not Done Yet models are all excellent checklists for guiding exploration of some aspect of some model of the program. Use these or develop your own models. Vary the product elements you look at, the quality criteria

you look for, and the test techniques you perform. Plot tests against coverage ideas as you ...

Work from a test matrix. Prepare a spreadsheet. Scribble a list of test ideas down the y-axis and list aspects of some test coverage model—product elements, quality criteria, platforms, test techniques—across the x-axis. As an experiment, create multiple sheets using the same tests, but with a different coverage model on each sheet, and observe how a single test can provide coverage in a number of different dimensions. For a given coverage model, denser coverage of the matrix suggests (but does not prove) deeper coverage of the particular set of ideas on that table.

Quantitative measures of coverage can be troublesome because they are so easily subject to reification error—treating conceptual things like test cases or requirements statements as though they were units instead of containers for ideas. When we apply models, though, we begin to enter a qualitative world.

Qualitative evaluation of coverage can be troublesome, too, because quality—"value to some person," in Jerry Weinberg's definition—is subjective, indefinite, and uncountable. But test completeness is always a subjective and arbitrary concept. Any map tells you something you might want to know, but no map can tell you everything. So we need to develop diversified sets of ideas about coverage and how we map it. Then we check, explore, and compare them to confirm what we believe we know, to guide discovery of what we don't, and to help tell the story of where we've been and what we've done. {end}

**What do your maps look like?
How do you describe coverage
to clients and your project
community?**

Follow the link on the StickyMinds.com homepage to join the conversation.

RALLY SOFTWARE

The Award-Winning-Est Agile Lifecycle Management Solution

 Three-time Jolt Product Excellence award winner in 2006, 2007 & 2008 for project management tools

 Two-time SD Times 100 award winner for tools "that made December 2007 a far more productive and efficient time to code than January 2007."

 Forrester Research says—
"Rally designed the requirements management capabilities in Rally Enterprise, a software-as-a-service (SaaS) ALM solution, to suit teams using Agile processes. The product performs flawlessly in this regard..."

 Get your FREE trial of Rally Enterprise at www.rallydev.com/bsm
No download, no installation, no commitment

Don't Fear the Repartee

by Nance Goldstein

Jill, project manager for her organization's new software product, picked up her phone and nervously dialed a three-digit extension. "Rich, this is Jill. I just received the test results from your shop, and I can't believe it. You're not testing the features that we think will make this product soar in the market. You're testing old capabilities that the market has moved beyond.

"We cannot show these test results to Frank and have any hope of getting the OK to move into production any time soon. Why didn't you give me a heads up that this was happening?"

Rich, the organization's head of testing, felt himself going icy; this is a conversation he'd had with Jill before. "You let your designers go crazy," he said. "It makes our job impossible. We spent a ridiculous amount of time and resources designing new tests for all the snazzy features you put in. Your fancy designs—as you can see by the results—are highly unstable. We guarantee Frank that our testing focuses on reliability. We can't possibly guarantee this product's performance."

"The deadline for getting this into production is on top of us!" Jill said. "Maybe we'll test here." Jill worried about how much she had risked for the new product. From the outset she knew that Rich's staff didn't have the skills to understand and test the new designs. And she didn't know how to talk to these old-school types who only think about "manufacturability."

Rich flinched. "What you should do, instead, is rein in your designers! Redesign this to improve its performance." If Jill took on testing, corporate management would force Rich to lay off one-third of his department, as others had in the retrenchment.

Both Rich and Jill are feeling attacked. They know that the actions of one threaten the professional credibility and security of the other. The steps that follow this confrontational trajectory



ISTOCKPHOTO

may cost one or both of them dearly—a missed deadline, the product's cancellation, staff layoffs, or exclusion from future projects or promotions.

There's also another challenge: They each feel unable to find a satisfactory way out of this jam, which has caused a fight. Flight is the other common response to conflict. Fighting to improve our position or disengaging to protect ourselves or our staff can lead to blaming, impasse, or losing face. Either choice fuels the conflict, now or in the future, escalating animosity and reducing the chances of solving future problems between the divisions.

Conflict reduces people's productivity and generosity toward the organization and their coworkers. In a recent study by the University of North Carolina, 53 percent of surveyed workers said they lost time at work worrying about a past or future confrontation with a coworker. The Society for Human Resource Management reports that 28 percent of office workers surveyed wasted time at work avoiding confrontational colleagues. Twenty-two percent put less effort into their work because of "bad blood" among colleagues. Even small-scale conflicts can reduce task focus and lock people into rigid black-or-white positions that do not create the best deci-

sions [1].

The key is to be curious. Any satisfactory solution asks at least one party to step back from the fray and ask questions. These questions should focus on what each person really wants at that moment and in the future. That information can create new options the conflicting parties cannot see in the heat of the moment.

Four steps can defuse the situation and improve the chances for a solution that, at the least, both parties can live with.

Decode. Challenging conversations skim the surface of the real problem. To understand what's going on and to find a path to a solution, require decoding the conversation. Revisit the conversation by writing down the following: What was I thinking and feeling after each person spoke? What assumptions was I making? What did I not say? What did she not say? This can rarely happen in the heat of a conversation; it may require a request for a five-minute break and callback. Stepping back and asking questions can uncover what really happened as the words flew.

Divide. Separate the person from the issue. These conversations often leave the participants disliking one another (and maybe themselves). That easily

STORY LINES

- **Being curious can bring you to good and great solutions. Try to understand your own interests—short-term and long-term.**
- **Ask the other parties what their interests are. Listen carefully.**
- **Create ideas that give others something they really care about. Bring something they value to the table, increasing the possible gains for both of you.**

leads them to avoid each other, which may be costly. Instead of writing off the issue as a personality clash, focus on the underlying problem that generated the words and behaviors.

Determine your own interests. Figure out why this situation and the work are important to you both in the short run and the longer term. Then contact the other person and ask what her interests are. What in this situation and its outcomes is important to you? Listen carefully, writing down her aims and asking her to explain more when you don't understand. This creates a list of what matters most to both of you.

Delight. Create deliverable options that offer the other person something she values. Even in impasses where the solution seems limited to My Truth or Your Truth, there are more options. Create ideas that will give the other person something that matters now or in the future. This is not giving in. This enables everyone to see previously unimagined outcomes that may exceed anything that was possible in the original either/or choice.

How do you do this in the heat of a challenging conversation when one (or both) of you feels insulted, assaulted, embarrassed, incompetent, manipulated, or just pressed for time?

You might stop for a moment—or a day—and have a very different conversation.

“Jill,” Rich said. “It will help us both if we figure a way out together—and quickly. What is really important to you for this product? For your life in this company? What could a good solution do for you?”

“C-level would see me as a successful contributor!” Jill said.

Rich took notes. “You said you want to make a contribution,” he said. “What kind? How would you like to benefit from all this?”

“I’d like to be invited to the strategy meetings that launch the company’s new initiatives. If we miss this deadline, they’ll cancel the launch and blame the failure on me. I can kiss good projects and a promotion goodbye!”

Jill talked a bit more, then took a deep breath and asked gingerly, “What about you?”

“I really want my team members to feel confident about keeping their jobs,” Rich said. “I think it’s time we learned new test methodologies. But, first, my team members have to feel secure, engaged, and involved. Being told how good they are would be nice, too!”

By talking about their future contributions to the organization and their concerns about job security and reputation, Rich and Jill have opened up the conversation and the possibilities. They have stepped out of a framework of blaming that would fuel their anger and frustration. They have stepped out of positions and are talking with each other, not with personality “types.” They have learned about one another’s thoughts—and constraints—about the future, each helping the other understand the organization and the context for this product’s success.

Jill and Rich suddenly switch to suggesting ways they can meet each other’s interests. They create options that may not have anything to do with the technical details of the new product or the tests. Jill may invite Rich and his staff to a design meeting to explain their innovations in testing new features. This information exchange might lead to designs that anticipate the limits in testing or to the two groups working together to improve testing. Or Jill and Rich might write a memo together to the executive VP for product development to

propose a more forgiving development timeline that could create a leap forward in product quality. This gives them both “face time” with the next level, showing their leadership qualities and value to the organization. Both feel part of a solution and an appreciation of their skills and experience.

Challenging conversations that lead to everyone feeling as if they got something they really wanted does more than just make people feel better. The next conversation will now start on a more positive note, rather than opening with hostility, hurt, or desire for revenge. More than that, Rich and Jill together create value that did not previously exist. Working to “delight” one another may shorten development cycles, produce surprising product ideas, and provide ways for the staff to collaborate more effectively.

Challenging conversations happen every day. Approaching them with curiosity creates opportunities to improve individual performance and relationships and may generate new value for the enterprise. **{end}**

REFERENCES:

- [1] DeDreu, Carsten. *The Vice and Virtue of Workplace Conflict: Food for (Pessimistic) Thought*. Journal of Organizational Behavior, 2008. 29: 5-18.



How do you handle conflict in the workplace?

Follow the link on the StickyMinds.com homepage to join the conversation.

Every manager has a story to tell, and we’d like to hear yours! The editors of **Better Software** magazine are looking for software managers with a literary flair to submit content for our monthly Management Chronicles department. For more information, email a query to editors@bettersoftware.com. Subject: Management Chronicles.



Can you achieve application quality without application security?

Many companies are under the impression that testing for web application security simply involves a cursory check for easy-to-guess user names and passwords. Yet application security testing can and should involve more complex audits, such as testing for SQL injection and cross-site scripting vulnerabilities. Often this sort of review does not happen until the web application is in production, when it is too late to stop a hacker or a malicious program from attacking and much more expensive to remediate the vulnerability.

While quality assurance (QA) departments have traditionally focused on functional or performance testing—it is a clear trend that QA is becoming a critical participant in application security testing.

Are you ready for security testing?

There are three ways that your QA department may become involved with web application security testing:

- Your company's web security experts may request that application security testing be done by the QA group to ensure that all fixes have been implemented and no security holes exist prior to releasing the product to production.
- Your compliance officer—facing concerns about Sarbanes-Oxley (SOX), Health Insurance Portability and Accountability Act (HIPAA), payment card industry (PCI), etc.—may request that further application security testing is performed during the QA process.

- Your QA department may request involvement with testing for web application security, because an application with potential security holes is not going to be perceived as high-quality by users.

No matter how the department gets involved, certain steps will need to be taken to establish the application security testing process. It will need to be determined whether there will be specific, dedicated staff members who will be performing web application security testing, or whether the task will be dispersed throughout your entire QA group. In addition, the timing of web application security testing during the QA process will need to be managed. Ideally, application security testing will be performed as early as possible, so that developers can fix any security issues in a timely manner without compromising the project's schedule. Finally, the right software for application security testing will need to be selected and implemented.

The right approach to application security testing

The QA department will need application security testing software that is able to perform three different types of testing to determine the vulnerabilities inherent in each user class: as a non-authenticated user, an authenticated user, and an administrative user. Additionally, the web application security tool should be able to perform both automated and manual crawling/spidering of your web application.

Run a free test of your web applications via our free 15-day trial of HP QAInspect® software and get a comprehensive vulnerability report.
www.hp.com/go/QAInspectdnlld

Automated application security testing software will spider the entire application by clicking every button and link, filling out data fields to identify the structure of the program, and then auditing each page for vulnerabilities. It should do this from the outside in, reviewing each portion of the site the way an external hacker might. This comprehensive approach is valuable to ensure that all security holes have been identified and can be fixed. On the down side, it can also produce false positives, and it may not be able to access all of your web pages due to the way that certain pages are coded.

Manual testing allows a user to focus on specific pathways or tasks on a website while the software follows silently behind, tracking the process. The program can then audit the particular path that the user has taken for security vulnerabilities and provide a report. Manually crawling an application can be time consuming, but it also ensures that specific pages are tracked and analyzed.

Choosing the right products

The following basic questions should be addressed when you are looking for a web application security testing product:

- How easy is the product to use?
- What kind of training will your QA department require in order to properly use the product?
- How well does the product integrate into the tools and software that are already used by your organization?
- How often is the product updated with new security checks—daily, weekly, monthly?
- What is the false positive rate of the product?
While no product is perfect, you want to find a product with as a low a rate as possible so that your resources are not wasted going through false positives.


- How well does the product integrate with leading quality management platforms?
- Does the product appear to evaluate each page of your application or does it get stuck on certain pages?
- Does the product allow the end user to easily modify scan settings?
- What kinds of restrictions are in the product's license?
- In which formats are reports offered (PDF, HTML, XML)? Are they easy to read? Do they contain information on the location of the vulnerability, how to execute it, how to verify it and how to fix it?
- Will the company allow you to evaluate the product before committing to purchase it? Confidential vendors will often provide a seven- to 15-day evaluation period.

HP Software makes it easy

Leading the charge in application quality and security, HP Software has recently completed the acquisition of SPI Dynamics, the leader in web application security testing. SPI Dynamics technology, which is already seamlessly integrated with HP Quality Center software, enables organizations to assess security vulnerabilities along the entire lifecycle of web applications—including development, QA and operations. Customers can also use SPI Dynamics software to validate application security and quality to meet auditing and compliance requirements, such as SOX.

To find out more about HP Software's integrated solutions for Application Quality and Security, please visit www.hp.com/go/software





What's A Manager TO Do?

FINDING YOUR
PLACE ON A
SELF-ORGANIZING
TEAM

by Esther Derby



Recently I attended a talk during which a leading agile consultant discussed implementing agile software development. The speaker waxed enthusiastically about self-organizing teams that would manage their own work and make most of the decisions about building software. The person sitting next to me squirmed uncomfortably. After a few minutes, he leaned over and whispered to me, “I’m a manager. With all this talk about self-organizing teams, what’s left for me to do?”

