



Build Better Software and Systems, and Improve Business Processes.

Strengthen your competitive advantage with Enterprise Lifecycle Management, an approach that aligns your technology investments with your business objectives by automating and integrating business processes, application lifecycle and development.

ELM enables you to align your technology investment with your enterprise objectives by automating and integrating:

BPO

Maximize the performance of your technology investments with Business Process Optimization (BPO). Manage your business processes, rapidly respond to marketplace change, eliminate wasteful business activity.

ALM

Improve the entire development lifecycle with Application Lifecycle Management (ALM). Increase productivity, develop innovative solutions, prove compliance.

MDD

Manage system complexity with Model Driven Development (MDD). Reduce development time, improve productivity and quality, lower project risk.

Strengthen your advantage with Enterprise Lifecycle Management. Learn more at www.telelogic.com/ELM

Telelogic

An IBM Company

May 2008

\$9.95 www.StickyMinds.com

BETTER SOFTWARE

CHAIN MAIL
OPTIONAL
Chivalry and
software
developers

TIRED OF
TESTING?
9 boredom-
busting heuristics

The Print Companion to StickyMinds.com

it's

a

BUG!

Testing
Lessons
from
Labor
Triage





PARASOFT®
We make software work.™

Concerned about the
security, reliability,
and compliance of
your SOA?

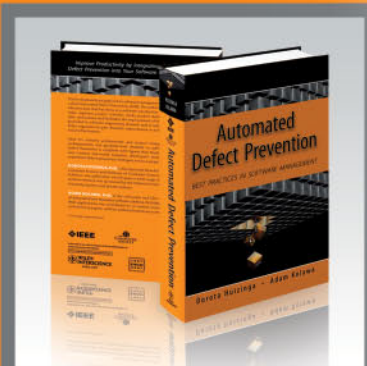
PARASOFT® SOA Quality Solutions

Parasoft SOA Quality Solution's
unique multi-layer approach delivers
a quality workflow that addresses
the architecture and automatically
prevents defects that can erode
confidence in your SOA.

Learn more.

www.parasoft.com/soaqualitysolution

Parasoft. Delivering quality as a continuous process.



***Want to
know more
about
Automated
Defect
Prevention?***

***We wrote
the book!***

Check it out.

www.parasoft.com/adp



Take the

handcuffs off

quality assurance

Empirix gives you the freedom to test *your way*.

Tired of being held captive by proprietary scripting? Empirix offers a suite of testing solutions that allow you to take your QA initiatives wherever you like.

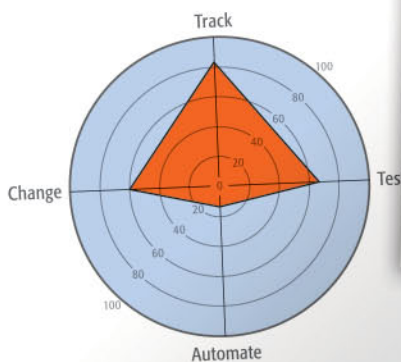
Download our white paper, "Lowering Switching Costs for Load Testing Software," and let Empirix set you free.

www.empirix.com/freedom

EMPIRIX
Be Confident.

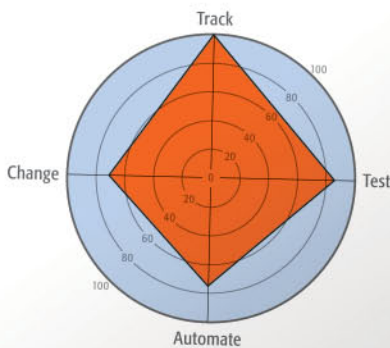
Seapine Software

Seapine Software Quality-Ready Assessment



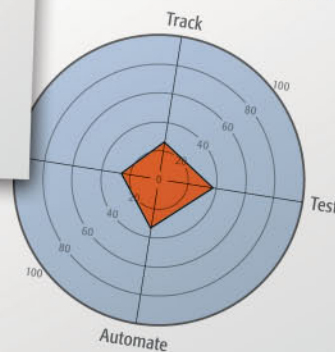
- Not prepared for an audit
- Little or no automation

Seapine Software Quality-Ready Assessment



- Audit-ready development artifacts
- Strong testing methodology and automation

Seapine Software Quality-Ready Assessment



- Lacking in most areas of quality
- Probably an "It's good enough" organization

What's Your Quality-Ready Score?

You need the right tools to build a quality-ready advantage. Take the Seapine Software Quality-Ready Assessment to measure your development and QA teams' ability to create a sustainable competitive advantage through integrated ALM and learn how to make your organization's productivity soar.

Find out your score at www.seapine.com/qra





Find the Bug
and win a **Wii™!**
Search through the digital
edition to find the **flying**
bug. Click it to be entered for a
chance to win a Nintendo Wii™.

Cover Story

IT'S A BUG! 26

Bug triage, like labor and delivery triage, is about deciding a course of action on the spot, often with minimal information to guide decision making. Discover what other lessons Robert has learned from Anne's experience in nursing that have practical applications in his hunt for bugs. *by Robert Sabourin and Anne Sabourin*

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.

THE CHIVALROUS TEAM MEMBER 32

Using the ten virtues described in Brian Price's modern code of chivalry, Martin and Mike illustrate the similarities between the best performing software team members of today and the Knights of the Round Table. *by Martin Kearns and Mike Cohn*

LET'S TALK AGILE 38

Agile development employs more oral communication, feedback, and interaction than traditional development. These communication tools can help ease the transition into the more interactive agile team relationship. *by Ken Pugh*

TECHNICALLY SPEAKING 13

What's the Deal with Investigators? • *by Lee Copeland*

"Investigators aren't sure" is a phrase that frequently pops up in the media. Information systems workers seem to share this uncertainty. So, what's the secret to success in this "aren't sure" world?

CODE CRAFT 17

The Accidental Complexity of Logic • *by Kevlin Henney*

Much code complexity and no small number of program defects can be traced back to confusion over logical expressions and the expression of logic. Find out how you can get that complexity under control.

TEST CONNECTION 20

Out of the Rut • *by Michael Bolton*

Do you feel as if all you do is repeat heavily scripted tests and as a result you aren't learning, discovering new problems, or finding bugs? These nine heuristics can help you get out of your rut and take back control of your testing process.

MANAGEMENT CHRONICLES 22

Communicate, Don't Assimilate • *by Melissa Sienkiewicz*

Opening an offshore office can be a tricky situation. Learn how to spread corporate values and processes to your new team members by working together instead of forcing them to adopt your way of thinking.

THE LAST WORD 47

When to Step Up, When to Step Back • *by Pollyanna Pixton*

It can be difficult to know when your team needs your help or when it needs more time to work out its own solution. How do you decide when to get involved?

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

June 2-6, 2008

Chicago, IL

September 8-12, 2008

New York/New Jersey Area

Agile Software Development

June 2-6, 2008

Seattle, WA

September 8-12, 2008

Washington, DC

SOFTWARE TESTING CERTIFICATION

www.sqetraining.com/certification

June 8-10, 2008

Las Vegas, NV

June 10-12, 2008

Bethesda, MD

June 17-19, 2008

Milwaukee, WI; Portland, OR; Ottawa, ON

CONFERENCES

Better Software Conference & EXPO 2008

www.sqe.com/bettersoftwareconf

June 9-12, 2008

The Venetian

Las Vegas, NV

STARWEST 2008

Software Testing Analysis & Review

www.sqe.com/starwest

September 29–October 3, 2008

Disneyland Hotel

Anaheim, CA

Agile Development Practices 2008

www.sqe.com/agiledevpractices

November 10-13, 2008

Shingle Creek Resort

Orlando, FL

BETTER SOFTWARE

Publisher

Wayne Middleton

Vice President of Publishing

Holly N. Bourquin

Editorial

Editor

Heather Shanholtzer

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

Production Coordinator

April Evans

Circulation and Marketing

Marketing Coordinator

Megan Brown

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

BETTER SOFTWARE

CONFERENCE & EXPO

JUNE 9-12, 2008
LAS VEGAS, NEVADA
THE VENETIAN

Agile Development ↑

Project Management ↑

People & Teams ↑

Testing & QA ↑

Requirements ↑

Process & Metrics ↑

Design & Architecture ↑

Conference Sponsor:



KEYNOTES BY INTERNATIONAL EXPERTS



Ken Schwaber
*Advanced Development
Methods, Inc.*



Michael Mah
QSM Associates



Jean Tabaka
*Rally Software
Development*



Johanna Rothman
*Rothman Consulting
Group, Inc.*



The Venetian

TUTORIALS WORKSHOPS CLASSES



Alan Shalloway	John Janakiraman
Andy Glover	Julie Gardiner
Andy Hunt	Ken Pugh
Andy Kaufman	Ken Schwaber
Ari Takanen	Kent McDonald
Beth Layman	Kevin Bodie
Bob Hartman	Lee Copeland
Chris Ronak	Lee Devin
Chuck Allison	Linda Rising
Dan North	Linda Westfall
David Garmus	Lisa Crispin
David Herron	Michael Mah
David Spann	Michele Sliger
Ed Weller	Mike Seavers
Elle Ringham	Mike Tholfsen
Guruprasad	Mitch Lacey
Gopalakrishnan	Nelson Perez
Herbert (Hugh)	Paco Hope
Thompson	Payson Hall
Hubert Smits	Pollyanna Pixton
James McCaffrey	Rebecca Wirfs-Brock
James Newkirk	Richard Bender
Jared Richardson	Rob Myers
Jean Tabaka	Robert Galen
Jeff Patton	Stacia Broderick
Jeff Payne	Tim Korson
Jerry Smith	Todd Little
Jimmy Xu	Vladimir Pavlov
Johanna Rothman	Will McKnight



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@sqa.com.



RICK CRAIG has a wide range of experiences as a tester and test manager. He is currently a consultant with SQE Training, specializing in metrics, management, and process improvement. Since 1984, Rick has spoken at conferences around the world including every STAR conference. He is the co-author of *Systematic Software Testing*. Rick is a colonel in the USMC Reserve.



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*.



MARTIN KEARNS is a certified Scrum coach and works for Renewtek, an Australian-based technical consultancy and project delivery specialist servicing Australasia. Responsible for designing training courses around Scrum and agile, Martin leads the promotion of agile principles within Renewtek and to its diverse client base. Martin also plays a continuing mentoring role to assist with follow-up implementation and agile retrospectives at key project milestones.



MELISSA SIENKIEWICZ is a project leader at Macadamian Technologies where she leads software teams, both local and remote, that work on technically complex projects. You can reach Melissa at melissa@macadamian.com.





MIKE COHN is the founder of Mountain Goat Software, a process and project management consultancy that specializes in helping companies adopt and improve the use of agile processes and techniques. He is the author of *Agile Estimating and Planning* and *User Stories Applied for Agile Software Development*. Mike is a founding member of the Agile Alliance, serves on its board of directors, and is a frequent presenter at the STAR and Better Software conferences. He can be reached at mike@mountaingoatsoftware.com.



POLLYANNA PIXTON developed collaboration tools for leaders through her thirty-eight years inside and consulting with corporations and organizations. She helps leaders create workplaces where talent and innovation is unleashed—making them more productive, efficient, and profitable while leading change in their marketplaces. She writes, teaches, and speaks internationally on collaborative leadership and business ethics. Currently, she is co-authoring a book entitled *Stand Back and Deliver: A Leader's Guide to the Agile Enterprise*. Pollyanna is a founding partner of Accelinova, president of Evolutionary Systems, and director of the Institute for Collaborative Leadership. You can contact Pollyanna at ppixton@accelinnova



KEN PUGH is a fellow consultant with Net Objectives. He consults, trains, mentors, and testifies on technology topics ranging from object-oriented design to Linux/Unix to system development processes. He has written several programming books, including the Jolt Award-winning, *Prefactoring*, and has served clients from London to Sydney. When not computing, Ken enjoys snowboarding, windsurfing, biking, and hiking the Appalachian Trail. Contact Ken at ken.pugh@netobjectives.com.