Moving toward self-organizing teams—whether driven by adopting agile methods or not—doesn’t mean that all the managers are out the door. There’s still plenty of management work to do; it’s just different work. And, for many managers, it is also more satisfying work.

In this article, I describe how self-organizing teams are different from manager-led teams and lay out how that changes the manager’s role.

All well-functioning teams serve their clients, meeting or exceeding expectations. But cross-functional, self-organizing teams can enhance both creativity and commitment. An effective self-organizing team increases the capability of the individuals and the team as a performing unit. Managers of self-organizing teams don’t track day-to-day progress or set individual goals. They work to remove obstacles that prevent the team from achieving success. They work to optimize the team’s ability to achieve the organization’s goals.

Most of us have an image in our minds of what “team” means. That image affects how we think about teams,

how we structure teams, and what we expect of the people on teams.

Manager-led teams are often like ski teams. Each person is selected for her individual skills and abilities as a skier. Each excels as an individual. All the members of a ski team are working for a common goal—winning the competition. Each contributes to success by making it down the hill as fast as she can, while passing through all the gates on the course. The coach’s job is to improve individual performance skills. A ski team trains together, but when the meet comes, each team member is on her own. They don’t win by working together; it’s the sum of the individual scores that wins the meet.

Self-organizing teams are more like soccer teams. Each player has a position on the team based on her skills; however, the team must work together in order to win the game. In addition, each team member constantly adjusts her position and actions to create the best opportunity to score (or to prevent the opposition from scoring). If the team doesn’t constantly coordinate and adjust to the current circumstances, it doesn’t stand

much chance of winning, even with star players on the team. Once the team is on the field, the coach can only observe and diagnose problems. He directs the team during timeouts and adjusts strategies through substitutions. Most of his work comes before the game, building individual skills. But even more importantly, he teaches team members to work together to achieve their goal.

Enough with sports analogies. What does this mean for you, a manager in a software organization?

Managers take a step back—but not too far back—so they can analyze performance on the team level, not just the individual level. Then, to make it easier for the team to produce software, managers get to work on the organization that surrounds the team.

How Much Self-organization Is Right for a Team?

Sometimes I see teams that reject all direction and go their own way, declaring, “We are self-organizing.” They are missing an important fact. When someone is paid by a company to be part of a team, that team exists within the organizational context. The team has a customer—someone who desires and will pay for its output (or not, in which case the team should cease to exist). If a team doesn’t have a customer who eagerly awaits its product, that’s a management problem.

On the other hand, some managers hear the word “self-organizing” and believe the team is on its own—that they can go into semi-retirement. But that’s not the case, either. In fact, it’s a risky oversimplification.

I talked to a manager in an organization that had expended millions of dollars to train teams to self-organize and self-manage. The division then eliminated all but a few management positions.

But the division wasn’t seeing greater creativity and engagement as it had hoped. Instead, highly trained technical people were leaving the organization in droves. Alarmed, the remaining managers started exit interviews. They learned that people were leaving for other companies that had traditional,

manager-led teams—not because they loved being told what to do, but because they wanted to focus on the technical work that they loved.

In the rush to embrace self-organizing and self-managing teams, top management had loaded too many management tasks onto the teams. Team members were spending less than 50 percent of their time on actual technical work. Now that’s a management problem.

In reality, it’s a delicate balance, and the “right” level of self-organization depends on the context.

Most self-organizing teams take on management in these three areas:

- Managing work and monitoring progress
- Managing team membership
- Setting direction within the organization

MANAGING WORK AND MONITORING PROGRESS

On self-organizing teams, the team creates the iteration plan, breaking down the tasks and activities to turn requirements into working software. And team members manage and monitor their own task accomplishment. But, self-organizing teams aren’t relieved of the responsibility to report their status to management. Self-organizing teams use burndown charts and task walls to make visible both progress and problems.

Task management is usually the starting point for self-organizing teams. And for some teams, this is where they remain. There’s nothing wrong with that, and it actually makes sense for teams that won’t be together a long time.

Managers can help teams attain this level by refraining from continually asking about progress and from adding more tasks.

If team members aren’t yet able to plan and monitor their own work, it’s more helpful to coach them to identify the steps and break tasks down into one-to two-day chunks of work than to step in and take over planning. Coach and show rather than do.

MANAGING TEAM MEMBERSHIP

Some self-organizing teams manage their own membership throughout the life of the team. In some companies,

management announces the projects, and the developers (the programmers and testers) with the relevant skills and understanding of the project form their own teams.

It’s far more common, though, for managers to assign people to teams, at least initially (although calling a group of people a team doesn’t actually make them a team). I recommend that managers use a collaborative hiring process for self-organizing teams once they’ve formed. Sticking people onto a team without consulting existing team members is a recipe for ungelling the team.

Members of mature teams may themselves remove people from the team. A colleague told me a story of how they managed a lone-wolf off the team. One engineer had committed to follow Extreme Programming engineering practices but later violated his agreements with the team. When two team members confronted him, he informed them they’d be lost without him and threatened to find another job. His teammates offered to help him buff up his résumé, and he was gone within a week—no manager involved. If an employee who is unable or unwilling to work within the team will not leave of his own accord, then the manager needs to step in and deal with the situation from an HR perspective.

Don’t count on this happening early in the life of the team. Before team members can manage someone off the team, they need to be able to manage someone *onto* the team by giving peer-to-peer feedback and constructively dealing with unwanted behavior.

SETTING DIRECTION WITHIN THE ORGANIZATION

Self-organizing teams make commitments, but they don’t set product direction. The vision and definition of the product comes from the customer or product owner. As a team matures, team members may influence and even help co-create the product as they develop trust and a collaborative relationship with the customer. But this is a relationship that requires high domain knowledge as well as technical knowledge and a high level of trust.

It seldom makes sense to push so

much management work down to the team that the team doesn't have enough time left to develop software. It's a balance between building ownership and engagement and keeping people focused on the kind of work they've trained to do.

The appropriate level of self-management depends on the context. Consider how long the team will stay together. It takes time for a team to learn self-management skills. If the team will only be together for a few months, it probably doesn't make sense to train team members to manage team membership or involve them in setting direction within the organization. On the other hand, if team members will have a long life together, it's likely they'll be adding team members, so teach them to participate in collaborative hiring.

Manager as Coach and Consultant

Teams need to know how their company makes money. This may seem painfully obvious, but somewhere in my distant past, I met a group of programmers in a financial services company who suggested that the fund managers avoid trading options because programming for options was difficult. Once the programmers understood how much money there was to be made trading options, they saw the wisdom in buckling down and figuring out how to write the program.

Team members need to understand how their company fits into the market and what their company's aspirations are. For example, one manager communicated that the company aspired to handle 100,000 simultaneous users. Team members had this fact in their minds as they built the Web site incrementally. While the software they built in the first few iterations couldn't handle the load, they made choices that made it feasible to build up to that level. When the team understands the business, the team will make better technical decisions.

Some teams explicitly defer to the customer or product owner all decisions about the cost and benefit of building features. I find that the more team members know, the more they can ask intel-

A Tale of a Too-Hands-Off Manager

I recently worked with a team that was struggling. One of the team members, Tad, wasn't playing by the rules the team had established. When the team formed, members agreed that each day they'd have a fifteen-minute stand-up meeting to report on progress. The team members agreed that they'd chunk their work into tasks that were a day or two long, knowing that would help them stay on track and make progress visible.

But all was not well. When the other team members picked a story off the task wall, they'd break it down into tasks and post them back on the wall. When Tad picked a story, the card disappeared from the wall. At the stand-up meeting, his reports were vague. "I'm coding," he'd say.

After a couple of rounds of such murky reports, two team members, Sally and Will, sat down with Tad.

"What's up?" Will asked. "Do you need some help?"

"No," Tad replied. "I'm working on it."

"But we don't know what progress you're making," Sally said.

"I'm working on it," Tad replied, staring at the notebook in his lap.

"Tad," Will implored. "You agreed to report tangible progress every day. We all did."

Tad closed his notebook. "I'm working on it. Now leave me alone so I can keep working on it."

The team's agile coach tried, too. After a vague report in the next stand-up, the coach probed for more information.

"What exactly are you working on, Tad?" he asked.

"The reset feature," Tad replied.

"When will you finish it?" the coach asked.

"I'll be done when I'm done," Tad replied. "I'm working on it."

"That's not good enough," the coach said, laying down the law. "When you don't report demonstrable progress, it hurts the team. We don't know if we're on track or not. We can't give our customer an accurate read on meeting our iteration goal."

"I'm working on it," Tad replied.

After weeks of Tad's odd behavior, the coach approached the development manager. "We've tried everything we can think of," the coach said. "We need you to step in and help us get Tad on track."

"You are self-organizing," the manager demurred. "You need to figure out how to deal with team members."

The team tried everything it could think of to induce Tad to report on his work, finish his work, and see his contribution—or lack thereof—to the team's iteration goals. And as their manager continued with his "hands-off" attitude, team members' morale plummeted. It felt like their manager had abandoned them. He had.

Fortunately for this team, the hands-off manager left the organization. The new manager recognized the team was struggling and handled the performance issue.

Managers of self-organizing teams need to discern when it's time to step in and when to step back, allowing the team to solve the problem on its own. How do you know when a light touch is called for and when action is needed? Ask:

- Does the team have the knowledge to solve this problem?
- Does the team have the experience to solve this problem?
- Does the team have the will to solve this problem?
- Does the team have the courage to solve this problem? If so, give the team some time to work out the issue. If you sense team members are stuck, coach them to use open-ended questions to recognize the resources they already have to address the issue.
- Is this a new area of responsibility for the team? If the team is taking on a new responsibility—one that's been yours in the past—offer a process to help the team make the decision or solve the problem. Be wary of stepping in. If it looks like you are trying to take the decision back or influence the team, your credibility is lost.
- What are the consequences of failure?

ligent and relevant questions that hone not only their own thinking but also that of their customers.

Team members still need coaching to improve their technical, interpersonal, and collaboration skills. Managers of self-organizing teams still meet with

sion. When members are encouraged to broaden their skills rather than deepen them in specific areas, managers need to help team members see how their cross-functional and collaboration skills will fit into the future of the organization. (And managers may have to influence the pos-

of work flowing into the team.

The second job is to eliminate unevenness in the work load—to create a steady flow. One way to do this is by using team velocity to allocate work (velocity is a measure of the work a team can finish within a specific time period).

“The first job of management is to reduce overburden.

In manufacturing, overburdened machines break down.

**In knowledge work, overburdened people
make mistakes, fall ill, or burn out.”**

people to coach, address HR issues, and facilitate career development. They help by providing context, developing skills, and helping people see new options.

I’m a big believer in one-on-one meetings. But with a self-organizing team, the timing and focus are a little different. For example, on one team, a team member distressed another team member by listening to his voice mail on speakerphone—even when the messages included intimate details of his relationship with his girlfriend. The manager used a one-on-one meeting to coach the team member to give feedback directly to the offending colleague.

Beware of discussing task progress with self-organizing team members in one-on-one meetings—unless you are coaching on how to accomplish tasks. Monitoring progress in a one-on-one meeting sends a mixed message. It conveys that team members are still making performance commitments to you and not to their teammates. It says you don’t really trust the team to manage its own task accomplishment.

With self-organizing teams, weekly one-on-ones usually aren’t necessary unless there is a specific coaching issue. With self-organizing teams, monthly or bi-monthly meetings seem to be about right.

Because self-organizing teams are cross-functional and most organizations have functional career ladders, members often struggle with career progres-

sibilities for that future by working with HR to change the way people are compensated and promoted.)

Work on the Process

Edwards Deming describes an approach where staff works *in* the process and management works *on* the process—using the detailed knowledge of the people doing the work. Managers of self-organizing teams work on improving the organization’s processes so that everyone can perform better.

Functional organizations tend to focus on specialized skills and excellence at the detail level. Managers of cross-functional, self-organizing teams see their organizations as systems of interdependent parts. An overall system view leads to improving the entire system, rather than focusing exclusively on individual performance. Companies still need to hire competent and skilled people, of course. And managers need to make sure that those competent and skilled people are supported—not hindered—by organizational structures, policies, and procedures.

Lean manufacturing suggests three things a manager of a self-organizing team should do. The first job of management is to reduce overburden. In manufacturing, overburdened machines break down. In knowledge work, overburdened people make mistakes, fall ill, or burn out. So help the team hold to a sustainable pace by managing the amount

Create an even pace by allocating work in timeboxes based on measured ability to complete work. Over time, as the team matures, velocity may naturally increase. Even flow of work means predictable delivery.

Finally, managers need to eliminate waste. Anything that does not directly add value to a product is considered waste. Extra processes, task switching, and partially completed work are examples of waste in software development.

These three things—reducing overburden, eliminating unevenness, and eliminating waste—work synergistically to increase the capacity of the system to produce software.

When managers focus on eliminating waste without attending to reducing overburden and eliminating unevenness, they may actually do damage by eliminating slack. When slack is eliminated, people cannot respond to unexpected events and things fall apart.

Managers of self-organizing teams have plenty to do. People will still want to develop their skills and further their careers. And you’ll work to improve the organization and the process to enable teams to deliver software. Manager, your work has just begun. **{end}**



Software Tester Certification

Certified Tester—Foundation & Advanced Level Training



More than 90,000 Certified Testers Worldwide. Why Not You?