ANNE SABOURIN, BScN, RN, has been caring for newborn parents at the Royal Victoria Hospital for more than twenty years. A tenured delivery room and surgical baccalaureate nurse, Anne has been involved in some of the most important pre-term and multiple birth deliveries in the history of McGill Teaching Hospitals, including the successful birth of twins born thirty-nine days apart and children born at twenty-four weeks gestation. Anne pioneered telephone support for expectant parents in Quebec when she hosted Info Grossess, which was the two-year pilot project that led to the current Info Santé system in Quebec. Contact Anne at anne@theheart2heart.com.

Robert and Anne have developed Heart2Heart, a popular prenatal course for expectant couples. See www.theHeart2Heart.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

A mechanic's tool box is not complete without a socket wrench set. I couldn't imagine getting anything done on my car without it. My grip strength and knowledge of car mechanics alone isn't going to loosen a lug nut.

The first time I tuned up a car, I brought out the socket wrench set. With relative ease, I removed the air filter housing, the spark plug wires, distributor cap, and rotor button. But when it came to removing the spark plugs, I found out I was missing the right tool for the job: a spark plug socket wrench. Not wanting to double the work because of this mistake (reassemble everything, go to the store, disassemble everything, and then reassemble the new components), I opted to walk to the local automotive parts store. During that walk I vowed to make sure I always had the right tools in hand prior to tearing apart a car. I've stuck to that vow ever since.

It's been a while since I've had to work on my car, but I still have that tool set complete with a spark plug socket. There are people out there who've laughed at my story and said, "Why didn't you just use a large socket to loosen the spark plugs?" Yes, I could've used a socket wrench large enough to do just that job, but removing the plug would have required additional tools and would have added more steps to a simple process. Why go through all the extra hassle when there's a tool specifically designed for the job?

I think David Gilbert would have argued in defense of the spark plug socket in my scenario. In his article, "A Hammer and a Nail," David opens with the old axiom "If the only tool you have is a hammer, you tend to approach every problem as if it were a nail." While there's nothing wrong with the hammer, it's not always the right tool for every job. David reminds us that we have to consider the level of efficiency and accuracy we're trying to achieve to decide what tool would work best for a given scenario.

In software development terms, we spend a lot of money buying tools that automate specific tasks. Many times we're so invested in these tools that we try to use them for a variety of tasks. David reminds us that there are other tools in our tool box—basic ones that we sometimes forget, like manual testing.

Before you use a hammer to pound out your problems, stop and think of what you're trying to accomplish. Always keep in mind that there's a right tool for every situation; sometimes it's the one buried beneath all the new tools—one you probably forgot you had all along.

Read this month's Editor's Pick, "A Hammer and a Nail," at
www.StickyMinds.com/editorspick10-4



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

Quotables

"No good test—except one performed by an automaton—is entirely scripted, either.' Excellent quote to walk away with! As far as when to stop testing? I almost always respond with 'When I'm done.' Which then (hopefully) leads to 'And how do you know when you're done?' One item we track and share is the bug discovery rates while testing. How often are we finding new issues or, when we're validating fixes on past issues, do we discover side issues that arose from the fix? If the discovery rate keeps climbing—or even remains flat—it's still time to keep testing. Only when we see the discovery rate decline do we start getting comfortable; and this is usually a time when others' eyes are needed on the project."

STICKYMINDS.COM MEMBER BO ROOP COMMENTING ON
MICHAEL BOLTON'S ARTICLE "HOW MUCH IS ENOUGH?"
www.stickyminds.com/quotables10-4a

"As tempting as it is to dismiss the need for testing COTS applications, it's just not that simple. Take the time to analyze the application so you can balance the cost of testing against the potential risk and cost of failure. And then, after you test, all there's left to do is hope that it works."

LINDA HAYES, "TESTING COTS:
WHEN, HOW, AND HOW MUCH?"
www.stickyminds.com/quotables10-4b

"A tester on an agile team may test a screen that's half finished, missing some validation, or missing some fields. It's incomplete—only one part of the software—but testing it in this incomplete stage helps reduce the risk of failures downstream. It's not about certifying that the software is done, complete, or fit to ship. To do that, we'd need to drive the 'whole car,' which we'll do when the whole car is complete."

JEFF PATTON, "AN UNCOMFORTABLE TRUTH
ABOUT AGILE TESTING"
www.stickyminds.com/quotables10-4c

"I was once told by a manager that you don't learn from good projects, and although that is not strictly true, you do learn more from the bad ones."

STICKYMINDS.COM MEMBER MEREDYTH MACKAY
COMMENTING ON FIONA CHARLES'
"PACK UP YOUR TROUBLES"
www.stickyminds.com/quotables10-4d

QA/IT professionals: do you know about QAZone?



This new online community from
Empirix is exclusively for you!

To start collaborating with industry peers, register for a
free QAZone account now at: <http://qazone.empirix.com>.

QA/IT professionals: do you know about QAZone?



This new online community from
Empirix is exclusively for you!

To start collaborating with industry peers, register for a
free QAZone account now at: <http://qazone.empirix.com>.

Pair Programming Observations

By JEFF LANGR

Say “pair programming” to a programmer and he’ll probably frown or turn his back on you. But add some rules the programmers must follow—rules that help maintain each person’s sanity—and he just might come to find this practice rewarding and beneficial. This article, reprinted from Jeff Langr’s Web site, explains the rules and how certain teams have reacted to this structured version of pair programming.

www.stickyminds.com/eletterpick10-4a

The Top 3 Myths of Requirements Management

By JOHN SIMPSON/ERIC WINQUIST

This paper challenges the status quo and explores the top myths that continue to exist in the world of requirements management. Even though requirements management has been around for decades, the fact remains that four out of five technology projects and new products developed don’t succeed, which is why it continues to be a major juggernaut and point of frustration for companies. This white paper contains advice for how to dispel the myths and bridge the requirements management gap.

www.stickyminds.com/eletterpick10-4b

Questions You Should Ask

By MICHELE SLIGER

It’s a technique children and teenagers have mastered: asking “why” until they get to an acceptable response (or until we’re too tired to continue answering). In this column, find out how Michele Sliger uses a similar approach designed by Six Sigma to drill down into the underlying cause of any problem within software projects. She then continues the inquisition with a series of other questions to find out how these problems affect business value and technology. Read this article to learn what these questions are and how you can start using them to find out why things aren’t going as planned.

www.stickyminds.com/eletterpick10-4c

A sampling of content from our eNewsletter archives

Sticky ToolLook: November 8, 2007

A Word with the Wise: Thinking Tools with Clarke Ching

by Joseph McAllister

Clarke Ching is an agile advocate living in Scotland. He has an MBA specializing in technology management and is currently writing a book, *Rolling Rocks Downhill*, about how to make managing software projects easier, more productive, and predictable. Some of the tools he uses most often in his work include software like Microsoft Word, Excel, Gmail, and Typepad, as well as his BlackBerry for email. But his two most useful tools, he says, are “thinking tools.”

Clarke Ching: Both of them are from Eli Goldratt’s Theory of Constraints. The first of these is called the “Conflict Cloud.” It is a simple way of drawing problems using five boxes. I used it yesterday, for instance, to walk two of my client’s managers through a problem with how they managed projects. Sales would quote prices to their customers based on optimistic estimates. Why? Because they wanted to make the sale. But this annoyed the development managers because they found themselves committed to aggressive schedules with low probabilities of success. The conflict cloud took about fifteen minutes to draw, and within another ten minutes we had identified a number of areas we need to tackle to evaporate the problem. One, for instance, was that the sales people are rewarded for revenue targets, not profit targets.

The second tool is to always look at problems by asking first: Where is the bottleneck? In any system the bottleneck determines the output of the system. The only way to increase the output of the system is to improve the output of the bottleneck. Say that a development team identifies its developers as the team’s bottleneck. Imagine that it then asks the team—not just the developers—“what can we do to squeeze more work out of the developers?” Every time I’ve done this we’ve found simple ways (“less meetings” is a common suggestion) to free up time so that the whole team can get more work done. It isn’t as easy to spot the bottleneck when you’re dealing with knowledge workers, but when you do, it’s usually very easy to improve their output by, say, 20 percent, which results in an increase in throughput from the entire system by 20 percent.

JM: How did you first decide to incorporate these tools into your work? Have you introduced these methods to other people?

CC: I discovered these tools over ten years ago when I first read Eli Goldratt’s business novel, *The Goal*. It took me a couple of years before I used the bottleneck idea in practice and another five years before I started using the conflict cloud. I’ve had a lot of success with both, but I find other people are happy enough for me to help them to solve problems using the tools, but a little reluctant to use them on their own.

JM: Perhaps there’s something to a person with a bank of useful knowledge being a tool of sorts?

CC: That’s an interesting way of looking at it. I suppose that as a consultant people “rent” me because I have specialized knowledge and skills, just like I rented a van when I was moving house. Never thought of it like that before.

JM: Will you be incorporating thinking tools like the ones you mentioned into your own book, *Rolling Rocks Downhill*?

CC: Yes. My book is a business novel like Goldratt’s *The Goal*, and I use the cloud either explicitly or more subtly within most chapters. Novels are about solving crises and problems, and that is what the conflict is about. The bottleneck tool features throughout. In fact, my next StickyMinds article is about bottlenecks.

Just thought of another tool. If I am trying to solve a particularly messy problem, then often I will fire up my word processor, give the problem to one of the characters in my book, and let the other character help him out. It’s surprising what they know and what they can figure out. All I have to do is sit back and type what they want to say. They’re very lucky that they know and trust each other well enough to be able to have fierce arguments but still respect each other. I edit out the swearing during the rewrite.

Book Review

Agile Java Development with Spring, Hibernate and Eclipse

Author: Anil Hemrajani

Reviewed by Rahul Khanna

sectorxii@hotmail.com

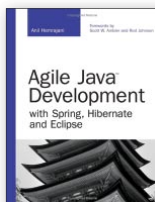
Anil has more than twenty years of experience and offers invaluable insight on agile model-driven development (AMDD), Extreme Programming (XP), Spring, Hibernate, Java, and Eclipse from a developer's perspective. And if these points won't convince you to check this book out, endorsements in terms of forewords on Anil Hemrajani's "brain dump" by the likes of Scott W. Ambler (agile) and Rod Johnson (Spring) should!

This book is an instruction manual on how to develop extensible and scalable applications quickly, with high quality. The author introduces proven concepts and technologies that are widely used throughout various industries, such as B-to-B, B-to-C, finance, healthcare, government systems, etc.

This book is organized to introduce concepts and build upon each topic. The structure of the text also provides a reference-like orientation for the experienced or veteran specialist. The author makes available (and keeps up to date) the book's example application on his Web site, www.visualpatterns.com. Readers can take advantage of the source code provided to rapidly build and deliver Web-based systems immediately. He also provides more than ample references, ranging from home bases for technologies (agile/XP, Spring, etc.) to supplemental tools (JUnit, MDA, test-driven methodology).

This book reads as if Hemrajani is sitting with the reader, sharing his knowledge and experience on agile Java development. Detailed introspection of Ant scripts, object/relational mappings (Hibernate), and the MVC layout reveals much to be appreciated that has been abstracted in these open source projects and toolkits. Even the cartoons in each chapter depict real-world scenarios, tried and proven.

All in all, this book provides an excellent reference to building Java applications with available, proven, and open source technologies. A must own!