SPRING 2009 FOUNDATION LEVEL TRAINING

Houston, TX	February 17–19, 2009
Toronto, ON	February 17–19, 2009
Mountain View, CA	February 24–26, 2009
Atlanta, GA	February 24–26, 2009
Los Angeles, CA	March 3–5, 2009
Tampa, FL	March 3–5, 2009
Raleigh, NC	March 10–12, 2009
Scottsdale, AZ	March 10–12, 2009
Nashville, TN	March 17–19, 2009
Vienna, VA	March 17–19, 2009
Boston, MA	March 23–25, 2009
Boise, ID	March 31–April 2, 2009
San Francisco, CA	April 14–16, 2009
New Jersey Area	April 14–16, 2009
San Diego, CA	April 20–22, 2009


Milwaukee, WI	April 21–23, 2009
Denver, CO	April 28–30, 2009
Bethesda, MD	April 28–30, 2009
Orlando, FL	May 3–5, 2009
Birmingham, AL	May 12–14, 2009
St. Louis, MO	May 12–14, 2009
Seattle, WA	May 19–21, 2009
Philadelphia, PA	May 19–21, 2009
Chicago, IL	June 1–3, 2009
Las Vegas, NV	June 7–9, 2009
Oakland, CA	June 9–11, 2009
Detroit/Ann Arbor, MI	June 9–11, 2009
Providence, RI	June 16–18, 2009
Albuquerque, NM	June 16–18, 2009
Portland, OR	June 16–18, 2009

www.sqetraining.com/certification

- **Basics of testing** – Goals and limits, risk analysis, prioritizing, completion criteria
- **Testing in software development** – Unit, integration, system, acceptance, and regression testing
- **Test management** – Strategies and planning, roles and responsibilities, defect tracking, and test deliverables



On-site Training Available—For additional savings, bring this course to your organization for team training.



6 THINKING HATS for TESTERS

*A Haberdashery of
Testing Improvement
by Julian Harty*



Over the past few years, I've been using some simple concepts that have helped improve my ability to deal with issues from multiple distinct viewpoints. I learned about these concepts from Edward de Bono's book, *Six Thinking Hats* [1]. I have applied his concepts to software development and software testing and found them to be both useful and practical. I believe many of you will benefit from using them, too.

Six Thinking Hats

The six thinking hats are a metaphor for six distinct viewpoints we may take when thinking. Rather than a complex style or name, each hat has a color associated with it to make the hats easy to recognize and remember.

HAT EXPLANATION	
	WHITE HAT Focuses on the facts
	YELLOW HAT Focuses on positivity and encouragement
	RED HAT Focuses on emotions, feelings, and hunches
	GREEN HAT Focuses on creativity and new ideas
	BLACK HAT Focuses on what can go wrong
	BLUE HAT Focuses on purpose, control, and organization

By identifying the different modes of thinking, we can choose how to think, rather than be led by our natural (and often limited) style. De Bono suggests we state the hat we're currently wearing ("I want to white hat this discussion for a while") or even ask others to put on a particular hat, especially one that is not natural for them ("Let's all put on our green hats and create some new ideas.

No black hatting for a while; don't evaluate the practicality of our ideas, just generate as many as we can").

In his book, de Bono provides a more detailed explanation of the hats.

WHITE HAT

The white hat is about information. The information can range from hard facts and figures to things we believe but don't yet know. It may include second-hand facts, such as "The CEO of our competition claims its product can sort widgets by color and texture," or state other people's beliefs or feelings, such as "Jack is angry with the project team and blames the testers for holding up the launch." Sometimes the information may be contradictory. If so, we decide its validity later. The white hat is neutral and is concerned with the world as we find it. Wearing our white hats, we ask questions such as: What information do we have? What information do we need? What questions do we need to ask to get the information we need?

YELLOW HAT

The yellow hat represents sunshine and optimism. It is deliberately positive. It challenges us to find benefits and to believe it is possible to put ideas into practice. We use the yellow hat to answer the

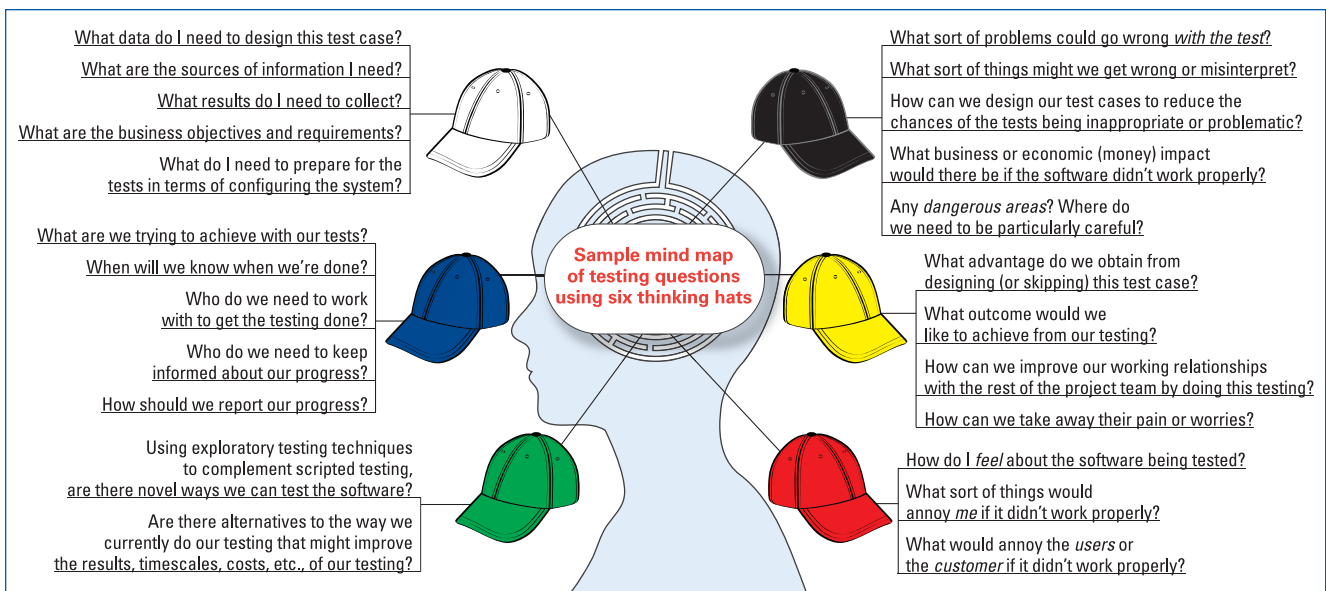


Figure 1

question “Which of these ideas can we apply, and what would the benefits be?” According to de Bono, people sometimes struggle to recognize the benefits in their own ideas, so the group may need to help nurture other people’s ideas using the yellow hat.

RED HAT

Red represents anger, passion, love, fire—what energizes us, annoys us, and upsets us. The red hat provides a safe, controlled opportunity for people to say how they feel. Putting on the red hat makes intuition, hunches, and emotions valid. When wearing the red hat, you do not have to justify your statements. Without the availability of the red hat, people’s feelings are often incorrectly expressed as facts—“It’ll never work here because no one wants it.”

GREEN HAT

Green represents fertility and new growth. With the green hat we seek new ideas and fresh approaches. Colloquially, we try to “break the mold” or “get out of the rut.” When wearing the green hat, we often brainstorm to develop new ideas. Note that ideas are not evaluated or criticized here. Sometimes ideas that seem silly when we first encounter them become key drivers. For example, “We will pay our customers to spend money with us” has led to loyalty schemes in a wide range of industries.

DE BONO’S THINKING HATS

 BLUE HAT	What have we learned? What are the next steps?
 WHITE HAT	What facts do we need? How can we get those facts?
 RED HAT	Gut feelings, instincts? What do my senses tell me about this?
 YELLOW HAT	Advantages? The best possible outcome?
 BLACK HAT	Risks and problems? Worst-case scenario?
 GREEN HAT	Creative approaches? Fresh, new ideas?

Table 1: Six thinking hats template. A simple table can help us make notes for each of the six hats.

BLACK HAT

Black represents darkness and allows us to be careful and cautious—without being overly negative. The black hat helps us identify things that might be incorrect, might go wrong, and might be risky or dangerous, which is particularly useful in software testing. The black hat taps into our natural fears about safety and security and helps us voice our concerns without their being viewed as negative or unhelpful.

BLUE HAT

With the blue hat, we think about our thinking, define our purpose, control the use of other hats, and set out the next steps. In a group, the blue hat may be worn permanently by the facilitator.

The Six Thinking Hats in Testing

Here are some examples of using the six thinking hats in testing:

- Improving our working relationships
- Reviewing documents and code
- Designing test cases
- Assessing the product
- Discussing release readiness

IMPROVING OUR WORKING RELATIONSHIPS

Software development may become adversarial when groups blame each other for problems and delays. For instance, the classic “If you’d given us better specs, we wouldn’t have had

grown quickly, work globally, and have teams with a high degree of autonomy in their software development practices. The six thinking hats are useful when performing code reviews. Use the black hat to identify events the code might not handle properly, the yellow hat to look for positive aspects of the code, the white hat when assessing the corresponding unit tests, and the green hat to collect new and exciting ideas for testing and improving the code.

DESIGNING TEST CASES

Generally our testing fits within a larger context—that of shipping working, desirable software cost effectively and on time. One standard prac-

“A key skill is the ability to review artifacts related to software development.

We may review material ranging from rough requirements documents

to thousands of lines of source code to test cases. Sadly, we are often less

effective than we’d like to be.”

When group members use the blue hat, individuals can make procedural suggestions—for example, which hat to use next. Participants may also use the blue hat to clarify points, such as “Is that a white hat comment or a red hat comment you’re making?”

A FINAL NOTE ABOUT HATS

Although we might be tempted to categorize someone as emotional or negative, we need to resist the urge to do so. Rather, we need to encourage people to try out each hat so they can contribute more fully to the goals and objectives of the team. Everyone needs to use any and all of the hats.

We can use the hats individually or in groups. When used in a group, every member of the group uses the same color simultaneously before the group switches to another color. Tip: I use inexpensive, colored baseball caps (less than \$15 for a set of six) in group situations to help us keep focused on the current hat. By using a common color, the group can be much more focused and productive.

to spend an extra three weeks and \$100,000 trying to find out what the users really wanted.”

By using the six thinking hats, we can disentangle facts from emotions (white and red hats) in order to present careful and cautious views (black hat) balanced by a mix of positive thinking (yellow hat) and creative suggestions to improve things (green hat). And with our blue hats we can consider the “bigger picture” of how our work fits with the project and company objectives.

REVIEWING DOCUMENTS AND CODE

A key skill is the ability to review artifacts related to software development. We may review material ranging from rough requirements documents to thousands of lines of source code to test cases. Sadly, we are often less effective than we’d like to be. The thinking hats allow us to review the material from six distinct perspectives. At Google, code reviews are a common practice that helps us maintain a high standard of working software, even though we have

tice is risk-based testing—evaluating risks in order to decide what, how, how much, and when to test.

We can improve our understanding of the risks and thus design appropriate tests using the six thinking hats throughout the process. Each hat helps improve the effectiveness and appropriateness of the resulting tests.

The mind map in figure 1 provides examples of questions created by viewing the problem from the perspective of each hat.

ASSESSING THE PRODUCT

The following case study provides an example where the project team reviewed some new software prior to accepting it from the supplier.

A small software testing consultancy commissioned a new version of its Web site to be developed by another company. The consultancy was responsible for user acceptance testing (UAT) and used de Bono’s six thinking hats as part of the UAT process. The initial review, including an introduction to the hats






TESTING IDEA	HAT	COMMENTS
Analytical School		Includes white box testing techniques and focuses on applying proven techniques. These may measure the effectiveness of techniques and even the testing—e.g., Dot Graham's defect-detection-percentage measure.
Factory School		Testing is only part of a bigger picture and must: <ul style="list-style-type: none"> • Match the business drivers • Integrate with other tasks • Fulfill the project and other objectives
Quality School		Process oriented; leaves little to chance. For instance, we may want to implement regression testing to reduce the chances of old bugs reappearing.
Context-driven School		The tests are based on the context, which tends to be an iterative, evolving, creative process.
Test-driven School		Commits to testing early, before the main software is written; passionate about testing. Some practitioners are described as "test infected."

Figure 2

took an hour. Each member of the team used a template similar to the one shown in table 1.

As team members reviewed the current version of the Web site, they made notes under the various hats. The hats encouraged everyone—even the non-technical team members—to voice their opinions rather than feeling excluded. At the end of the meeting, the notes were collated and analyzed. They covered various key aspects of the site, including:

- Core functionality: how to update the content
- Usability: page size, aesthetics, look and feel, etc.
- Privacy and security: Who can access information? How well is the client data protected? etc.
- Performance: only in general terms with no quantitative definitions
- Process testing: including the interaction between people and the functionality offered by the Web site; who does what and when

As a result of the feedback, the launch of the new site was delayed until various changes were made. The entire team also felt much more involved in the success of the project and were more committed

to helping improve the new site.

DISCUSSING RELEASE READINESS

Another area where the hats can help is for release decisions—should this software go live or not? Using the hats can help provide fresh insight into whether the software is likely to meet its release criteria.