Agile Java Development with Spring, Hibernate and Eclipse

By: Anil Hemrajani

Pages: 360

Publisher: SAMS Publishing

Published: 2006

ISBN: 0672328968

Description:

Agile Java Development with Spring, Hibernate and Eclipse is a book about robust technologies and effective methods that help bring simplicity back into the world of enterprise Java development. The three key technologies covered in this book—Spring, Hibernate, and Eclipse—help reduce the complexity of enterprise Java development significantly. Furthermore, these technologies enable plain old Java objects to be deployed in light weight containers versus heavy-handed remote objects that require heavy EJB containers.

This book extensively covers technologies such as Ant, JUnit, and JSP tag libraries and touches upon other areas such as logging, GUI-based debugging, monitoring using JMX, job scheduling, emailing, and more. Extreme Programming, agile model-driven development, and refactoring are methods that can expedite the software development projects by reducing the amount of up-front requirements and design; thus, these methods are embedded throughout the book but with just enough details and examples not to sidetrack the focus of this book. In addition, this book contains well separated, subjective material (opinion sidebars), comic illustrations, tips, and tricks, all of which provide real-world and practical perspectives on relevant topics.

Last but not least, this book demonstrates the complete lifecycle by building and following a sample application, chapter by chapter, from conceptualization to production, using the technology and processes covered in this book.

By using the technologies and methods covered in this book, the reader will be able to effectively develop enterprise-class Java applications in an agile manner!

Have you read this book?

Visit www.StickyMinds.com/

bythebook10-4 to post your comments,

or email editors@bettersoftware.com for

information on how you can have a book

review considered for publication on

StickyMinds.com.

Top

Most Read on

StickyMinds.com

Don't be the last one in the know. Read five of the most popular articles on StickyMinds.com from recent weeks.

11 An Uncomfortable Truth about Agile Testing

By Jeff Patton

www.stickyminds.com/10-4Top1

21 Lessons Learned about Starting a Development Group in India: Part Two

By Peter Clark

www.stickyminds.com/10-4top2

31 Planning the Endgame

By Fiona Charles

www.stickyminds.com/10-4Top3

41 Testing COTS: When, How, and How Much?

By Linda Hayes

www.stickyminds.com/10-4Top4

51 Making Sense of Root Cause Analysis

By Ed Weller

www.stickyminds.com/10-4Top5

WEB

SEMINARS

Special Announcements

StickyMinds.com and *Better Software* magazine Presents Web Seminars: Learning by the Hour

Do you want to learn more about the latest trends and techniques in testing and project management, but don't have much time to spare? Then our Web seminar program is just for you! This May, we'll broadcast two Web seminars from Cognizant and Telelogic. Check out the current Web seminar schedule at www.stickyminds.com/10-4WebSeminars.

If you cannot make a live performance, no worries! Web seminars are archived for your convenience. That means you can watch these seminars whenever you want. A list of archived presentations can be found on www.stickyminds.com/WebSeminarArchives.

POWERPASS POINTER

Branch Out Using Classification Trees for Test Case Design

By JULIE GARDINER

Classification trees are a structured, visual approach to identifying and categorizing equivalence partitions for test objects to document test requirements so that anyone can understand them and quickly build test cases. In this session presented at the 2007 STAR WEST conference, Julie Gardiner covered the fundamentals of classification trees and how they can be applied in both traditional test and development environments. Using examples, Julie shows you how to use the classification tree technique, how it complements other testing techniques, and its value at every stage of testing. She demonstrates a classification tree editor that is one of the free, commercial tools now available to aid in building, maintaining, and displaying classification trees.

- Develop classification trees for test objects
- Understand the benefits and rewards of using classification trees
- Know when and when not to use classification trees

www.StickyMinds.com/powerpasspointer10-4

WINNERS!

Jolt Product Excellence Award
2006 + 2008 + 2007



Sign up for FREE Community Edition

Rally's
award-winning
Agile life cycle
management
tool for a
single team!



Scaling Software Agility

www.rallydev.com/bsm



NEW

**Fall 2008
Schedule!**

*Accelerate Your Career
& Empower Your Team*

SOFTWARE TESTING TRAINING

BUILD-YOUR-OWN TRAINING WEEK

Maximize the impact of your training by combining courses in one location to create a customized training week. Pair two courses and save up to \$300. For a complete list of courses available, visit www.sqetraining.com or call 888.268.8770 or 904.278.0524 for pairing discount options.



www.sqetraining.com

Improve your skills and help your organization increase its performance through targeted high-value training. Delivered by top software consultants, training through SQE Training is one of the best investments you can make to meet your business objectives.

TRAINING WEEK LOCATIONS

September 8-12, 2008	<i>New York/New Jersey Area</i>
September 15-19, 2008	<i>Washington, DC</i>
October 20-24, 2008	<i>San Francisco, CA</i>
November 17-21, 2008	<i>Tampa, FL</i>



The following courses are PMI Certified:

- Software Testing Certification
- Systematic Software Testing
- Mastering Test Design
- Test Management

Choose from 12 specialized training courses:

THREE-DAY COURSES (Monday - Wednesday)

- Systematic Software Testing
- Software Testing Certification
- Test Management
- Writing Testable Requirements
- Just-in-Time Software Testing

TWO-DAY COURSES (Thursday - Friday)

- Mastering Test Design
- Performance, Load, and Stress Testing
- Lean-Agile Testing Practices
- Requirements Based Testing
- Software Security Testing and Quality Assurance
- Exploratory Testing in Practice

ONE-DAY COURSE (Thursday)

- Test Process Improvement



Let SQE Training come to you. For more information about on-site training courses, contact SQE Training at 904.278.0524 x212 or 888.268.8770 or email onsitetraining@sqe.com.

What's the Deal with Investigators?

by Lee Copeland

Recently, a British Airways B777 crashed just short of the runway at Heathrow Airport near London. Due to the skills of the pilots and the proximity to the airport, tragedy was averted. The British Air Accident Investigation Branch announced, “The autothrottle demanded an increase in thrust from the two engines but the engines did not respond.” The announcement continued, “Investigators aren’t sure of the reason.”

As I read, “Investigators aren’t sure,” the phrase echoed in my brain. Haven’t I read that before? So I Googled “investigators aren’t sure” and got more than 3,300 hits. Examples include:

- Investigators aren’t sure what instigated an attack.
- Investigators aren’t sure whether the fire was set intentionally.
- Investigators aren’t sure how long the body was in the area.
- Investigators aren’t sure if the same crew is responsible for all of the break-ins.
- Investigators aren’t sure what kind of gun was used.
- Investigators aren’t sure how long they will look for evidence.
- Investigators aren’t sure what she (Britney) is wearing now.
- Investigators aren’t sure if that’s the whole story.
- Investigators aren’t sure they believe it either.

Now, this “Investigator” job sounds like it’s right up

my alley. Apparently you don’t have to be sure of anything and the pay is good. But as I was reading the Google search results, it occurred to me that this sounds a lot like information systems work. In our business, we are the investigators, and we aren’t sure of a number of things:

- Business analysts aren’t sure they understand the totality of what the stakeholders expect the system to do. In fact, they aren’t sure the stakeholders understand the totality of what the stakeholders expect the system to do.
- Stakeholders aren’t sure business analysts understand enough of the details of their business to really understand the details.
- Developers aren’t sure why they’re required to write code before the requirements are complete, correct, and consistent. They’re not sure why those least qualified to make technical decisions have the authority to do so.
- Testers aren’t sure about either the documented requirements or the system under test. They aren’t sure what they can safely assume about the requirements. They aren’t sure which are the most important parts of a system to test. They aren’t sure if they have created test cases with the capacity to detect hidden defects. They aren’t sure if they have created sufficiently varied data. They aren’t

sure why all the people on the front end of the project get paid more than they do.

- Project managers aren’t sure if they’ll be given the resources they really need to carry their project to a successful conclusion.

“We live in an ‘aren’t sure’ world, so I’m constantly surprised at how many people are ‘totally sure’ of their positions on anything and everything.”

- All aren’t sure why the others don’t recognize the intellectual creativity and prowess required to do what they do.

We live in an “aren’t sure” world, so I’m constantly surprised at how many people are “totally sure” of their positions on anything and everything. In software development, people are sure of this elicitation process, this requirements notation, this programming language, this test technique, and this project management approach.

Giving us fair warning about our “aren’t sure” world, Mark Twain wrote, “It ain’t what people don’t know that hurts them, it’s what they know that ain’t so.”

But “aren’t sure” applies even to this oft-quoted

saying—some attribute it instead to Josh Billings [1] while others claim it was Will Rogers [2] who penned it.

So what’s the secret to success in an “aren’t sure” world? It’s no secret. It’s the Plan-Do-Check-Act (PDCA) process cycle invented by Walter Shewart, popularized by W. Edwards Deming, and incorporated into every iterative/incremental software development process model today [3].

Plan and do are ubiquitous. Everyone does that part. It’s the check the results of your planning and doing for effectiveness and then the act of improving the process that is the magic. Measure your results objectively. Set aside what is not effective in favor of something else. Try again, repeating the PDCA cycle.

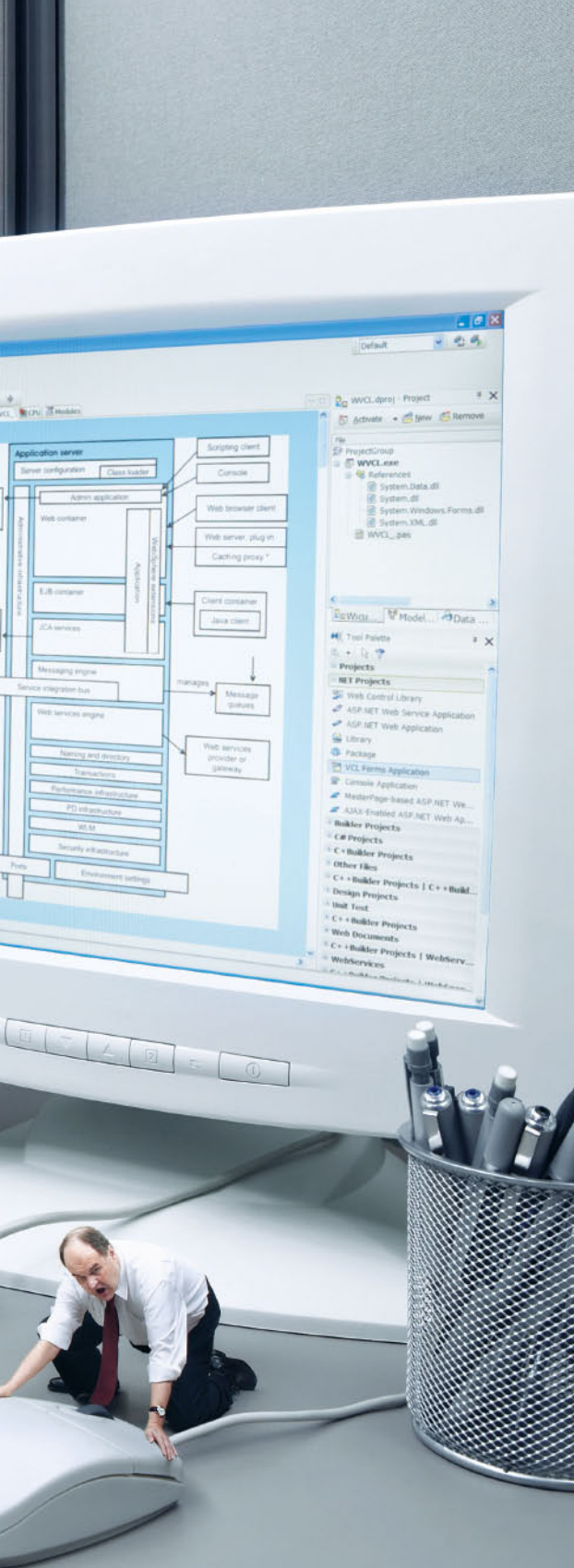
With regard to acceptance of new ideas in physics, Max Planck wrote, “A new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it.”

I hope that we as software development professionals we do not fight against improvements in the same manner. Remember, what we set aside is not failure; it is learning and improving. And that’s what good Investigators do. {end}

REFERENCES:

- [1] en.wikipedia.org/wiki/Josh_Billings
- [2] www.plumbingsupply.com/quotes-ourteamfavorites.html
- [3] en.wikipedia.org/wiki/PDCA





Rational

_INFRASTRUCTURE LOG

_DAY 90: Our software engineers are completely overwhelmed by this system we're building. It's so complex. The deadline is tight. Performance and quality can't be anything less than perfect. Everything depends on the software we're developing. It's all so...I don't know...huge.

_DAY 92: The team is tired of typing one key at a time.

_DAY 93: I've taken back control with IBM Rational Systems Development solutions. They'll give us the tools we need to manage our complex systems development through the entire lifecycle, from inception to deployment. Their disciplined approach looks at the full range of processes holistically.

_We're all back to normal size again. Except Gil's brain. Which is still a bit on the tiny side.

Download the Systems Development e-Kit at:
IBM.COM/TAKEBACKCONTROL/SYSTEMS



SOFTWARE TESTING ANALYSIS & REVIEW

The Greatest Software Testing Conference on Earth

Sept. 29–Oct. 3, 2008 • The Disneyland® Hotel



**99.7% of 2007 Attendees
Recommend STARWEST
to Others in the Industry**

**www.sqe.com/starwest
REGISTER EARLY AND SAVE \$200!**



The Accidental Complexity of Logic

by Kevlin Henney

Logic lies at the heart of computation. Boolean algebra guides much of the course of a program's flow. It is the stuff of programming. It is also, as Kernighan and Plauger noted in *The Elements of Programming Style*, the source of occasional confusion: "Boolean algebra is not used nearly as widely as ordinary arithmetic, so we must write logical expressions more carefully lest we confuse the reader." They wrote this thirty years ago, but our ability to be confused has not lessened any in that time.

Much code complexity, many programming thinkos, and no small number of program defects can be traced back to confusion over logical expressions and the expression of logic. In spite of our best aspirations and the relative simplicity of Boolean algebra, this is not the kind of thinking that comes instinctively to humans—even programmers.

In many ways this state of affairs can be considered ironic. In formulating the calculus that bears his name, George Boole was pursuing an ambitious goal, as indicated in the title and opening of his 1853 work, *An Investigation into the Laws of Thought*: "The design of the following treatise is to investigate the fundamental laws of those operations of the mind by which reasoning is performed."

In practice, although we use Boolean logic correctly and effectively much of the time, the shortfall between the way we think and the needs of code presents sufficient opportunity for incorrect and unruly code to creep in.

Dead Code Walking

Consider the fragment in listing 1a, which is based on some code I saw a few years ago.

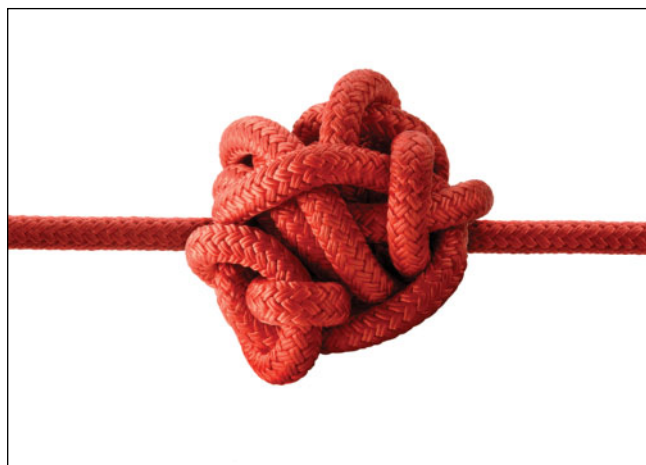
```
if(collection.isEmpty())
{
    for(Item item : collection)
    {
        ... // about 30 lines of code
    }
    ... // about 20 lines of code
}
```

Listing 1a

Because the body of the loop is unreachable, this becomes a rather longwinded way of writing the fragment in listing 1b.

```
if(collection.isEmpty())
{
    ... // about 20 lines of code
}
```

Listing 1b



ISTOCKPHOTO

Given the chaos and low quality of the project as a whole, it is exceedingly unlikely that the illusory, lines-of-code productivity boost that the dead code gives was intentional.

Omitting a *not*, or including a surplus one, or using an *or* where an *and* was meant, or vice versa, are common examples of errors that take code down the wrong path. Unit tests can be a great help in such situations: Staring at a piece of code for too long, you may possibly fool yourself into thinking that your inverted logic is correct, but a simple assertion is less easily duped. Of course, in the project in question there was no coherent form of testing—manual or otherwise—at any level, and because of the application's high coupling, unit tests would have been pretty much impossible. Code review is a complementary way of uncovering such defects and would have been a suitable alternative. Code reviews do not have to be formal, and an informal pair walkthrough is often less time consuming and less threatening than walking through code in a meeting with many others.

```
if(distance > recommendedMaxDistance &&
    userSelectedOverride &&
    measuredValue >= minimumValue &&
    measuredValue <= maximumValue &&
    distance < recommendedMaxDistance)
{
    result = false;
}
else if(!userSelectedOverride)
{
    result = true;
}
else
{
    result = false;
}
```

Listing 2a

Seeing the Forest for the Trees

Simple thinkos are one source of error, but another comes from accidental complexity—making things more complex than necessary so that it becomes difficult to see what is going on. Consider the fragment in listing 2a, which is based on code I ran across (well, tripped over) many years ago.

The code didn't start out this way. It probably made sense originally—and even after a change or two—but eventually an addition led to a contradiction, which means the code reduces to the snippet in listing 2b.

```
if(!userSelectedOverride)
{
    result = true;
}
else
{
    result = false;
}
```

Listing 2b

Or, more concisely, to the code in listing 2c.

```
result = !userSelectedOverride;
```

Listing 2c

Signal to Noise Ratio

But unclear logic is not just about defects and dead code. Code can be functionally correct but developmentally poor. Clumsy logic also introduces accidental complexity, making the process of reading code more labored than it should be. The code in listing 3a includes a redundant check.

```
if(!file.exists() ||
    (file.exists() && !file.hasWritePermission()))
    return false;
```

Listing 3a

In this particular case, another programmer suggested the “simplification” shown in listing 3b, but it is not exactly an improvement.

```
if(!file.exists())
    return false;
else
    if(!file.hasWritePermission())
        return false;
```

Listing 3b

Instead, for such a simple case, group conditions that have a common result, as shown in listing 3c.

```
if(!file.exists() || !file.hasWritePermission())
    return false;
```

Listing 3c

Should the condition become longer than is either reasonable

or readable, extract and name a method that represents the whole condition rather than disassembling the logic into an `if` else tree.

Boolean-to-Boolean Converters

One particularly common and noisy kind of construct is what a conference attendee I once met referred to as the “Boolean-to-Boolean converter,” a name that nicely sums up the redundancy of such constructs. These are often characterized by extensive use of `true` and `false` literals and control flow structures. They all can be reduced to much simpler expressions. Here is a typical example of an identity converter:

```
if(found)
    return true;
else
    return false;
```

Which is nothing more than:

```
return found;
```

And here is an elaborate negation:

```
if(enabled)
    enabled = false;
else
    enabled = true;
```

Which is sometimes written slightly more compactly as:

```
enabled = enabled ? false : true;
```

This still reduces to something simpler and more direct:

```
enabled = !enabled;
```

Listing 4a is a slightly more nested example.

```
if(isOldest)
{
    if(timeLastAccessed < cutOffTime)
    {
        return true;
    }
    else
    {
        return false;
    }
}
else
{
    return false;
}
```

Listing 4a

Its jagged formation still reduces to something briefer and more direct, as shown in listing 4b.

```
return isOldest && timeLastAccessed < cutOffTime;
```

Listing 4b

Sometimes a truth is established on one line, only to be restated using literals in repetitive form across subsequent lines as shown in listing 5a.

Listing 5b demonstrates that both the decision structure and the use of the Boolean literals are unnecessary.


```
if(value > threshold)
    markOutOfLimit(true, timeStamp());
else
    markOutOfLimit(false, timeStamp());
```

Listing 5a

```
markOutOfLimit(value > threshold, timeStamp());
```

Listing 5b

Perhaps one of the most pervasive examples of a Boolean-to-Boolean converter is the explicit comparison of Boolean results against Boolean literals:

```
if(checkCacheExists() == true)
    ...
```

The result of explicitly comparing a Boolean against true makes it no truer than it already was. Such tautological phrasing is often a consequence (or a cause) of imperative names that should follow a more predicate-like naming style. Rather than naming the action, name the truth that is being acted on:

```
if(cacheExists())
    ...
```

Names beginning with *check*, *validate*, and *verify* are typical candidates for such renaming.

Flag Waiving

Overreliance on flags leads to code with a lot of raw Boolean literals—often an indicator that logic can be revisited and simplified. Consider the example in listing 6a.

```
boolean failed = false;
if(message != null)
{
    if(!enqueue(message))
        failed = true;
}
else
    failed = true;
if(failed)
    throw new MessageEnqueueException();
```

Listing 6a

```
if(message == null || !enqueue(message))
    throw new MessageEnqueueException();
```

Listing 6b

```
boolean result;
int indexOfNext;
indexOfNext = mapping.nextAvailable();
if(indexOfNext == INVALID_INDEX)
    result = true;
else
    result = false;
return result;
```

Listing 7a

It simplifies to listing 6b.

Similarly, listing 7a represents an impressive attempt to mask a one liner.

```
return mapping.nextAvailable() == INVALID_INDEX;
```

Listing 7b

The separation of declaration from initialization introduces more noise on top of the flag-driven logic. From all this we can extract the delightfully brief listing 7b.

Conclusion

In each of these cases it is worth keeping in mind that it is not necessarily ignorance or sloppiness that has led to unnecessary or incorrect logic, and it has nothing to do with the programming language—the examples may have been presented in Java, but they were based on published and production code in a variety of languages. Nevertheless, to paraphrase the author Martin Amis, it is terrible to see a programmer being beaten up by a programming language. Revisit, review, and revise. Truth (or falsehood) will out. **{end}**

What examples of logic have you seen that, on revisiting, have proven to be either incorrect or reducible to something far simpler?

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

To load test your website, you could type this:

```
Definitions
! Standard Defines
Include "RESPONSE_CODES.INC" Include "GLOBAL_VARIABLES.INC"
CHARACTER*512 USER_AGENT Integer USE_PAGE_TIMERS CHARACTER*
CHARACTER*1024 cookie_2_0 CHARACTER*1024 cookie_2_1 Timer T_
Code
!Read in the default browser user agent field
Entry[USER_AGENT,USE_PAGE_TIMER Start Timer T_OBFUSCATED
PRIMARY GET URI "http://yahoo.chHTTP/1.1" ON 1 &
HEADER DEFAULT_HEADERS &
,WITH {"Accept: image/gif, image/xbitmap, image/jpeg, image/p
"application/x-shockwave-flash, application/msword, */*", &
```

or this:

www.webperformanceinc.com



Why code every test case by hand, when our unique software detects and automatically configures the test cases for you – quickly and accurately, then gives you superior reports that are easy to understand? With Web Performance automatic load testing, the time and money you save could increase productivity as much as 500 percent.