For example, I participated in a launch meeting for software due to ship in five days. I had nagging doubts about whether the software was ready but didn't want to be perceived as the doom-ridden, dissenting voice on the project team. Just before the meeting, I spent about forty minutes sketching out what I knew about the state of the software using the template. In the blue box, I recorded my aims and objectives for this exercise and for the launch meeting. I collected facts in the white box such as the known bugs and release cycles. A couple of the bugs particularly worried me, so I noted my feelings in the red box; I'd be embarrassed if the software launched with them. Key project and product risks were added to the black box indicating concerns, problems, and risks. In the yellow box, I noted our goals and what we hoped the software would

achieve once it had been launched. After filling out the remainder of the boxes, I started brainstorming ways we could increase our chances of launching quality software, soon, and listed them in the green box.

When the meeting started, I was well prepared and able to explain my concerns and goals and even suggest ways we could help increase the likelihood of delivering working software. The ideas and suggestions were well received, and we decided to run a mini test-fest later that day, which found several high-severity bugs and identified a number of areas where usability could be significantly improved. As a team, we increased our velocity, improved our working relationships, and produced better software—all by using the six thinking hats.

Discovering Testing Hats

In his presentation, "Schools of Software Testing" [2], Bret Pettichord describes a number of different approaches to software testing:

- The analytic school—sees testing as rigorous and technical with many proponents in academia
- The factory school—sees testing as a way to measure progress with



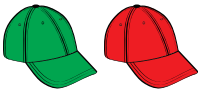
TESTING IDEA	HAT	COMMENTS
Benefit-driven testing		Looking to show the benefits of the software being tested—e.g., showing what works rather than what is broken
Risk-based testing		Only test to assess or mitigate risks related to the project
Exploratory testing		Creative, seeking alternatives and fresh ideas for finding bugs

Figure 3

emphasis on cost and repeatable standards

- The quality school—emphasizes process, policing developers, and acting as the gatekeeper
- The context-driven school—emphasizes people, seeking bugs that stakeholders care about
- The agile (or test-driven) school—uses testing to prove that development is complete; emphasizes automated testing

Sticking to a single school is risky and may limit the effectiveness and productivity of testing. I use Pettichord's schools to help me consider my testing from various perspectives—to reduce the chances of missing things. By using the six thinking hats, we can review each of the schools in order to pick ideas and concepts that should improve the quality and effectiveness of our testing. For example, from the analytical school, I may select several test-case-design techniques to use.

These various ideas seem to map to particular testing hats, which combine the concepts proposed by de Bono with some of these ideas from the software testing community, as shown in figure 2. The goal is to help us improve the ways we test software by giving us at least six different ways to approach testing.

Figure 3 maps some additional approaches to testing with the six thinking hats.

Using these tables, which map the various testing ideas to the colors of the

hats used by de Bono, we can now reassess our software testing and hopefully find additional ways to look for bugs that may have hidden from us in the past.

You can even use the six thinking hats to explore the application of each of the approaches mentioned in this section—for instance, what are the advantages and risks of using test-driven development? By applying each of the hats in turn, you may decide that the effort in applying one or more of the schools is worthwhile.

Now, It's Your Turn

Now that you've read about how the six thinking hats worked for me, I'd like you to try them out in some of your work. Pick a current issue, big or small, related to your work. Use a copy of the template in figure 2 and take thirty minutes for the activity. Start with the blue hat and jot down what you'd like to achieve from this activity—are you more interested in learning how to use the hats or in solving a technical problem? Then spend a few minutes (two to four) “wearing” each of the thinking hats in turn; record ideas that come to you. If you think of an idea that belongs with another hat, write it down quickly under the hat that seems most suitable and then return to the current hat until you've finished the allotted time.

Once you've used each hat once, return to the blue hat and decide what to do next. Would you like to use each hat again, think of ideas (using the green hat), or concentrate on deciding what to

do next (remaining with the blue hat)? Just before you finish the exercise, record your experiences with using the hats. Were they useful? Would you pick a different hat order next time? How can you improve the effectiveness of using the hats?

Don't be constrained by the mapping presented here; you may want to adapt the model to suit your needs and interpretation. What's more important than picking a particular hat (or hats) is for you to gain fresh insights and perspectives on how to test effectively and efficiently.

Like any skill, applying the six thinking hats is something that improves with practice. Thankfully, the learning curve is gentle and short, and you should be able to make tangible improvements in your testing within a few hours. **{end}**

REFERENCES:

- [1] De Bono, Edward. *Six Thinking Hats*. Back Bay Books, 1999.
- [2] Pettichord, Bret. “Schools of Software Testing.” 2007. www.io.com/%7Ewazmo/papers/four_schools.pdf

Sticky Notes

For more on the following topic go to www.StickyMinds.com/bettersoftware.

- Further reading

THE KEY TO INTERVIEW



ISTOCKPHOTO

D GOOD EWING

ASK THE RIGHT QUESTIONS AND THE QUESTIONS RIGHT

by Robert Sabourin & Lee Copeland



It is better to know some of the questions, than all of the answers.
—James Thurber

Recently, Lee Copeland and I completed a test process assessment for a major financial services company. Our assignment was to understand this organization's current testing process and make recommendations for improvements.

The foundation of any successful assessment is interviewing a diverse cross section of the staff. We interviewed team members who were involved in current software testing projects. We spoke with people from different project types including new development, product updates, and emergency releases. We interviewed people working with different technologies such as mainframe, client-server, Web, and voice technologies, and we met with team members from different office locations. Listening to this varied group helped us build a picture of this organization's software testing process.

Although Lee and I have different professional experiences, our elicitation techniques are quite similar. We interviewed team members, carefully listening to their stories, and took detailed notes. Through their responses, we were able to piece together a clear and realistic picture of our client's testing process. As we reviewed our notes to analyze the situation, it occurred to us that we could also analyze our questions to learn more about how we perform interviews and how we could improve our own elicitation processes.

Our Elicitation Process

While we had a formal list of topics we needed to ask about, most of our questioning was exploratory. We wanted to learn about the client's testing realities, so we captured information without bias or interpretation. As we learned, we tried to uncover more of the narrative. We asked further questions based on our findings: Open-ended questions led to rich stories; focusing questions closed gaps capturing important, missing details; and closing questions helped wrap up the interview. We wanted to identify

additional stories and any important, omitted details.

We asked interviewees to share stories about recent critical incidents. We sought examples of excellence or demonstrations of failings: "Tell a story about a project that worked really well"; "Tell us a story about a project that failed miserably." We also asked for stories of typical project experiences. These stories helped us identify several objects of testing, including plans, documents, reports, test cases, and other artifacts.

Once we understood the objects of testing, we could relate them to the actions of testing. "What do you do with these objects?" "How do you create them?" "How are they processed?" Often the nouns of the narrative told us what the objects of the testing project were. Often the verbs of the narrative told us what actions were done to these objects. We inquired about the sequence of actions and the evolution of objects, so we were able to compare and contrast process flow between similar stories from different sources. Differences led to rich comparative questions, which helped us ask test leads and managers "why?"

After analyzing the client's current practices and writing test process improvement recommendations, we returned to our interview notes and undertook a thorough analysis of our questioning styles. We made a list of the questions we had asked, organized them into categories, and sought to discover commonalities and flows. The following describe the types of questions we found effective.

SETTING THE STAGE

Interviewees need comfort; the appearance of outside consultants may appear threatening. Interviewees are not sure why they have been "summoned to appear," and they may be worried that the things they say will be used against them at some future time.

We began by putting the interviewees at ease by setting the stage for our dis-

cussion. We made sure they knew why we were there. We explained our commission from their management, what approach we were using, and what we were trying to learn from the information we collected. We also explained why *they* were there—that they had been selected because of their in-depth knowledge of the organization's processes.

During the interview, we took notes as the interviewees related their stories. At the beginning of the interview we mentioned that we were going to take notes so that we could capture and remember the details of their stories. We also stated that they could and should speak freely about their organization—that their names would not be attached to any negative comments they made. It is absolutely vital that this promise be strictly honored.

Our handwritten notes were of two types—directly related to the test process improvement model we were using and free-form notes recording anything interesting we learned. Rob also used mind maps to create a visual outline of their stories. Mind maps allowed us to create a visual understanding of the testing process described by each interviewee and helped us detect missing actions, objects, inconsistencies, and asymmetries in each story. See the StickyNotes for more information on mind maps.

We often began the interview with these questions: "Do you know who we are and why we are here?" and "Do you know why you are here?"

Interviewees would tell us what they thought we were doing, then we discussed and clarified our purpose before diving into the interview. We wanted to show respect to the interviewees, so before starting in with questions, we thanked them for taking time out of their chaotic schedules to meet with us.

BUILDING RAPPORT

"I didn't expect a kind of Spanish Inquisition" is a classic line from a Monty Python skit. We didn't want interviewees to feel they had been dropped into the

middle of an inquisition, so, in order to build rapport, we often began by asking questions about the local community—what to see in town, interesting cultural activities, sporting events, entertainment, and, of course, fishing. We asked where we could find good catfish dinners and Indian cuisine. We also asked where test consultants could do their laundry—always a great ice breaker. These questions were good natured and never scripted. We sincerely wanted to learn. Note that it is generally not a good idea to ask personal questions of interviewees. While acceptable to some, others might be offended, and starting an interview this way is sure disaster.

TELLING STORIES

We encouraged interviewees to share specific, recent experiences to help us learn what they do. There was always a risk that interviewees would start quoting process manuals even though the documents may rarely be followed. Our goal was to learn what really happens. Our questions were designed to elicit a narrative. Many testers have stories of heroic initiatives or terrifying failures, and relating such tales can help expose important evidence.

We let the interviewee tell his story with minimal interruption, but occasionally redirection was needed to get back on topic. We were interested in the facts and avoided jumping to conclusions. Interviewers must avoid interrupting the story with suggestions or recommendations; patience is in order. Once the narrative was told, we used non-judgmental, clarifying questions to fill gaps or explore new ideas.

We used these tactics to learn about interviewees experiences:

- Tell us what you do. (State facts, not theories.)
- Tell me a story about when _____ worked. (Describe a positive experience.)
- Tell me a story about when _____ failed. (Describe a negative experience.)
- What happened the last time? (Describe a recent experience.)
- How do you know it's time to start? (Define the beginning and entrance criteria.)

Task Analysis

An informative survey of task analysis and associated interview techniques can be found in the book *Task Analysis Methods for Instructional Designers* by Jonassen, Tesmer, and Hannum. Their description of the critical incident method is a particularly useful questioning strategy and has been used to learn critical elements of many important domains such as how to fly a fighter jet and how to implement exploratory testing.

The interviewee is asked to relate a critical incident which illustrates how their expertise was used to resolve an important problem. The following questions frame the interview:

- What led up to the incident?
- What did you do?
- Why was this incident important?
- When did it occur?
- What was your role at the time?
- What was your level of experience and expertise at the time of the incident?

These questions guide a narrative that then can be reviewed with the interviewee to identify important tasks, skills, deliverables, and priorities in accomplishing the task at hand. Once the story has been told, a timeline is made and the tasks are reviewed in chronological order. Many gaps can be identified and deeper elaboration of parts of the story comes out during the chronological review.

- How do you know when you are finished? That testing is complete? (Define the end and exit conditions.)
 - Can you give me an example of _____? (Relate actions or activities.)
 - Can you paint a picture for me? (Create a metaphor encouraging an expanded or more complete story.)
 - How do you do _____? (Elaborate on methods, techniques, approaches, and any means of selecting them.)
 - A solution is _____. What is the problem that it solves? (This question exposes an interviewee's knowledge of the relationship between actions or objects and his purpose in the project.)
 - Can you quantify that? (Help the interviewee describe things in objective, measurable terms to allow comparison of elements of the narrative.)
 - Have you seen any evidence of that? (Encourage the interviewee to point out a specific instance or event that illustrates a stated generalization.)
 - You do ____; do others do ____? (This allows us to learn about other people or roles covering similar actions. We may identify a new source of information or an insight into a widespread practice.)
 - How did it get this way? (Probe for the root cause of a situation or condition.)
 - We were talking about _____. Are there other _____? (Is the object a member of a set, are there related objects or actions we should be studying?)
 - When someone says "I don't know" reply "I think you do know," then listen. (In some cases the interviewee may state possible reasons or speculations given time to reflect.)
 - And what are the benefits of that? (Does the interviewee know the value of an action?)
 - And what difficulties does that cause? (Does the interviewee know the consequence of an action?)
- ## PROBING FURTHER
- Some candidates omitted relevant

stories or experiences. These are a few questions that were useful to learn more and often led to new or related stories:

- What else would you like to tell us?
- Is there anything else you'd like to share with us?
- What else should we know?

To find out what really happens on projects, avoid asking questions that require the interviewee to judge his peers. Avoid questions that would place blame on individuals or teams. Instead, focus on behaviors and deliverables rather than the individuals.

MAKING DISCOVERIES

Interviewees occasionally need guidance to help them advance their narratives. Developers and testers can get pretty absorbed in describing a single activity, losing sight of where it fits into their stories. Some questions helped interviewees become better story tellers. They learned to question themselves as they spoke. As interviewers, we needed to prime the pump a few times, but once the juices were flowing, a rich narrative often followed. Some questions that helped to move the conversation along include:

- What happens next? (Sometimes the interviewee just stops the narrative; prodding about the next step helps it advance.)
- How was it previously? (We encouraged the interviewee to elaborate on why this particular in-

cident was different from others. We tried to trigger objective and subjective comparisons. Better or worse?)

- Is this recent? (We tried to get the interviewee to give us a place in time to anchor the story.)
- Can you give me an example? (When an interviewee used a generalization we tried to elicit a specific example that demonstrated the point. Occasionally an interviewee offered a generalization—"We always do this"—but has trouble identifying a specific example.)