For more information about how you can increase performance and productivity using Web Performance automated load testing, visit www.webperformanceinc.com

Out of the Rut

by Michael Bolton

I'm testing, and I just realized that I'm bored. This is a Bad Thing. I'll have to do something about it. I'll sneak a few moments of *disposable time*, defined as "the time that you can afford to waste without getting into trouble." No tester that I know of is really supervised every minute of every day. We have moments in which we might try a new test idea, do some side research, look briefly at a different area of the product, or just do something else for a while.

If it turns out that I've wasted disposable time, it's OK; by definition I can afford to waste it. But maybe I'll learn something cool. I ruffle through the piles of paper on my desk, and I find a copy of *The New Yorker* magazine. The first article that I turn to is a review of a new TV series about psychotherapy [1]. To my surprise and delight, in the very first paragraph, it says, "... boredom isn't the same thing as being in stasis. Being bored doesn't mean there's nothing to do ... It means that something big ... is keeping us from doing what we want to do, from playing outside, from expressing ourselves, from moving forward." That reminds me of the cover of a recent *Scientific American Mind* magazine [2]. More shuffling in the pile, and then there, on the cover: BORED. The cover story says that boredom is triggered by repetition, minute and fragmented tasks, insufficient motivation, the absence of a need for intellectual engagement, and low levels of arousal (the psychologists' way of saying "the absence of things that wake us up"). These pathologies can build on each other. Without motivation, I lose engagement, and without engagement, motivation becomes more difficult.

The article also suggests that repression of a person's drives and desires leads to aimlessness and disconnection from the task at hand. If our work lacks meaning or purpose over time, we may experience existential boredom or ennui. I often hear from testers who



ISTOCKPHOTO

feel this way. Not surprisingly, they report that they are mandated to repeat heavily scripted tests and that they're not learning, not discovering new problems, and not finding bugs. They don't feel as though they're in control of their own testing process. Now I have a question I can ask myself: What can I do to retake control? Here are some heuristics.

Trade assignments. When I'm testing, I'm strongly motivated to explore and to exercise my own judgment as to what to do next, based on what I've just observed and evaluated. Some people may be more comfortable with a more directed, routine, confirmatory process, and some of that kind of work might be important in certain contexts. On a diversified team, someone else might be a better choice for that kind of work than I am. On a well-managed or self-organizing team, we might be able to trade assignments to play to our strengths. The downside of simply trading off work is that I might deprive myself of an opportunity to learn something valuable.

Exploit variation. When I'm feeling bored, I try to change the work in subtle but interesting ways. In a 2005 paper [3], James and Jon Bach identified a number of polarities in exploration—examples include doing vs. describing, careful vs. quick, working with the product

vs. working with the developer, design vs. execution, data gathering vs. data analysis, and solo work vs. team effort. When I'm bored, I pause, note whatever approach I'm using at the moment, and try going the other way. The same paper suggests branching and backtracking—deliberately choosing a different path of execution, and then aborting it and backing up several steps. This can be very useful for revealing state-based bugs. The downside of variation is that too much of it might take me off my charter or testing mission.

Collaborate. Chatting with a programmer, asking a user about workflows or pitfalls, or pairing with another tester are all ways in which I've refocused productively. A conversation—one with a whiteboard is almost always engaging—might allow me to model the system, see new risks, and take a different approach. One risk of collaborating is that, when we're both looking at the same area, we might lose the opportunity to spread out into testing different areas of the system—but I've always been impressed at the way two people see different things when looking at the same screen or whiteboard.

Focus on something else. Maybe I'm bored because I've been paying attention to one thing too closely, to the same

thing for too long, or to the wrong thing. In this case, I run a risk of falling into *inattention blindness*, a psychological phenomenon wherein we can miss significant things that are happening right in front of our eyes [4]. I've found that consistent alternation between focusing and defocusing—looking at some detail, then looking at the big picture, then looking at some other detail—helps keep me engaged and helps me see a different set of potential problems.

Put the machine to work. If it's genuinely better done by a machine, get a machine to do it. If I'm doing something that is monotonous and repetitive that includes a decision that the machine can make, there's a good chance that some little tool I cobble together will be extensible or reusable. Moreover, I need to exercise programming skill regularly, or I get rusty quickly. The risk is that automating a task limits my observations to things that I can program the machine to observe, greatly reducing my ability to spot an unanticipated problem. I have to consider opportunity cost. That reminds me to ...

Assess cost vs. value. My boredom might be a subconscious trigger that I'm doing something that's not terribly valuable. Maybe I'm whacking on an empty piñata and the value of what I'm doing no longer supports the cost of doing it. Perhaps it's not just uninteresting to me but uninteresting to my client, too. That leads me to the ...

Mission check. Maybe I should have a chat with my client or manager to make sure that we agree that I'm doing something worthwhile. Their perception of risk might not match what I've been observing in the product. If my client isn't currently available, I might change my focus to recording so that I'm prepared when she comes back. I like to manage this by using session-based test management and time-boxing my testing charters. In organizations that don't use this approach, I'll deliberately change charters on my own every ninety minutes or so. That helps keep me fresh, because the end of a session marks a good time to ...

Just take a break. Go for a walk, read a magazine, take a bike ride, grab a coffee, get a snack, run an errand, or

have a shower. Over the years, I've noticed a lot of people who are virtually chained to their desks. They claim it's because they have important work to do and that they can keep flow going. Fair enough, but if they're genuinely in flow, they're engaged—not bored. Making some change to clear out the cobwebs is important, too, sometimes. Just now for instance, I've glanced at one magazine, read an article in another, reflected on my process, and come up with some ways to make my testing more engaging and more valuable. Now, back to work.

Here's a key heuristic: Testing is *only* boring if you're not doing it well. Maybe you're bored because the information you're seeking is trivial, you're not in control of your own process, you're not learning anything new, the tests that you're running and the observations you're making are rote and mechanical (and therefore probably best left to automation), or you're stuck in a rut. Testing is *interesting and fun* when you feel like you're seeking important information, when you get to make choices about your process, when you're learning, and when the tests and observations are something that you believe only a human—and maybe only *you*—could do as well as you can. {end}

REFERENCES:

- [1] Franklin, Nancy, "Patients, Patients" in New Yorker, February 4, 2008. www.newyorker.com/arts/critics/television/2008/02/04/080204crite_television_franklin
- [2] Gosline, Anna, "Bored?" in Scientific American Mind, December, 2007. www.sciam.com/article.cfm?id=bored--find-something-to-live-for
- [3] Bach, James and Bach, Jon, "Exploratory Testing Dynamics," www.satisfice.com/articles/et-dynamics.pdf. See also Bach, Jon, Inside the Masters' Mind: Describing the Tester's Art, STAREAST 2006, Orlando, FL.
- [4] Visual Cognition Lab at viscog.beckman.uiuc.edu/djs_lab/



**What obstacles do you encounter that diminish your engagement with your process?
What do you do to get around them?**

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



ARE YOU IN SEARCH OF QUALITY? SO ARE WE.

Quality Assurance Analysts
Quality Assurance
Software Engineers

Apply online at
www.blackbaud.com/QAcareers
or contact Stephanie McDonald,
senior technical recruiter, at
843.654.3547 or
stephanie.mcdonald@blackbaud.com.

RELOCATION ASSISTANCE
IS AVAILABLE

Quality of Work

Work at the high-tech industry leader in software and services for nonprofits.

Use your talents to make a difference for our customers, who make a difference in the world.

Be an integral part of our product development team and help bring the best possible products to our customers.

Quality of Life

Enjoy the history, culture, and charm of Charleston, South Carolina. Receive great benefits and a competitive salary.

Get started today!

Blackbaud®

Communicate, Don't Assimilate

by Melissa Sienkiewicz

“So, we’d like you to go to Romania to help the new team ramp up over there,” Stephane told me.

I shifted in my chair and glanced at the fake golf ball stuck to the glass door of his office. “What exactly would you like me to do?”

“We’ve never opened an offshore office before, Melissa. We need you to figure out the best way for the Romania and Ottawa teams to work together.”

My stomach churned for a second. I was supposed to fly halfway around the world and, in two weeks, get a group of developers I only knew as voices on the phone to change their entire way of developing software to match our long-standing and ever-evolving process. Did I even have the tools to get this job done?

Then I realized that I did. The core of my company consists of great development services and our code of conduct standards: transparency, responsiveness, agility, collaboration, and constant improvement (TRACC).

What I needed to do was get the Romanian team members to really understand these principles. If they bought in to TRACC, they would naturally start acting like we did. It would be like having another Ottawa office in Romania.

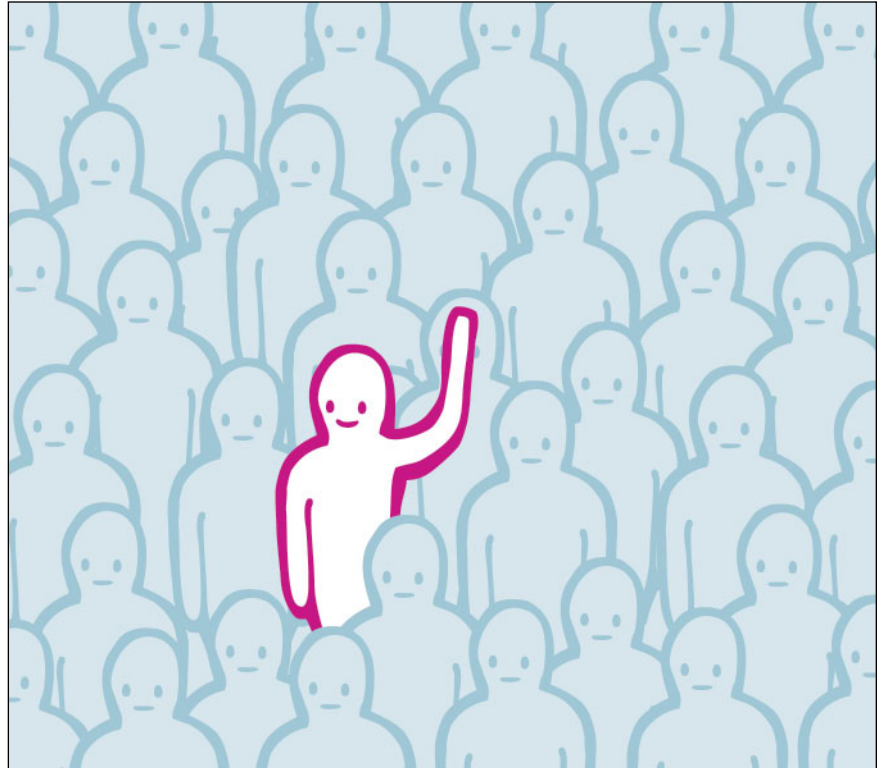
My anxiety died, replaced by excitement about teaching them to work like us. My first step would be a PowerPoint presentation aimed at convincing them how right our values are.

I’d get the new office on track and still have enough time to visit Dracula’s castle.

“OK,” I said to Stephane. “No problem. By the time I leave Cluj you won’t know the difference between them and us.”

The plane’s nose tipped down just about the time I put the last spinning, flaming bullet point on my final slide.

Ahh. Everything fell into place, and I



ISTOCKPHOTO

relaxed into my seat. Yep, it would all be fine now.

But something tickled at the back of my brain. It took me a few minutes before I realized that it was Stephane’s voice. *We need you to find the best way for the Romania and Ottawa teams to work together.*

Hadn’t I done that? Obviously we wanted to work together like one seamless team with no differences between the two offices. They could work like us and everything would be great.

I felt a chill, and it wasn’t from the overhead fan. I glanced at my presentation and saw it from another point of view—the Romanian team’s.

Was I really figuring out the best way for us to work together? No, I’d just assumed that Ottawa’s way was the best and intended to imprint it on them as if they were baby ducks following their mother around.

If I had come into the Cluj-Napoca office with a presentation on how they would be working from now on, I would have set up the dynamic that the Ottawa

office was going to provide all the leadership.

In short, I’d been thinking about the Romanian team as “them” and the Ottawa team as “us.” So wrong. There was no them. We were all us. The first C in TRACC stands for “collaboration,” after all, not “commanding people to be like you.”

I saved my TRACC presentation. Maybe I could recycle it for training new Ottawa hires or post it on the wiki. I wasn’t going to need it this trip.

John, the general manager, met me at the door to the Romanian office. “Do you have a presentation or something?”

I thought about my spinning bullet slides and laughed. He raised an eyebrow at me.

“No presentation. Why don’t you show me around the office? Then we can have coffee. I have some questions about what you guys are planning.”

“We have some ideas,” John said, a little tentatively. “If you want to hear them.”

STORY LINES

- When opening an offshore office, meeting with the team in person reduces ramp-up time and helps spread corporate values and processes.
- Forcing corporate values on a secondary office may create resentment and set up a bad dynamic.
- Rather than expecting a new office to work exactly the way the head office does, approach it from the angle of finding the best way to work together.
- Instead of pushing training, let the offshore team learn from you by asking questions and getting to know your processes.

"Sure do. I also wondered if there was anything you wanted to ask me."

He brightened. "You've been doing this for a long time and we want to take advantage of your expertise. You can help us avoid some mistakes."

"Sure, I'm happy to answer anything you want, but I'm not here to tell you guys how to work. Maybe we can work together to figure out the best way for us to work together." I grinned. "Hey, is that a foosball table?"

"We heard you had one in Ottawa," John said.

"You don't have to do everything just like we do in Ottawa," I told John. "But in this case, I think you made a smart decision. Are you up for a quick game?"

{end}



If you were asked to open an offshore branch, how would you identify challenges? How would you deal with the integration?

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

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@sqa.com.

STOP THE FREAK, KILL THE CREEP, BRING ORDER TO YOUR ALM TECHNIQUE

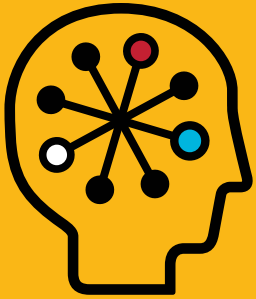
THE COMPLETE ALM SOLUTION ON TIME, ON BUDGET, ON THE MARK

Without oversight, software projects can creep out of control and cause teams to freak. But with Software Planner, projects stay on course. Track project plans, requirements, test cases, and defects via the web. Share documents, hold discussions, and sync with MS Outlook®. Visit SoftwarePlanner.com for a 2-week trial.



www.softwareplanner.com





Load Testing 2.0 for Web 2.0

With Rich Internet Applications, performance validation is critical.

The arrival of the Web 2.0 wave has far-reaching implications for web applications, web users and companies that do business over the Internet or use web-enabled business applications in enterprise environments. In short, Web 2.0 is a huge technology trend that will revolutionize the way users interact with the Internet and the way enterprises use and manage web-enabled applications.

Web 2.0 applications leverage advanced Rich Internet Application (RIA) technologies (such as Ajax and Adobe® Flex), along with enablers (such as RSS and blogs). From an end-user perspective, these technologies allow a richer, faster, more interactive experience with browser-enabled applications and services. From an enterprise perspective, these same technologies can help your company deploy new, customer-friendly application functionality in less time and at a lower cost.

With the arrival of RIA technologies, there are many new, more granular performance questions. If you can't answer these questions, you can't expect success with Web 2.0 applications.

By avoiding the time lags associated with interactive web applications, Web 2.0 applications promise to deliver a faster, easier and smoother user experience. But this enhanced experience isn't a given. While rich next-generation applications promise a radically different customer experience, they also have the potential to overwhelm the servers and networks that deliver content to client browsers.

Both the promise and the potential pitfalls stem from the unique ability of RIA technologies to continually exchange small and granular amounts of data with a server.

When thousands of users are interacting nearly continuously with an application, the strains on back-end servers can be enormous.

Are your applications ready for Web 2.0?

Given the potential peak demands of these next-generation applications, it's wise to approach the Web 2.0 wave with caution. Do you know how well your applications will perform? And do you know whether they will collapse under the weight of certain loads? Before Ajax or other RIA technologies can be ready for prime time in your business, you need to verify that they can deliver enterprise-scale performance.

All of this points to the need for sophisticated tools to test the performance of Web 2.0 applications and services in a pre-production environment. Yet with conventional performance testing solutions, this is easier said than done. There are many unknowns in stress testing Web 2.0 applications.

Conduct performance testing with HP Software solutions.

Test the performance of Rich Internet Applications.

Web 2.0 services can use Ajax, Flex and various other RIA and next-generation programming technologies. To validate the performance of services, you must be able to test these advanced technologies.

HP performance validation solutions give you:

- Built-in record and replay capabilities for Rich Internet Applications

- Ajax framework solutions based on LoadRunner Click and Script technology, which reduces the scripting process to a few mouse clicks
- Performance validation for Flex
- Performance validation for web services
- The planned addition of client-side breakdown

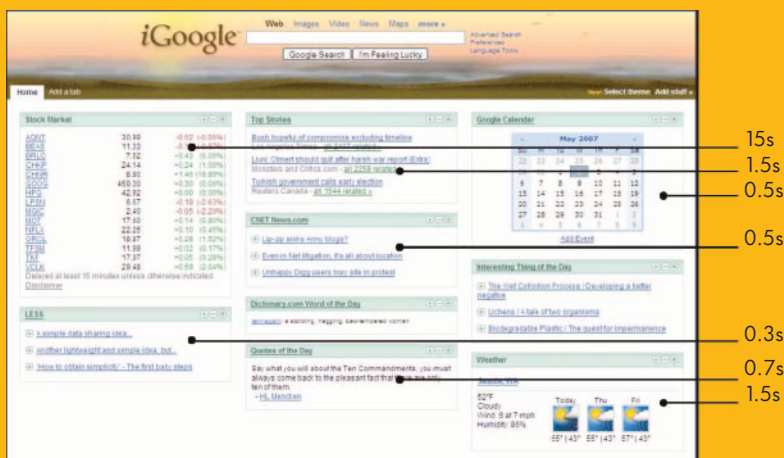
Break down the page to specific components and isolate performance issues.

With Web 2.0 applications, it is no longer enough to test the amount of time that it takes for an entire page to load. Instead, you need to test the amount of time that it takes for different components of a page to be updated. To do this, you need to break down a page into specific components and then isolate the performance for each.

How fast does the skeleton of the page load? How much time does it take for stock quotes to appear on a user's home page? Or how much time does it take for an updated list of employee names and numbers to appear in an ERP application? HP performance testing solutions allow you to answer questions like these by emulating the actions of hundreds or even thousands of users and then capturing the response times for key processes and transactions—so you can optimize your applications for better performance.

Test the performance for complex web operations.

With Web 2.0 applications, there are fewer web-page refreshes but much more interaction with back-end servers. While some changes to pages might not require interaction with a server, many user actions trigger interactions with back-end servers.



In this example, load times for different page components range from 0.3 seconds to 15 seconds.

To validate performance under these circumstances, you need to be able to test the end-user experience under different loads. For example, what happens when one user completes a drag-and-drop action?

How much time does it take for the page to reflect the action? And then what happens when a thousand users complete the same action at the same time?

To understand how your applications will work in a Web 2.0 production environment, you need to be able to simulate not just the actions of one user, but the actions of many users, perhaps even thousands of users. These are among the actions that cannot easily be duplicated without the proper automation tools.

With HP Software, you're ready for Web 2.0.

HP Software offers the key capabilities needed for performance validation of Web 2.0 applications.

Product offerings for Web 2.0 performance validation include HP LoadRunner software and HP Performance Center software.

To gain a firsthand look at HP load-testing capabilities for Web 2.0, download a trial:

www.loadrunner.com or contact HP at 1-800-TEST911. Or to learn more about HP Software products, visit www.hp.com/go/btsoftware.





it's
a

Bug!

by Robert
and Ann

Testing Lessons from Labor Triage

Bug triage is about making decisions about the fate of software bugs: Should we keep a bug? Should we fix it now? Can we fix it later? Can we live with it? What other choices do we have? What should we do next?

I have implemented bug triage in every software project I've run since 1992. My inspiration in organizing bug triage comes directly from the world of labor and delivery nursing, which I learned about from my wife, Anne, a nurse at the birthing center of the Royal Victoria Hospital in Montreal. Anne has triaged hundreds of expectant mothers arriving at the hospital in anticipation of the childbirth experience.

Labor and delivery triage is about deciding the course of treatment of patients arriving at the hospital. Anne must decide the course of action on the spot, often with minimal information guiding life-critical decision making.

I apply the four basic steps of labor and delivery triage directly to bug triage:

- Preliminary Assessment
- Interview
- Exploration and Observation
- Taking Action



Preliminary Assessment— Looking the Bug Right in the Eye!

The labor triage assessment begins as soon as the triage nurse sees the patient. The triage nurse observes the patient: Can she talk? Is she agitated? Is she about to faint? Is she excited, anxious, or in any sort of trauma? Her first judgment is related to the urgency of the situation. Sometimes the patient needs immediate medical care, even before Anne knows the patient's name—the baby will not wait. The triage nurse must be prepared for anything and remain calm despite the adrenaline rush. In some cases the triage nurse delivers the baby on the spot. It's a job that has all the excitement of a TV drama. The triage nurse knows what to look for and is trained to recognize and act in response to critical situations.

The bug triage assessment begins as soon as I see the bug. I take a good look at the bug: Will critical user tasks be blocked? Is the bug infectious? Will it break other things? Does it violate laws, regulations, or contractual obligations?

Did we lose client data? Does this bug block us from doing other work? My first judgment is related to the urgency of the situation.

In the preliminary assessment of bugs I look for those bugs that demand immediate action and then I initiate whatever course of action is required. In some projects I call this bug filtering. I cut out any paperwork or bureaucracy involved and get right to resolving the bug. When a bug demands action, I take action.

To hone my preliminary assessment skills I have to study a lot of bugs. I want to avoid crying wolf and getting developers to start fixing a problem that is not really urgent. Knowledge gleaned from studying bug taxonomies is a great source of information. I always study real bugs experienced in similar projects. In order to get a good sense of the urgency of a problem, I try to understand the impact on the end-user of not fixing the problem immediately.

The preliminary assessment helps trigger immediate action before going to bug review meetings or doing any further testing. Being prepared for a preliminary assessment includes a blend of knowing what to look for and knowing how to get things done. I make sure project managers, development leads, and all team stakeholders know that I do preliminary assessments and that I can sidestep bureaucracy in some urgent cases.



Interview— Understanding Context

In labor triage context is everything. The same conditions observed with a different context can lead to dramatically different results. One example is that of labor contraction timing. Imagine that a patient calls triage reporting contractions lasting sixty to ninety seconds with mild intensity and taking place every ninety minutes. If you knew that the mother had recently experienced a car accident, you would want to see her right away. If you knew the mother was just watching a soap opera on TV, you might suggest she call back in a few hours and take it easy.