CLARIFYING QUESTIONS

Filling gaps with an occasional clarifying question can help build a better understanding of objects, actions, roles, responsibilities, and the relationships between them. Mind mapping the interview can really help find holes in the story. Clarifying questions can also remove ambiguity. Some clarifying questions we used are:

- Which attribute is associated with which object?
- Which object was input?
- Which was output?
- What do you mean by ____?
- When you say ____, do you mean all? Some? Many? Few?
- Repeat a word or phrase with a question mark at the end.
- Can you give me an example of ____?

We used these tactics when the interviewee pivoted to a new topic or prematurely concluded the story:

- Make a statement; wait for response. (Something like: The requirement document is always reviewed before development starts [PAUSE].)
- Make a statement; then ask, do you think it's like that?
- Tell a story to trigger a response. (We interviewers know many relevant anecdotes. Each can be used to elicit related insights. These mini-experience reports only take a couple of minutes to share and they ask the question "Does something similar happen here?")

When we are concerned that the interviewee is overgeneralizing, a clarifying question can help us understand scope. For example:

- Are there different types of ____? (Is this an isolated example or are there other related objects or actions or roles?)
- Could you give us an example? (This question helps ground the narrative to a specific example—essentially we are saying "show me.")
- Let me tell a story to explain _____. (Take a small part and make a story around it because that is a story in and of itself.)
- Who should we talk with to find out more about ____? (Can someone else help complete the narrative?)

Story Telling

In *Qualitative Research and Evaluation Methods* by Michael Quinn Patton, many different elicitation and interview styles are discussed. Patton's research indicates that story telling can help add depth, detail, and meaning to qualitative analysis.

Some organizations referenced use story telling to interchange knowledge. Story telling can help in sharing key process information and illustrate to future projects what worked and what failed. Stories that are real, recent, and relevant provide compelling evidence of what current processes are and may become credible evidence supporting process improvement initiatives.

Lee Copeland's StickyMinds.com article "Telling Our Story" is another useful reference.

Not all are great story tellers even though they may have important stories to tell. The challenge is eliciting and elaborating these stories so that their retelling spreads the lessons learned across the organization and into different product or project areas.

RECAPPING AND REFLECTING

It is vital that we understand key concepts from the interviews, so we ask interviewees to validate our interpretation by restating the story from our notes. This recap often leads to additional clarification.

We sometimes retell the story in a chronological order to make sure all events are understood in sequence. We ask interviewees to identify missing activities or gaps in time:

- Could I review this to make sure I understand? (Repeat the story back to the interviewee to make

sure we are communicating.)

- So you're telling me _____. (Listen for confirmation.)

GUIDING

Interviewees sometimes stray to a part of the story that contains a lot of details (some even interesting) but does not advance the narrative. We bring the interviewee back to the main thread of the discussion with a gentle nudge: "We were talking about _____."

MAINTAINING A CONNECTION

Keeping the interviewee engaged is all about making a great connection with the interviewer. The interviewer must be sincerely interested in the interviewee's story.

Statements like the following can help the interviewee feel more comfortable:

- The reason I ask is _____. (Make sure interviewee knows why the answer could help.)
- That must be exciting/rewarding/gratifying. (Sincere statements like this can help the interviewee remember how he felt and could lead to a rich story about why he felt that way.)
- That must be frustrating. (Empathy is key when the interviewee is clearly strained or emotionally hurt by the experience being related.)
- I understand. (Show that an experienced practitioner shares his pain.)
- I love your story. (When we love the story we say so, and we say why, too. Some stories can be paradigmatic examples. These make great case studies, training materials, or exemplars and—occasionally, even a magazine article.)

CLOSING

We used two great questions to help close the interview. In doing so, we discovered a number of important pain points. The questions are:

- What question should we have asked you that we didn't? (This helps the interviewee expose problem areas and concerns outside of the scope of issues discussed so far.)

The Language of the Story

The interview takes place in the language of the interviewee, not in the language of the interviewer.

Interviewers are experts. They have a rich vocabulary of domain-specific terms. They have names for almost any object, activity, event, or report that the interviewee might mention. Unfortunately in the testing world, we do not yet have a generally accepted standard glossary of terms.

Interviewers should be very tactful. Ask appropriate clarifying questions about what an interviewee means by a term but take care not to correct the usage of an specific term. If the interviewee has special definitions for test cases, test objectives, or test procedures, it is up to the interviewer to make sure the story is understood.

Encourage the story telling to be in the language of the organization, project, or team. As the interview progresses, the candidate will be more comfortable as the interviewer uses the interviewees' terminology in further questions. Using the client's language helps keep interviewees in their comfort zone.

- If I gave you a very powerful magic wand, what would you wish for? (Think out of the box and imagine a perfect world. If you could change anything, what would it be?)

At the end of each interview, we always thanked the person for sharing his time and his story. We expressed our sincere appreciation for his time and knowledge. We explained that his comments have helped us put some more of the organizational puzzle pieces together.

PLANNING AHEAD

Between each interview we took some time to review what we'd heard, document our understanding, and plan for the next interview. We always used the results of previous interviews to develop questions for future interviews. We tried to identify gaps in what we heard and sought to fill in those gaps in subsequent discussions. Our questioning style was an "exploratory" one in which we sought to understand what was being shared, and what we had not yet heard, seeking to add pieces to the puzzle and helping us develop a better mental model of the current situation.

Questions Make a Difference!

Perhaps the most famous question in literature is from Shakespeare's *Hamlet*:

To be, or not to be: that is the question;
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them?

Lee and I may have been graced with our fair share of inquisitiveness, tact, and patience. But I doubt we have the eloquence of Shakespeare. Asking the right questions can elicit valuable information and focus us on fundamental truths and core values that really make a difference. Asking the questions right makes the interviewee comfortable in sharing his real and relevant experiences. Strategic questioning exposes rich stories whose narratives reveal what is really happening. People and their experiences help us learn how processes can be improved. Quality questioning involves concurrent exploration, learning, and adaptation. In your future work, don't just focus on the answers, also focus on the questions.

{end}

Sticky Notes

For more on the following topic go to www.StickyMinds.com/bettersoftware.

- Mind maps

Better Software magazine
StickyMinds.com

2008

SALARY SURVEY

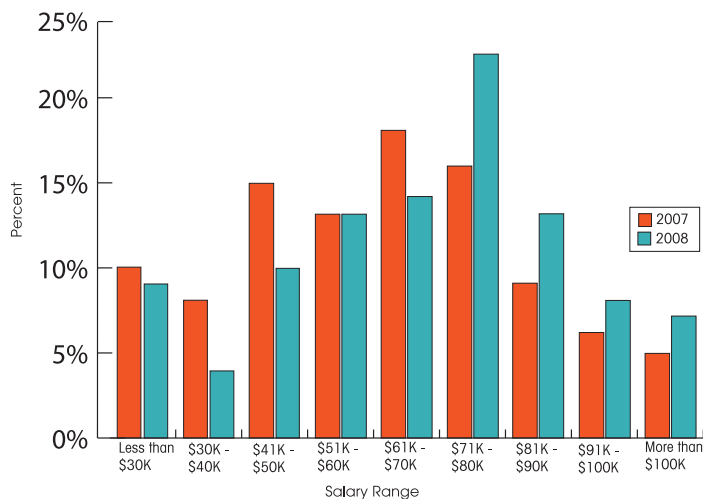
BY HEATHER SHANHOLTZER

Our 2008 salary survey summarizes the results of 700 industry respondents and was conducted on SurveyMonkey.com. This year's salary survey tracked salary figures for director-, management-, and staff-level professionals across a wide variety of fields. Our survey included questions that measured not only pay but also demographics, education, and experience.

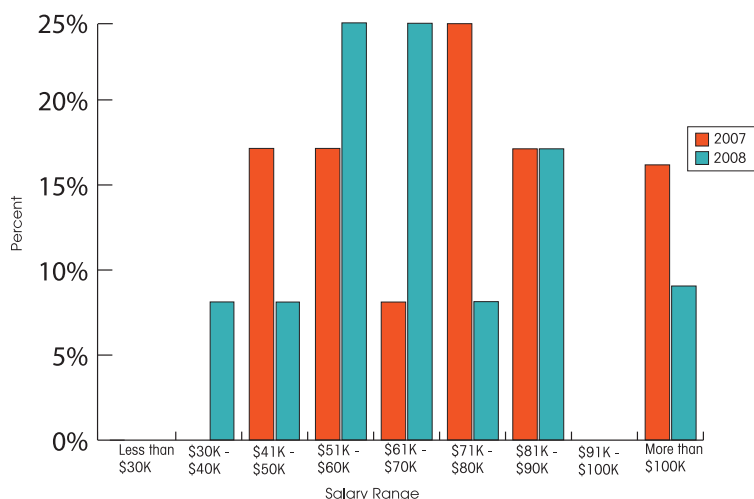


STAFF LEVEL

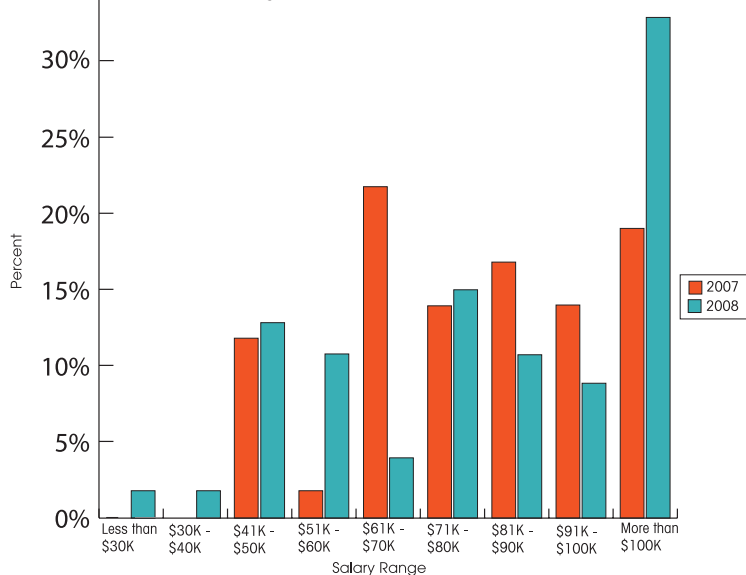
Test/QA
Change in Salary: 2007 vs. 2008



IS/IT
Change in Salary: 2007 vs. 2008



Development
Change in Salary: 2007 vs. 2008



GENDER

Male	47%
Female	53%

AGE

40 or under	50%
Older than 40	50%

EDUCATION

Some college	27%
Bachelors	53%
Masters or higher	20%

DEGREE

CS/IS	36%
Engineering	15%
Business	14%
Liberal arts	10%
Other	25%

GEOGRAPHIC DISTRIBUTION

Northeast	16%
Southeast	11%
Midwest	23%
Northwest	11%
Southwest	11%
Canada	9%
India	5%
UK/Ireland	3%
Other	11%

YEARS IN SOFTWARE INDUSTRY

5 years or fewer	23%
6 to 10 years	28%
More than 10 years	49%

YEARS AT PRESENT COMPANY

5 years or fewer	60%
6 to 10 years	19%
More than 10 years	21%

YEARS IN CURRENT POSITION

5 years or fewer	80%
6 to 10 years	16%
More than 10 years	4%

JOB FUNCTION

Test/QA	83%
Development	10%
IS/IT	3%
Other	4%

CERTIFICATION STATUS

Yes	40%
No	60%

MANAGEMENT LEVEL

GENDER

Male	60%
Female	40%

AGE

40 or under	49%
Older than 40	51%

EDUCATION

Some college	21%
Bachelors	55%
Masters or higher	24%

DEGREE

CS/IS	24%
Engineering	22%
Business	19%
Liberal arts	12%
Other	23%

GEOGRAPHIC DISTRIBUTION

Northeast	17%
Southeast	11%
Midwest	21%
Northwest	5%
Southwest	12%
Canada	7%
India	5%
UK/Ireland	6%
Other	16%