The triage nurse interviews the patient to learn about context. She asks questions that help identify the phase of labor the woman is in. Some basic information about the patient is collected: name,

TRIAGE NURSING NOTES

A principle used by triage nurses is “If you do not see it in writing, it was not done.” Note taking is a critical skill of the triage nurse. Almost any medical professional who will be involved in the case may reference these nursing notes. A triage note includes entries for each and every event, interaction, observation, test done, and test result. Notes include the questions asked, responses given, and action taken by the nurse. Each notation includes the date and time the event took place. The triage nurse must take detailed notes on the entire clinical encounter coupled with all test results and observations. Experienced practitioners use a terse, unambiguous style to capture their notes. They use a template form to guide note taking, which focuses on the observations and actions taken but does not include subjective assessment—as Sergeant Friday would say, “Just the facts.”

NOTE TAKING IN TESTING

I have learned over the years that communication is one of the most important skills in testing. Imagine the audience for a bug report: We are writing to other testers, test leads, development leads, help desk staff, developers, product managers, technical writers, project managers, and many different product stakeholders. It is important for testers to meaningfully communicate to this varied audience without over simplifying, confusing, or complicating the information. Software testing session notes and bug reports should be able to stand up to the same level of scrutiny as triage nursing notes.

age, gestational age, doctor, any previous pregnancies, and what happened during those pregnancies. In addition, some information about the pregnancy is collected, such as any special conditions, results of ultrasounds, and information about any medical interventions. Information about the actual condition of the patient is collected: frequency of contractions, whether the “water” has broken, and any special pains or indicators. The interview takes only a few minutes and provides information that is used to decide if the patient should be admitted, observed for a few hours, or sent to another department.

In bug triage context is everything. The same bug may require urgent intervention in one context and be easily deferred or worked around in another context.

Testers interview bugs all the time. Testers build up information to help decide which bugs to fix and which bugs to keep. They collect basic information in a bug-tracking system: When did it occur? What version of the software was being used? What operating system? What build, locale, state of the database, and state of the system? What else was going on at the same time? Testers ask questions about the specifics that exposed the problem: Does it happen all the time? What steps could reproduce it? What other tests related to the problem have been done and what were the results? Other questions relate to the condition of the bug: What is the severity? What is the consequence of not fixing the problem? How much damage has the bug caused?

I consider the following three sources of context information about the bug before taking action:

Business context: Why is the bug of importance to our business? What would the impact be if the bug were not fixed? Would workarounds be acceptable?

Technical context: Are there any special technical concerns about the bug? Is it in our code? Do we depend on a third party? Could fixing this bug break something else?

Organization context: Was the issue reported as part of testing or from the field? Will there be further levels of testing downstream? Can we gracefully

update the client after deployment? Do we have access to developers who can fix it?



Exploration and Observation—Learn More About It

After the interview the triage nurse performs some medical tests to learn more about the patient’s condition. The triage nurse checks body temperature, blood pressure, and fetal heart rate and does some basic blood and fluid tests. These test results combine with interview and preliminary assessment data to help guide decision making. When conditions are uncertain, the triage nurse will monitor the patient for a few hours. Monitoring uses different medical testing techniques, such as ultrasounds, to help observe emergent behaviors before a medical course of action is taken.

Sometimes I need more information before deciding what to do with a bug.

I may assign a tester to further investigate the problem or to work directly with developers to get a better understanding of the bug. Exploratory testing around the problem area can be used to gather additional data to help guide decisions. Are there other ways to trigger the bug? Are there other emergent behaviors associated with the problem?

I encourage testers to capture data about the software being tested and to observe the environment in which the software is running. It may be important to measure how much CPU capacity is being used by the application under test. How fast is the application responding to requests? Do we have basic data integrity? How are systems resources consumed?

I also want to confirm that other parts of the application work well enough to handle typical transactions. When a bug shows up in one area of the software, it is important to confirm quickly that other parts of the code or data are still working.



Taking Action—Getting Things Done

Three outcomes may result from labor triage: The mother is admitted, observed, or sent home.

The triage nurse has access to established medical protocols to help her decide appropriate actions based on the data collected during the preliminary assessment, interview, and clinical observations. The protocols are described in a clear one-page format in which the presenting condition, key context drivers, and recommended course of action are all spelled out. The triage nurse does not rely on the protocol manual when faced with the day-to-day realities of labor triage. She is in the hot seat and must react to the realities of the situations with which she is confronted. She must act. She must combine her experience and knowledge on the fly.

I triage bugs to determine one of three possible outcomes: Fix it now, fix it later, or do not fix it.

I want to make sure all bug triage decisions are influenced by business, technical, and testing factors. I have never been able to put together a crisp series of protocols such as those used in labor triage. I make sure that a small team of stakeholders makes decisions about bug priority. I like to involve a product manager who advocates for the customer, a development lead who is aware of the

technical risks of the project, and a test lead who is driving the testing initiative of the project. Generally the test lead provides objective information about the bug from the preliminary assessment, interview, and exploration steps. The team weighs business and technical concerns in order to come up with a decision about the bug.

This final decision draws upon all of the information gathered so far. I find bug triage teams work best when the team is 100 percent in sync regarding the purpose of the project. Why are we doing this project? What are the key business issues? What are the technical challenges? What value does this project offer and to whom? If team members are in “value sync,” then they will be better able to make difficult decisions.

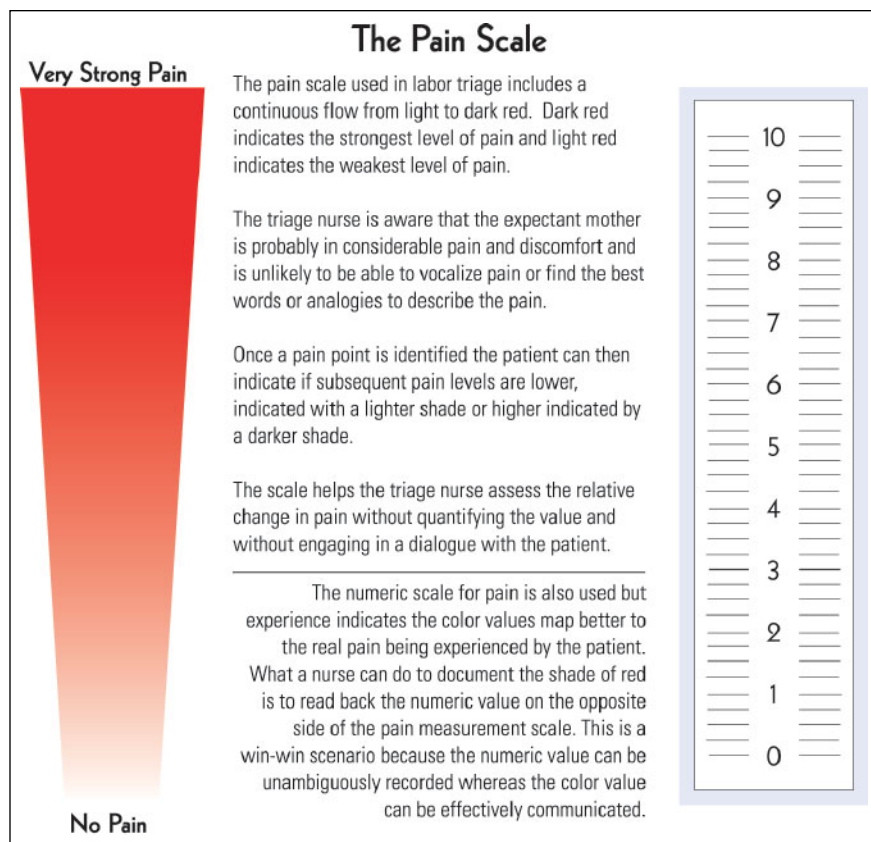
I ask the team to consider these important questions for each bug: What is the benefit of fixing the bug? What is the consequence of not fixing the bug? I also want to make sure they consider the reverse: What is the *consequence* of fixing the bug? What is the *benefit* of not fixing the bug? The team considers the tradeoffs related to fixing the bug or leaving it in there.

QUANTIFYING PAIN

The triage nurse has many interesting methods to help quantify factors of the pregnancy that would otherwise be very difficult to describe. One tool is a color-coded pain scale that includes a range of shades of red presented from light red to very dark red. The color system is used when patients are asked to describe their level of pain. Although this does not give an absolute measure, it is very effective in helping patients communicate when pain levels are increasing or decreasing as time progresses or as a result of different medical interventions.

QUANTIFYING SEVERITY

One of my most difficult problems is trying to find a meaningful way to quantify the severity of a bug. How much damage could the bug cause? I have used many schemes over the years, and on a recent project I took the labor triage pain scale to a customer site. Developers, testers, and product managers would point to the shade of red associated with a bug to indicate the severity. This let us immediately see disagreements between project stakeholders, which subsequently led to better understanding. We also could see the relative severity of different bugs. Color coding severity may appear subjective but it definitely helped us focus and communicate more effectively!





Final Words

In *Dynamics of Software Development*, Jim McCarthy suggests that project teams should “triage ruthlessly” to make all decisions that shape a product. Although they are dramatically different domains, labor triage offers a lot of valuable lessons that we can apply to software testing projects. Testers can model some of their workflow based on the stages and activities the triage nurse uses to guide decisions.

Just as a labor triage nurse must know when to send a patient home, when to start treatment, and when more knowledge is needed, the tester should learn to decide which bugs to fix, which bugs to keep, and when to get more information before making a decision. Triage helps testers react and adapt to the critical context drivers on our testing projects and helps us focus on delivering the value that makes a difference. **{end}**

WHEN IT'S TIME



TO OUTSOURCE SOFTWARE TESTING



IT'S TIME TO CALL **ACULIS**.

ON-SITE, OFF-SITE, OFF-SHORE
A BLENDED APPROACH TO SOFTWARE TESTING.

WITH STATE-OF-THE-ART FACILITIES IN THE U.S. & ARGENTINA

HOW DOES YOUR QA PROCESS RATE?