YEARS IN SOFTWARE INDUSTRY

5 years or fewer	5%
6 to 10 years	24%
More than 10 years	71%

YEARS AT PRESENT COMPANY

5 years or fewer	58%
6 to 10 years	19%
More than 10 years	23%

YEARS IN CURRENT POSITION

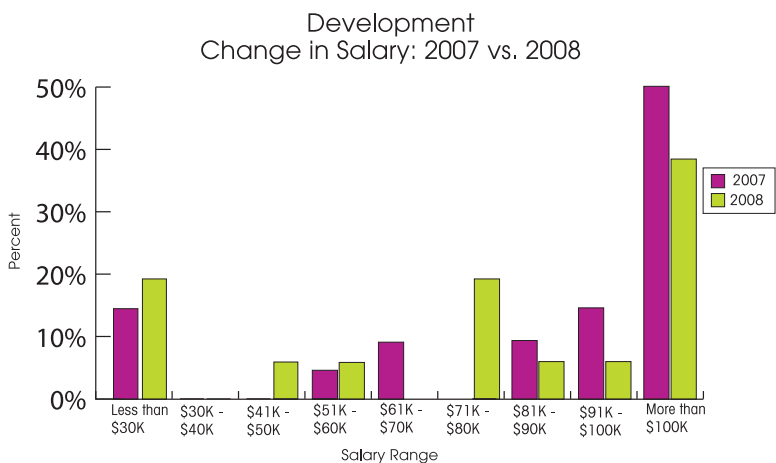
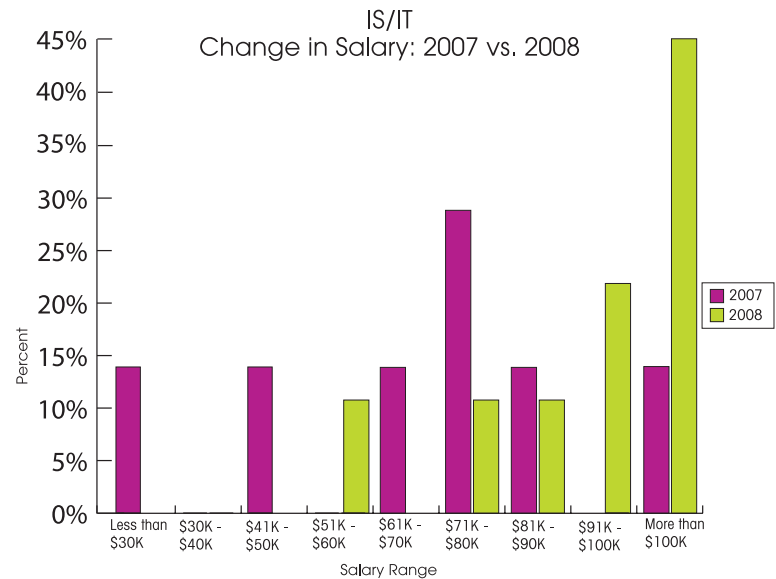
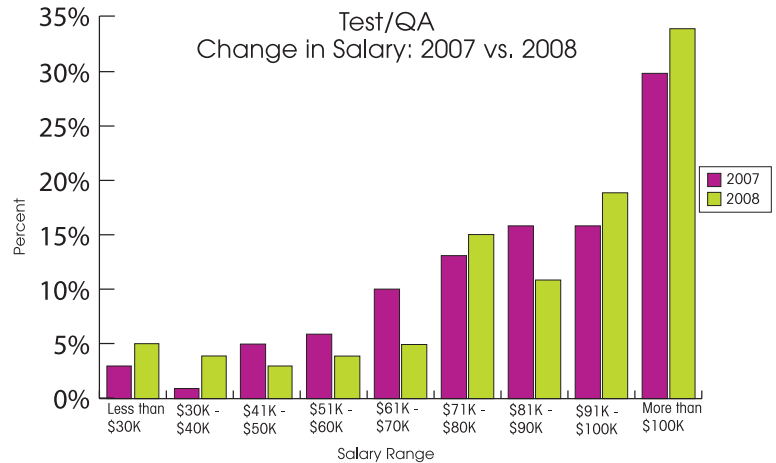
5 years or fewer	83%
6 to 10 years	11%
More than 10 years	6%

JOB FUNCTION

Test/QA	86%
Development	7%
IS/IT	4%
Other	3%

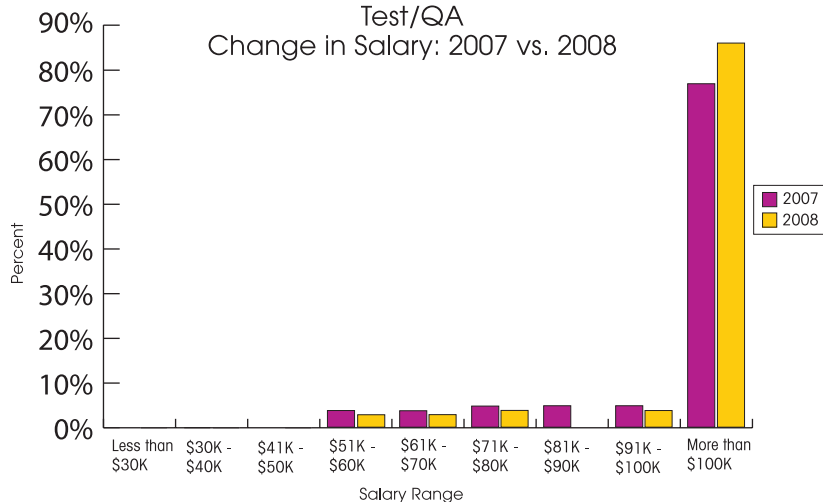
CERTIFICATION STATUS

Yes	44%
No	56%



DIRECTOR LEVEL

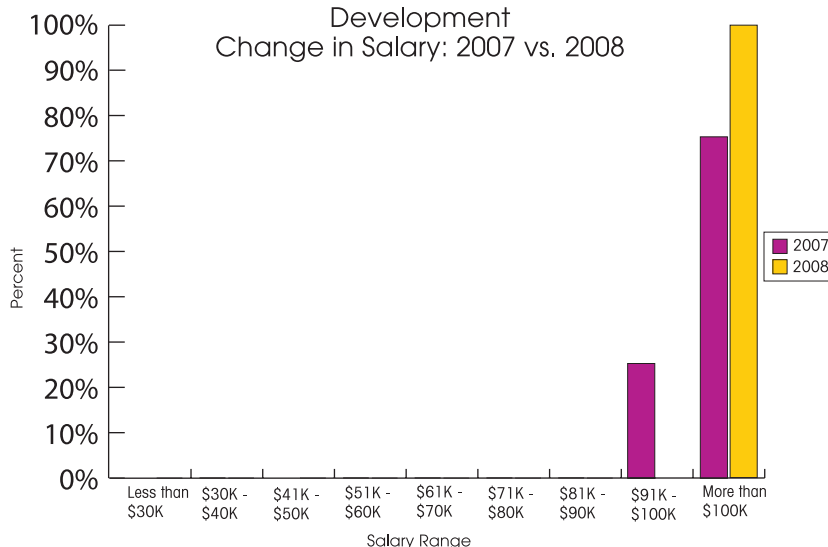
Test/QA
Change in Salary: 2007 vs. 2008



IS/IT
Change in Salary: 2007 vs. 2008



Development
Change in Salary: 2007 vs. 2008



GENDER

Male	66%
Female	34%

AGE

Under 40	40%
Older than 40	60%

EDUCATION

Some college	25%
Bachelors	44%
Masters or higher	31%

DEGREE

CS/IS	29%
Engineering	10%
Business	26%
Liberal arts	23%
Other	12%

GEOGRAPHIC DISTRIBUTION

Northeast	33%
Southeast	10%
Midwest	18%
Northwest	18%
Southwest	18%
Canada	3%
India	0%
UK/Ireland	0%
Other	0%

YEARS IN SOFTWARE INDUSTRY

5 years or fewer	0%
6 to 10 years	23%
More than 10 years	77%

YEARS AT PRESENT COMPANY

5 years or fewer	58%
6 to 10 years	18%
More than 10 years	24%

YEARS IN CURRENT POSITION

5 years or fewer	89%
6 to 10 years	11%
More than 10 years	0%

JOB FUNCTION

Test/QA	72%
Development	10%
IS/IT	10%
Other	8%

CERTIFICATION STATUS

Yes	29%
No	71%

Product Announcements

Palamida Enterprise Edition

NEW YORK, NY—Palamida has announced a new revision of its enterprise edition product. Palamida's new product rollout meets the requirements for the new era of open and secure development of software by enabling, for the first time, managers, security professionals, and lawyers to have both direct insight into the vulnerability and IP risks of applications and the ability to easily remediate the issues before they become headlines.

Newest features include:

- Online Vulnerability Updates: Email alerts with detailed information about the vulnerability, including severity and impact, and patch and remediation information as available
- Composition Markup: Enables users to annotate and tag all files and directories—from open source, commercial, or proprietary sources—with unique identifiers
- Latest Palamida Data Library: Largest DB on the market with

ability not only to report on more than one million open source releases but also to detect and verify versions as well

For additional information, visit www.palamida.com.

Software Readiness Manager

SAN FRANCISCO, CA—Coverity, Inc., has announced the availability of Coverity Software Readiness Manager for Java. The product allows development managers, release managers, and executives to objectively assess the release readiness of their critical code by combining essential data from multiple sources including Prevent, Coverity's industry-leading static analysis product. Software Readiness Manager helps development teams deliver high-integrity code that successfully meets quality standards to align product development with business goals.

Features include:

- Automatic identification of failure-prone (high-risk) code across large and complex software systems

- Translation of large amounts of data from multiple tools into actionable, prioritized recommendations for improving code
- Correlation of test coverage data with high-risk code to determine if failure-prone areas are being sufficiently tested
- Elimination of issues due to poor coding practices earlier in the software development lifecycle.
- Creation of quality and risk benchmarks to identify code appropriate for reuse

Visit www.coverity.com for more information.

Testuff

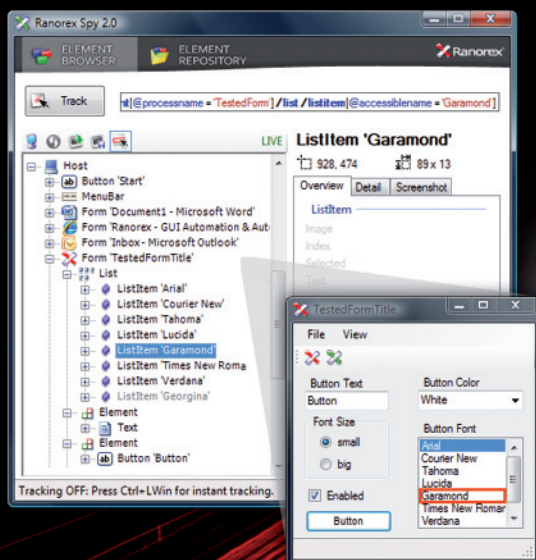
TEL AVIV, ISRAEL—Testuff is an on-demand service for managing and executing manual software tests and for reporting defects. It is a test management suite that includes management for various test stuff: cases, runs, defects, and more.

Here are some of the highlights:

- No more “works for me”—cap-

Ranorex 2.0

Next Generation Automated GUI Testing



- » Excellent Object Recognition
- » Object-based Capture/Replay Editor
- » Graphical Automation Editor
- » Professional Library for C#, VB.NET & Python



Download - Learn more
www.ranorex.com/trial



ture a video of each test and attach it to bug reports. Teststuff drastically reduces the time testers spend explaining the bugs to developers and helps developers quickly reproduce and eradicate elusive bugs.

- Doesn't require an IT department to install and maintain. Teststuff is an on-demand service with a desktop client application and a secure Web-based backend. It saves you both time and money, which can be better spent on improving your software.
- Integrates with many bug trackers and can automatically generate bug reports in your bug tracker. All reports include a link to the video recording of the test and a lot of additional useful information for reproducing and correcting bugs.
- Gives you peace of mind by storing your test data on secure and routinely backed up servers that are monitored around the clock.

Morph AppSpace

ATLANTA, GA—ProjectLocker has announced an integration with Morph AppSpace to enable developers to easily deploy applications directly to the cloud. The integration enables ProjectLocker users to quickly deploy their applications to Morph's high-quality production environment directly from within the ProjectLocker user portal, without the hassles of procuring, configuring, and managing the Web application environment. This integration now allows ProjectLocker to expand its service offering to encompass the deployment and management phase of the software development lifecycle. By providing a managed development environment along with a managed deployment environment, software teams can focus even more on using best practices to build their software without the expense in resources and man hours of building and maintaining their own solutions.

The initial integration enables Ruby on Rails projects to quickly deploy to the Morph AppSpace environment with a few simple steps. Support for Java and

Grails applications will follow. Morph Labs provides a free developers account while production quality environments including a high availability computing environment, automated backups, and 24/7 monitoring start at \$31 per month. The Morph AppSpace solution is built on top of the Amazon Web Services cloud computing environment including Amazon EC2 and Amazon S3.

Visit www.projectlocker.com for more information.



WANT TO RECEIVE COMPLIMENTARY COPIES OF SOME OF THE LATEST BOOKS ON SOFTWARE DEVELOPMENT?

Then you may be interested in the **StickyMinds.com Book Review Program!**

If you're an experienced software professional who likes to read and thrives on sharing opinions, join our unique book review program that caters exclusively to the software development community!

Tell us about your background, experiences, and which topics you'd like to review. If accepted into the program, you will receive a book selected for you to review—up to four a year. And the best part of the program? You keep the book—No Charge!

For an application or more information, contact Cheryl M. Burke, cburke@sqe.com.

Better Software Has Gone Digital!

Read your favorite magazine straight from your desktop before it hits the street—check out the digital edition online today at www.StickyMinds.com/DigitalPreview

Want to switch to the digital edition? Contact our subscription department at 1-800-450-7854 or info@BetterSoftware.com to find out how.



Things You Might Not Know About

Software Bugs

by Richard Bender

- 1 **THE FIRST COMPUTER BUG WAS SAVED** Most people know the story of Grace Hopper and how on September 9, 1945, at 3:45 p.m. she discovered that a moth caught in the wiring of the Mark II computer was causing the machine to behave erratically. Thereafter, every time the computer behaved in an unexpected manner, her group said there must be another bug. However, most people do not know that Grace taped the moth into her log, which can be seen at history.navy.mil/photos/pers-us/uspers-h/g-hoppr.htm.
- 2 **THE TERM "BUG" IN TECHNOLOGY PREDATES SOFTWARE** A newspaper report in 1889 stated that Thomas Edison "had been up the two previous nights searching for a bug in his phonograph." Even earlier, in 1878, Edison writes to a friend that "Bugs show themselves, and months and months of anxious watching, study, and labor are requisite before commercial success – or failure – is certainly reached." He sure sounds like a tester to me. Similarly, the term "debugging" predates the above mentioned moth.
- 3 **THE RATE OF RESIDUAL BUGS IS HIGH** The residual bug rate (i.e., the defects left in the software after the testing has been "completed") is five defects per thousand lines of executable code. If a high-end chip had this defect rate, it would be delivered with 5,000,000 defects.
- 4 **RESIDUAL BUGS ARE EXPENSIVE** In the US we spend about \$350 billion a year on software including development, maintenance, packages, etc. The direct cost of dealing with these bugs is about \$60 billion. The consequential costs exceed \$290 billion.
- 5 **ONE BUG CAN COST BILLIONS OF DOLLARS** A round-off error in calculating the market index for a Canadian stock exchange was not caught for twenty-two months. Since the calculation was performed 2,800 times a day, by the time it was caught it resulted in the index being miscalculated as half of what it should have been. This in turn drove the value of the stocks on that exchange down by a similar margin.
- 6 **SOFTWARE BUGS CAUSE AUTOMOBILE RECALLS** Hundreds of thousands of cars are recalled every year due to software bugs. Many of these are safety-critical issues such as the car's disagreeing with the driver about stopping when the brakes are applied.
- 7 **A BUG WAS FEATURED IN PEOPLE MAGAZINE** In 1985, software bugs in the Therac-25 radiation treatment machine caused three deaths due to overdoses of radiation.
- 8 **SOFTWARE BUGS CAN GET YOU ARRESTED** In 2004, a bug similar to no. 7 caused twelve patients in Panama to be overradiated. The developers and testers involved were arrested for murder by the Panamanian government, which also tried to extradite those who worked on the software in the US.
- 9 **SOME BUGS ARE THERE ON PURPOSE** In 1982, the CIA learned that the Soviets were trying to steal an oil pipeline process control system. So the CIA inserted a bug in it before letting the Soviets steal it. The result was the largest non-nuclear explosion ever seen up to that time.
- 10 **A BUG ALMOST DESTROYED THE WORLD** On September 26, 1983, a bug in a Soviet missile warning system made it appear that the US had launched a strike against them. A counterstrike to the phantom missiles was almost launched, which would have resulted in the destruction of a good portion of the planet. It was stopped by Lt. Col. Stanislav Petrov who refused to pass on the alert based on his hunch that it was a system bug. Lt. Col. Petrov is literally the man who saved the world.

This list is extracted from a planned book called the *Bender Book of Bugs*. Additional bug stories are welcomed. Please send them to rbender@BenderRBT.com.

The Abolition of Ignorance

by Alan Page

I've been a software tester for more than fifteen years, but I still remember when I realized I didn't really know anything about testing. It's not that I was a bad tester. I found a ton of bugs, I used automation and other test tools appropriately (most of the time), and I received great feedback from management. Despite these signs of success, three or so years into my testing career, I decided I would like to *study* testing. Since I was getting to a point where I thought testing may actually become a career for me, I decided I wanted to learn more about the hows and whys of testing and build a formal base of knowledge. I started by reading a book on testing that a colleague recommended. In many ways it opened my eyes. I felt that the knowledge I gained from this book, combined with my experience, would make me some sort of "super-tester" (it didn't). A few weeks later, flushed with success, I read another book on testing. I enjoyed reading the second one, too, but it stressed different practices and even contradicted the first book in places.

Now, I was confused.

By the time I completed a third book on testing, two things happened. The first was that I realized not every book on testing was very good at teaching testing. The second was that I began to form my own opinions on what works and what doesn't when testing software. I've read dozens of books and thousands of articles since then on testing and on software engineering, and all I've figured out from all this reading is that I've barely scratched the surface of what there is to know about testing.

I question much of what I read now—not necessarily as a skeptic who thinks that none of this stuff really works in practice but as a learner who wonders how the ideas might apply to situations I have experienced or could imagine. Theory isn't enough when you're trying to make a career out of testing software. So I try things. I experiment. The more I

learn through reading and practice the more I realize how much there is still to learn and how many undiscovered paths remain for me to explore.

Phillip Armour first wrote about the five orders of ignorance (5OI) in 2000 [1]. The 0th order of ignorance (the orders are 0 based, of course, because Armour is a programmer) is *lack of ignorance*. You have 0OI when you know something ("I know how to speak English"). The 1st order of ignorance is *lack of knowledge*. You have 1OI when you know you don't know something ("I know that I can't speak Chinese"). The 2nd order of ignorance is *lack of awareness*. You have 2OI when you are unaware of what you don't know. The 3rd order of ignorance is *lack of process*. You have 3OI when you don't even have a way to figure out what you don't know. The final level of ignorance—*meta ignorance* (4OI)—is when you don't know about the levels of ignorance (a level of ignorance that readers of this article can cross off now).

When I first heard of the five levels of ignorance, I realized that testing lives at 3OI and 2OI. Our job in examining software is to figure out what questions to ask about the software and then to determine what the answers are. Chances are that the programmers with whom you work don't write your test cases for you or suggest, for example, that you "try a really big number in this field." Instead, we testers examine the software, hoping to learn enough about it to ask that question, and subsequently, find the answer. Our job as testers isn't to find bugs; it's to find knowledge!

The more I think about this, the more concerned I get about testers who live entirely at 0OI and think they know all the answers—not in a snobby, know-

“Theory isn't enough
when you're trying to
make a career out of
testing software.”

it-all kind of way but in a way that relies only on the knowledge they have today in order to test each new piece of software that comes their way. The state of the art in test has not kept up with advances in product design and implementation because too many testers have stayed inside their 0OI comfort zone when their true place in

the food chain is 3OI. Too many testers think that they “get” testing or that they know “enough” to do the right thing. If you think you have testing all figured out, you are part of the problem!

The 3OI processes, of course, contribute new knowledge to software engineering, but the consummate 3OI process is the critical thinking of intelligent and experienced testers. Testers who can navigate the unknowns of 3OI will elevate the science and craft of testing to a level it deserves but too often does not enjoy. The greatest testers I know realize there is much more to learn about testing than they know today. They know that there are questions they haven't thought of yet and know that there will always be new means to discover these questions. They never assume that they know “enough” about testing to always make the best choices. Obviously, if you are reading this article, you have *some* interest in learning more about testing, but how often do you challenge yourself to discover something new about testing? Do you have a vision of what software testing can become and strive toward that vision? Or do you think that merely repeating and refining what you're already doing is enough?

A passion for learning drives the work of every great tester I know. What drives you? {end}

REFERENCES:

Armour, Phillip G. “The Five Orders of Ignorance.” *Communications of the ACM*, October 2000/Vol. 43, No. 10.

We deliver web applications

www.railsware.com/be-agile

railsware

Have you been wondering how you can get a daily dose of what's new and popular on StickyMinds.com and in *Better Software* magazine?

We've been reading your mind!

StickyMinds.com and *Better Software* magazine are now on **Twitter**. If you're already on **Twitter**, follow @StickyMinds for regular updates about weekly columns, news, discussion boards, eNewsletters, and more, as well as information about *Better Software* magazine articles and Software Quality Engineering conferences. If you don't have a **Twitter** account, you can follow our **Twitter** feed at www.twitter.com/StickyMinds.



tweet!
tweet!



Index to Advertisers

Agitar	www.agitar.com	Inside Back Cover
<i>Better Software Magazine</i>	www.BetterSoftware.com	45
Borland	www.borland.com/enterprisetalk	2
Hewlett-Packard	www.hp.com/go/software	20
Hewlett-Packard	www.hp.com/go/software	Back Cover
Railsware	www.railsware.com	48
Rally Software	www.rallydev.com/bsm	17
Ranorex	www.ranorex.com	44
Seapine	www.seapine.com	Inside Front Cover
Better Software Conference & EXPO 2009	www.sqe.com/BetterSoftwareConf	15
SQE Testing Certification Training	www.sqetraining.com/STF	27
STAREAST 2009	www.sqe.com/STAREAST	5
ThoughtWorks	www.thoughtworks.com	1

Display Advertising advertisingsales@sqe.com

All Other Inquiries info@bettersoftware.com

Better Software (USPS: 019-578, ISSN: 1532-3579) is published ten times per year. Subscription rate is US \$49 per year. A US \$35 shipping charge is incurred for all non-US addresses. Payments to Software Quality Engineering must be made in US funds drawn from a US bank. For more information, contact info@bettersoftware.com or call (800) 450-7854. Back issues may be purchased for \$15 per issue (plus shipping). Volume discounts available. Entire contents © 2008 by Software Quality Engineering (330 Corporate Way, Suite 300, Orange Park, FL 32073), unless otherwise noted on specific articles. The opinions expressed within the articles and contents herein do not necessarily express those of the publisher (Software Quality Engineering). All rights reserved. No material in this publication may be reproduced in any form without permission. Reprints of individual articles available. Call for details. Periodicals Postage paid in Orange Park, FL, and other mailing offices. POSTMASTER: Send address changes to Better Software, 330 Corporate Way, Suite 300, Orange Park, FL 32073, info@bettersoftware.com.

Trying to be **agile** when your Java code is **fragile**?



Feeling the pressure to release software faster?
Are you bringing new features to market as
quickly as your business demands?

If enhancing or extending your Java application
feels risky – if you need to be agile, but instead
find yourself hanging on by a thread – AgitarOne
can help.

With AgitarOne's powerful, automated unit
testing features you can create a safety net of
tests that detect changes, so you know instantly
when a new feature breaks something.

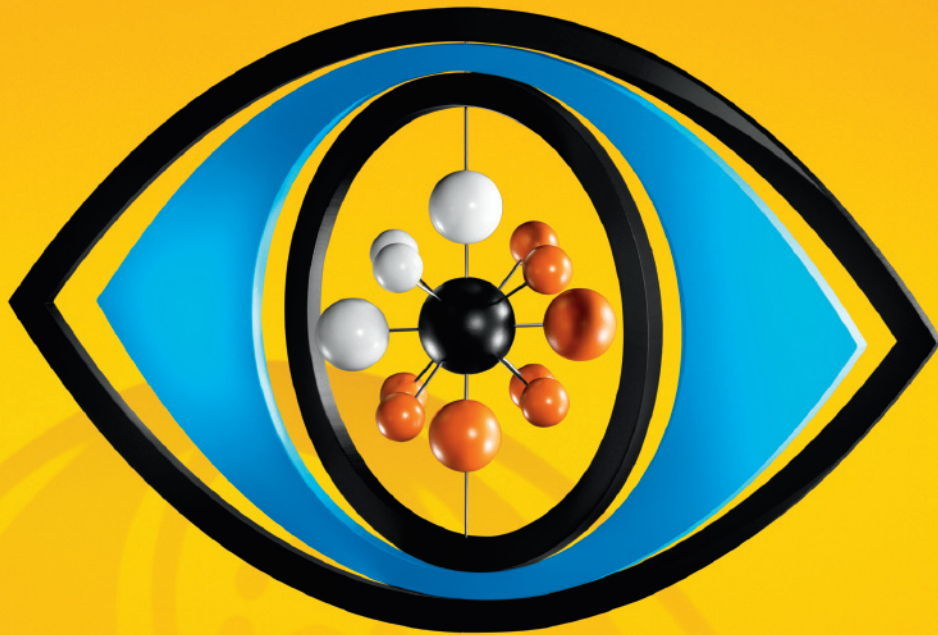
Now you can enhance and extend your Java
applications – fast and without fear.

AgitarOne's interactive capabilities for
exploratory testing makes it easy to test your
code as you write it.

Whether you're building a new application,
adding new features to existing software, or
simply chasing a bug, AgitarOne can help you
stay a jump ahead.



Visit www.agitar.com/bettersoftware to request a free evaluation.



ALTERNATIVE THINKING ABOUT QUALITY MANAGEMENT SOFTWARE:

Make Foresight 20/20.

Alternative thinking is "Pre." Precaution. Preparation. Prevention.
Predestined to send the competition home quivering.

It's proactively designing a way to ensure higher quality in your
applications to help you reach your business goals.

It's understanding and locking down requirements ahead of
time—because "Well, I guess we should've" just doesn't cut it.

It's quality management software designed to remove the
uncertainties and perils of deployments and upgrades, leaving
you free to come up with the next big thing.

Technology for better business outcomes. hp.com/go/quality





Software Tester Certification

Certified Tester—Foundation & Advanced Level Training



More than 90,000 Certified Testers Worldwide. Why Not You?

SPRING 2009 FOUNDATION LEVEL TRAINING*

Houston, TX	February 17–19, 2009
Toronto, ON	February 17–19, 2009
Mountain View, CA	February 24–26, 2009
Atlanta, GA	February 24–26, 2009
Los Angeles, CA	March 3–5, 2009
Tampa, FL	March 3–5, 2009
Raleigh, NC	March 10–12, 2009
Scottsdale, AZ	March 10–12, 2009
Nashville, TN	March 17–19, 2009
Vienna, VA	March 17–19, 2009
Boston, MA	March 23–25, 2009
Boise, ID	March 31–April 2, 2009
San Francisco, CA	April 14–16, 2009
New Jersey Area	April 14–16, 2009
San Diego, CA	April 20–22, 2009

Milwaukee, WI	April 21–23, 2009
Denver, CO	April 28–30, 2009
Bethesda, MD	April 28–30, 2009
Orlando, FL	May 3–5, 2009
Birmingham, AL	May 12–14, 2009
St. Louis, MO	May 12–14, 2009
Seattle, WA	May 19–21, 2009
Philadelphia, PA	May 19–21, 2009
Chicago, IL	June 1–3, 2009
Las Vegas, NV	June 7–9, 2009
Oakland, CA	June 9–11, 2009
Detroit/Ann Arbor, MI	June 9–11, 2009
Providence, RI	June 16–18, 2009
Albuquerque, NM	June 16–18, 2009
Portland, OR	June 16–18, 2009

www.sqetraining.com/certification

- **Basics of testing** – Goals and limits, risk analysis, prioritizing, completion criteria
- **Testing in software development** – Unit, integration, system, acceptance, and regression testing
- **Test management** – Strategies and planning, roles and responsibilities, defect tracking, and test deliverables



On-site Training Available—For additional savings, bring this course to your organization for team training.

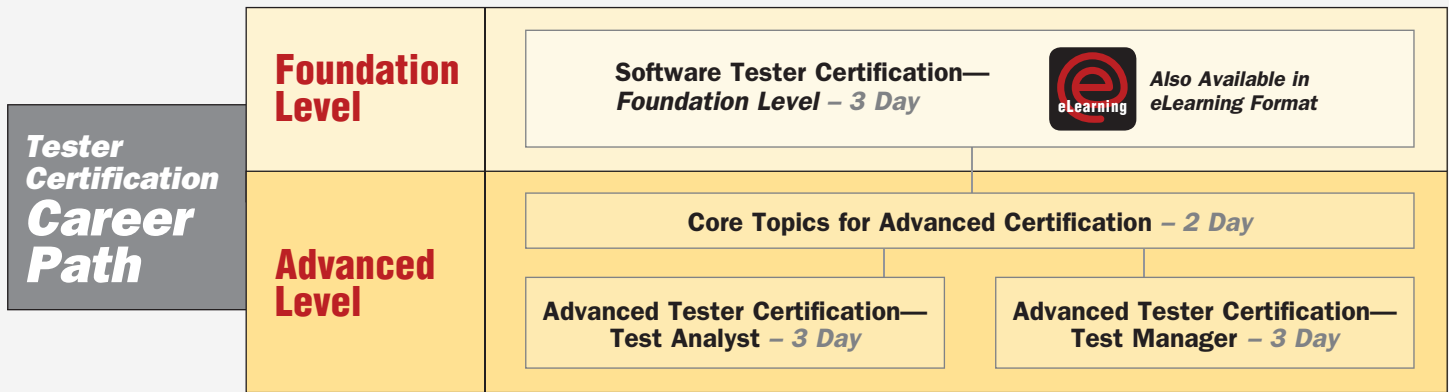
* See Inside for Advanced Level Training Schedule



Why is certification important for you?

Become a certified software tester and...

- Increase your value in both your organization and the industry
- Stand out from your peers with your professional certification
- Add value to your career path
- Get your test organization up to speed



Software Tester Certification—Foundation Level is an accredited training course, designed to help you prepare for the ISTQB certified tester—foundation level exam. This course is a prerequisite to the ISTQB™ advanced level certification. The advanced certification is divided into three main career tracks: Test Manager, Test Analyst and Technical Test Analyst. All of these certifications require a common set of skills. SQE Training has created a two-day “Core” course that covers the core topics for the advanced ISTQB certification. Additionally, we have created a three-day course that covers the topics in the advanced Test Analyst syllabus, and a three-day course to cover the topics in the advanced Test Management syllabus. Take either of these three-day courses in conjunction with the two-day “Core” to prepare you for either the advanced Test Analyst or Test Manager certification exam. For more information please contact our SQE Training Client Support Group at sqinfo@sqe.com or call 888.268.8770 or 904.278.0524.



eLearning — The eLearning version of the Software Tester Certification—Foundation Level training course is now available. This new, self-paced, and highly-interactive course offers the same expert instruction and valuable content that hundreds of organizations and thousands of testers have come to expect from SQE Training’s instructor-led classes. Based on the ISTQB foundation syllabus, the eLearning version of Software Tester Certification—Foundation Level allows testers to master new concepts and skills on their own schedule within the bounds of their current projects—all while avoiding the expense of travel.

Students experience a dynamic, asynchronous course developed by eLearning professionals with the advice of respected testing experts. Delivered in bite-sized lessons with clearly stated learning objectives and regular progress checks, this course offers a unique learning approach for software testers. For more information about eLearning please contact the SQE Training Client Support Group at sqinfo@sqe.com or call 888.268.8770 or 904.278.0524.

EASY TO REGISTER



Online:
www.sqetraining.com/register



Phone:
888.268.8770
904.278.0524



Email:
sqinfo@sqe.com

On-site Training

Looking for ways to save training and travel dollars? Take advantage of the cost-effective convenience of on-site training to get your team the training they need without requiring them to sacrifice project schedules or incur travel time and expense. Our on-site training offers many benefits:

- Save time and money—Bring team training to your location.
- Manageable workloads—Schedule training around your projects, not the other way around.
- Customizable content—Offer your team a training curriculum that adheres to your corporate goals, technology environment, and business needs.
- Consulting services—Learn from instructors who are world-class consultants with exceptional qualifications and a broad range of real-world experience. Augment your training programs with SQE Training’s consulting services.
- Small groups—Benefit from focused training that offers your team members individual attention with plenty of time for questions. Class sizes can range from 6 to 25 people.
- Employee development—Develop the talent already on your team, increase employee satisfaction—and save company dollars.

If you have six or more people to train, consider the advantages of on-site instruction.

For additional information, call 904.278.0524 or email onsitetraining@sqe.com.

WHO'S BEHIND THE TRAINING?

SQE Training provides the widest selection of specialized software training courses available. Developed and taught by top industry consultants, all courses are based on the latest industry practices and updated regularly to reflect current technologies, trends, and issues. Find the training you need for testing, development, management, requirements, and security. **www.sqetraining.com**



is affiliated with:



Topical Outline

Tester Certification—Foundation Level 3-day Course \$1,995 + \$250 for the ISTQB™ exam

Are you looking for an internationally recognized certification in software testing? Delivered by top experts in the testing industry, Software Testing Certification—Foundation Level is an accredited training course, designed to help prepare you for the ISTQB™ Certified Tester—Foundation Level exam. This certification program, accredited by the ISTQB™ through its network of National Boards, is the only internationally accepted certification for software testing. The ISTQB™, a non-proprietary and non-profit organization, has granted more than 90,000 certifications in more than thirty-two countries around the world.

Who Should Attend?

The Software Testing Certification—Foundation Level training course is most appropriate for individuals who recently have entered the testing field and for those currently seeking certification in software testing.

Course Guarantee

Course attendees who do not pass the exam within 60 days of completing the course, will be provided with 45 days of free access to our online eSoftware Tester Certification—Foundation Level course for additional learning.

Course Outline

Fundamentals of Software Testing

Software context, why software fails
Why testing is required?
Principles of testing
Debugging vs. testing
Scope and focus of testing
Understanding risk
Product (software) risks and project risks
Risk analysis, prioritizing using risk analysis
Goals of testing
Basic testing process
Test planning and control
Test analysis and design
Test implementation and execution
Evaluating exit criteria and reporting
Test closure activities
Test psychology—viewpoints on testing

Testing Throughout Software Development

Testing and development
Early testing
Models and testing
The “V” model
Iterative models
Verification and validation
Testing levels/stages within software development
Component (unit) testing
Integration testing
System testing
Acceptance testing
Maintenance testing
Understanding regression testing
Understanding test types
Functional testing
Non-functional testing
Structural testing
Confirmation and regression testing

You Will Learn

The Software Testing Certification—Foundation Level course covers the topics needed to prepare you for the ISTQB™ Certified Tester—Foundation Level exam:

- Fundamentals of software testing—Concepts and context, risk analysis, goals, process, and psychology
- Lifecycle testing—How testing relates to development including models, verification and validation, and types of tests
- Static testing—Reviews, inspections, and static tools
- Test design techniques—Black-box test methods, white-box techniques, error guessing, and exploratory testing
- Test management—Team organization, key roles and responsibilities, test strategy and planning, configuration management, defect classification and management
- Testing tools—Tool selection, benefits, risks, and classifications

Through the Software Testing Certification—Foundation Level training course, learn the basics needed to become a software test professional and understand how testing fits into software development. Find out what it takes to be a successful software test engineer and how testing can add significant value to software development. Study all of the basic aspects of software testing, including a comprehensive overview of tasks, methods, and techniques for effectively testing software. In addition, learn the fundamental steps in the testing process: planning, analysis, design, implementation, evaluation, and reporting.

Static Techniques

What is static testing?
Reviews, inspections, walkthroughs, etc.
General review process
Common types of reviews
Roles and responsibilities in reviews
Success factors for reviews
Limits of reviews
Understanding static analysis tools
Tool benefits

Test Design Techniques

Test design process
Overview of test design and the design approach
Documentation decision
Types and characteristics of documentation
Types of test design techniques
Black-box (functional) testing
White-box (structural) testing
Experienced-based techniques
Equivalence partitioning
Boundary analysis
Decision tables
State transition diagrams
Use cases and test design
Understanding control flow
Understanding paths and complexity
Coverage and what it means
Error guessing
Exploratory testing
Selecting the appropriate test technique

Test Management

Team organization
Roles and responsibilities
Understanding the test manager
Understanding the tester
Test planning
Planning and strategic thinking
Controlling the testing
Key strategic issues for test planning
Selecting a test approach
Understanding estimation
Test monitoring and reporting
Ending the testing, exit criteria
Configuration management and testing
Library controls
Change control
Defect/incident classification and management

Tool Support for Testing

Tool introduction
Tool selection process
Tool benefits
Tool risks and concerns
Tool classifications
Management tools
Static testing tools
Test specification tools
Test execution and logging tools
Performance and monitoring tools
Application area tools
Non-test specific tools



INSTRUCTOR

Dale Perry has more than thirty years of experience in information technology. He has been a programmer/analyst, database administrator, project manager, development manager, tester, and test manager. Dale's project experience includes large systems development and conversions, distributed systems, on-line applications, both client/server and Web based. He has also been a professional instructor for more than fifteen years and has presented at numerous industry conferences on development and testing. With Software Quality Engineering for eleven years, Dale has specialized in training and consulting on testing, inspections and reviews, and other testing and quality related topics.

Additional instructors for this course include Rick Craig, Conrad Fujimoto, Dawn Haynes, Claire Lohr, Jamie Mitchell, Gary Mogorodi, Eric Patel, Robert Sabourin, and Ed Weiler.

At the conclusion of the course, the ISTQB™ Certified Tester—Foundation Level exam will be given. The exam is held at 3:30 p.m. on the third day of the course. The ISTQB™ Certified Tester—Foundation Level certification exam is independently administered by the American Software Testing Qualifications Board, Inc. (ASTQB). To learn more about this certification program or to download the syllabus, visit www.astqb.org

Software Testing Certification

*Certified Tester—
Foundation & Advanced
Level Training*



www.sqetraining.com/certification

**Globally recognized
certification training
presented by
international experts**

ON-SITE TRAINING

For more information about on-site training courses, contact SQE Training at 904.278.0524 or 888.268.8770 or email onsitetraining@sqe.com.

EASY TO REGISTER



Online:
www.sqetraining.com/register



Phone:
888.268.8770
904.278.0524



Email:
sqeinfo@sqe.com

SOFTWARE TESTER CERTIFICATION

Foundation Level



eLearning

ADVANCED SOFTWARE TESTER CERTIFICATION

Core Topics for Advanced Certification

Advanced Tester Certification— Test Analyst

Advanced Tester Certification— Test Manager

ON-SITE TRAINING

Training Course Fee Includes*

- Tuition
- Course Notebook
- Continental Breakfasts and Refreshment Breaks
- Lunches
- Letter of Completion

*Your registration fee includes \$39 for a one-year digital subscription to Better Software magazine. If you are a current subscriber, your subscription will be extended an additional ten issues.

Training Course Schedule

7:30 a.m. - 8:30 a.m.	Registration (on first day) and Continental Breakfast
8:30 a.m. - 12:00 p.m.	Course
12:00 p.m. - 1:00 p.m.	Lunch
1:00 p.m. - 5:00 p.m.	Course
3:30 p.m. - 5:00 p.m.	Exam (on third day of course)

Satisfaction Guarantee: SQE Training is proud to offer a 100% satisfaction guarantee. It's part of our commitment to provide you with the highest quality education and training products. If we are unable to satisfy you, we will gladly refund your registration fee in full. Course attendees who do not pass the exam within 60 days of completing the course, will be provided with 45 days of free access to our online eSoftware Tester Certification—Foundation Level course for additional learning.

Public Training Policy: SQE Training reserves the right to make changes in course schedules, dates, locations, and accommodations and will make every effort to notify students within a reasonable period of time. However, SQE Training is not responsible for personal travel, accommodations, or other incidental expenses in connection with changes to a course. For questions or concerns, please call the Client Support Group at 888.268.8770 or 904.278.0524.

Cancellation Policy: Attendee substitutions are permitted. Registrants who fail to attend are subject to the full fee if they have not obtained a cancellation code from SQE Training at least six business days prior to the event start date. To obtain a cancellation code, call 904.278.0524 or 888.268.8770.

Register Early: The number of students per course is limited, and many courses fill to capacity. Register early to ensure your space in a given course.

Forms of Payment Accepted:

- Visa, MasterCard, or American Express
- Check or company purchase order is accepted. However, payment must be received before course registration is confirmed.

Confirmation: After payment, you will receive a confirmation notice containing course details (e.g., hotel, accommodations). Please bring the letter to the course for admittance.



www.SQETraining.com

330 Corporate Way
Suite 300
Orange Park, FL 32073

Presorted
Standard
U.S. Postage Paid
Gainesville, FL
Permit No. 726