Visit Us at StarEast 2008 - May 7th & 8th - **Booth #45**

For a Chance to Win a **FREE** 3-Day QA Assessment & Scorecard Report



ACULIS / **IT ACCELERATOR**
DEVELOP IT / TEST IT / GLOBALIZE IT / STAFF IT

1-866-4ACULIS / WWW.ACULIS.COM

PATIENT ADVOCACY IN TRIAGE NURSING

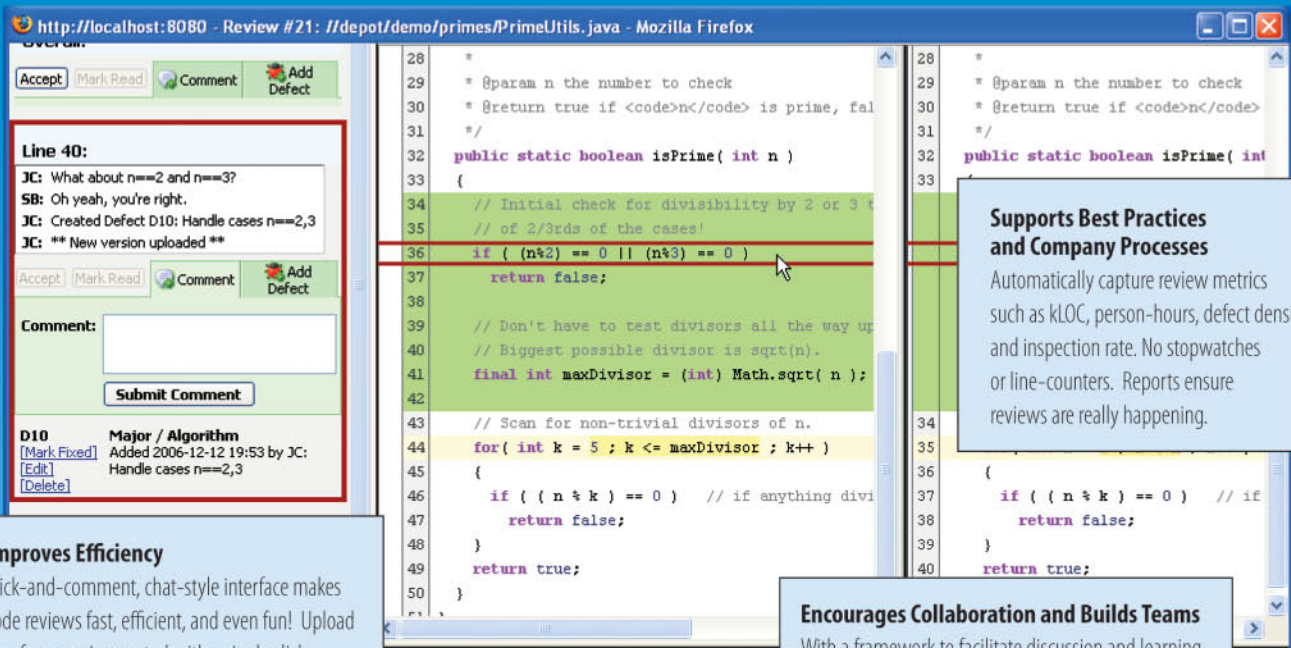
All expectant mothers do not need the same blend of tests, interventions, and medical care. The triage nurse does not just kick off processes. The triage nurse also actively represents the patient to the other practitioners called upon to consult or act in the case. The triage nurse gives other medical professionals a heads up on the upcoming delivery. The triage nurse communicates the relative urgency of the patient's situation and ensures that everyone knows what may be required. In many ways the triage nurse must be aware of the work required by all medical professionals involved in the delivery process. Miscommunication can result in serious and potentially fatal problems. Triage nurses cannot exaggerate situations to draw too much attention to a case. To succeed the triage nurse must be credible. Consistency in objectively identifying the severity and priority of the case leads to this credibility. This earned credibility then triggers people to prepare and act based on the request of the triage nurse. Without credibility the triage nurse would be ineffective because nobody would listen.

BUG ADVOCACY IN TESTING

Testers without credibility find it difficult to influence projects. Testers build credibility in many ways. Testers should not exaggerate the potential impact of bugs, nor should they trivialize bugs. Testers can advocate bugs effectively by consistently providing clear and objective input about the issue being described. Testers should anticipate the type of information needed by all members of the development team. Just showing that the bug exists may be insufficient. Testers can provide information that may be useful in evaluating the priority of the bug, such as the consequence to the user community if the problem is not corrected. Testers can provide information that may be useful to the developer such as the technical state of the application and actions that took place before the bug was observed. The tester can anticipate the type of questions that may arrive from the help desk or customer support departments. The tester can provide a pithy summary of the bug, which helps non-technical executive stakeholders understand the nature of the concern.

Code Review

No Meetings. No Paper. No Kidding.



Supports Best Practices and Company Processes
Automatically capture review metrics such as KLOC, person-hours, defect density, and inspection rate. No stopwatches or line-counters. Reports ensure reviews are really happening.

Improves Efficiency
Click-and-comment, chat-style interface makes code reviews fast, efficient, and even fun! Upload files from version control with a single click.

Encourages Collaboration and Builds Teams
With a framework to facilitate discussion and learning, developers work together instead of in isolation, even if separated by an ocean.

Code Collaborator

Tool-assisted peer code review with Code Collaborator means no more busy-work, marked-up print-outs, or meetings. Start a free trial today and cut up to 75% of the time out of code review.

STARTING AT

\$199

www.CodeCollaborator.com



Learn how with this FREE book

Ten essays about code review including the largest case study of lightweight peer code review.
www.codereviewbook.com



Another great tool from SmartBear,
the experts in code review.

www.SmartBearSoftware.com
(877) 501-5651

The Chivalrous Team
Member

10 Knightly
Virtues
Applied to
Software
Development

by Martin Kearns
& Mike Cohn

In seeking to improve how we develop software, we continually inspect and adapt. While thinking recently about the characteristics of the ideal team member, we found similarities between the best-performing software teams of today and the Knights of the Round Table. Robert Goulet, as Lancelot in ye olde movie Camelot, sang:

“A knight of the Table Round should be invincible,
Succeed where a less fantastic man would fail.
Climb a wall no one else can climb,
Cleave a dragon in record time,
Swim a moat in a coat of heavy iron mail.
No matter the pain, he ought to be unwinceable.
Impossible deeds should be his daily fare.”

While we do not think our team members need to be able to cleave a dragon in record time and most would look strange in a coat of heavy iron mail, we do think the ancient code of chivalry serves as an excellent code of conduct for team members on today’s complex software projects. Our demons are no longer dragons, fair maidens in distress, or a wicked cousin (next in line for the throne!). Yet, certainly, given the schedules that many are asked to meet, impossible deeds are often our daily fare.

Hang on—before you think we’re crazy or that we’ve spent too much time watching Robin Hood: Men in Tights, allow us a moment to plead our case. Ancient knights were chosen as much for their personal traits as for their strength and agility. We suspect you hire team members as much for their personal traits as for their technical knowledge and experience. We prefer team members who are dependable, communicate well, are flexible in their preference for technologies or tools, generous in sharing their time with others, and loyal to the team are preferable. In fact, many of the sought-after attributes for knighthood in “auld times” are still desirable in team members today.

It was with this thought in mind that we mounted our trusty steeds (Firefox) and visited our magical search engine, Merlin (aka Google), on a quest to discover the exact code of chivalry by which ancient knights lived. What we found was that there was no single ancient code—ancient knights and authors described knightly virtues differently. However, the various ancient codes were combined and updated into a modern code of chivalry in 1997 by Brian Price, an expert in medieval history.

In Price's modern rendering of the ancient code we believe we've found the Holy Grail of desirable attributes for today's chivalrous team members. We start each of the following sections with one of the ten virtues described in Price's modern code of chivalry.

Prowess

To seek excellence in all endeavors expected of a knight, martial and otherwise, seeking strength to be used in the service of justice, rather than in personal aggrandizement.

We'd certainly like to work with team members who seek excellence in all endeavors. Before Mike began his career as a consultant, his main criterion in deciding whether to join a company or team was whether there were individuals on that team from whom he could learn. Not only does being around others who pursue excellence in their endeavors provide learning opportunities but it also pushes each of us to be our best. If we know that our team's DBA is writing sloppy code in the database because he doesn't think this product will be around for the long term, then we, too, are more likely to write sloppy code.

Of course, if excellence is sought solely for the sake of personal gain, then it does not benefit the team. The ideal team member is one who seeks excellence because it helps the team as a whole.

Prowess extends to resolving problems that are "not mine." This should be done not for fame but for the betterment of the product. Martin has observed that often when a team starts to work together more collaboratively, some of the stronger subject matter experts struggle to become part of the team. Strong subject matter experts are used to acquiring fame for resolving difficult problems. As

the team works collaboratively to solve problems, individual fame evaporates and is instead shared among team members. For some this is a difficult transition.

Justice

Seek always the path of "right," unencumbered by bias or personal interest. Recognize that the sword of justice can be a terrible thing, so it must be tempered by humanity and mercy. If the "right" you see agrees with others, and you seek it out without bending to the temptation for expediency, then you will earn renown beyond measure.

Ruby on Rails is currently a hot technology, so Mike wasn't surprised to see it in use at one of his client sites. What did surprise him was that *this particular* client was using Ruby—it had used and was still using Java for all its other development work. A number of Java systems built over the past five years, new projects beginning in Java, and one new project in Ruby on Rails seemed an odd combination. After a bit of questioning it became apparent why the new project had been started in Ruby on Rails, and the reason is one that underlies too many technology decisions—the project was being done in Rails because the two senior developers on that project wanted to add Rails experience to their résumés. Any doubts about this evaporated six months after the project started when one of the senior programmers left for a shiny new Rails job.

Someone who always seeks the right path for a project, unencumbered by personal interests or bias, is definitely someone we would want as a team member.

Loyalty

Be known for unwavering commitment to the people and ideals you choose to live by. There are many places where compromise is expected; loyalty is not amongst them.

A team member displays loyalty in a number of ways. You need to believe in the principles and goals your team has agreed to follow. If your team has chosen to do a daily build and to never go home while the build is broken, you must be loyal to this precept. You must be loyal to your fellow team members even in times of stress or conflict. A team can

only be self-organizing when each team member freely gives information and assistance to other team members without a need for personal gain. True belief and loyalty in the development ideals you choose to live by must never be waived.

Loyalty to our customers and stakeholders is also important. As developers we give the business control over priorities. As team members we must stay loyal to these objectives and not think we know better or make our own judgment on what the business "really needs."

Courage

Being a knight often means choosing the more difficult path, the personally expensive one. Be prepared to make personal sacrifices in service of the precepts and people you value. At the same time, a knight should seek wisdom to see that stupidity and courage are cousins. Courage also means taking the side of truth in all matters, rather than seeking the expedient lie. Seek the truth whenever possible, but remember to temper justice with mercy, or the pure truth can bring grief.

It can sometimes be difficult for a team member to be the bearer of bad news. No one wants to admit at a daily stand up that he needs assistance. Martin recalls the first time he needed to seek help from his team:

"I had just been exposed to object-oriented programming after a decade of procedural programming on mainframes. I was struggling and needed help. However, a feeling of guilt and failure came over me in the daily stand up, discouraging me from expressing my problem to the team. I tried to avoid asking with delay tactics and some dodgy procedural programming hacks. Eventually, when I did ask, I was pleasantly surprised with the reactions of my team members. People congratulated me for saving days on the project by expressing my problem quickly so that a resolution could be found and acted on by the team instead of burning money in a solo effort that would most likely have ended in failure."

It takes courage to tell your teammates you need help. Such courage should be encouraged, because by speaking openly you are seeking the best results for the team.

Courage is also required when speaking with clients or management.

For too long we have deceived our clients by saying progress was steady, when in truth deliverables could not be met on time or on budget. In the past we continued blindly, hoping to “catch up” but inevitably failing our clients. The courage in giving a true and honest reflection of progress (velocity) will be rewarded as you are supplying the client with the ability to change perception, scope, and so on in enough time to achieve success.

Faith

A knight must have faith in his beliefs, for faith roots him and gives hope against the despair that human failings create.

Every team member needs to have faith in the scope accepted into an iteration and in the goals and development principles of the agile methodology in general. Expressing dissatisfaction after the planning meeting and continually attempting to railroad the iteration via a series of interruptions will have a very negative influence on achieving the goals set. Faith in all decisions made in iteration planning is needed, which is why it is so important to seek consensus. Doubts should be noted and revisited in the retrospective where success will be inspected and, based on team dialogue, adaptations may be required going forward.

Agile can be effectively implemented only when all team members have complete belief in the processes. Martin found himself in a difficult situation when he left a first iteration planning meeting with a team member clearly not happy about using agile techniques. This individual predicted failure of the agile process and was quite public in voicing that opinion.

The iteration was railroaded and a lot of Martin’s time was spent doing damage control, resolving the rifts that this vocal negativity wrought within the organization. This experience underscores not only the importance of consensus when a team decides on its beliefs but also the strength of faith a team needs to have in its process when that process is challenged.



Nobility

Seek great stature of character by holding to the virtues and duties of a knight, realizing that though the ideals cannot be reached, the quality of striving towards them ennobles the spirit, growing the character from dust towards the heavens. Nobility also has the tendency to influence others, offering a compelling example of what can be done in the service of rightness.

Teams usually take their first steps toward agility in one of two ways: They focus on learning to work in short, timeboxed iterations, or they focus on learning a couple of new technical practices, such as pair programming, emphasizing automated unit testing, or doing test-driven development. These initial steps often lead to great results that should serve as the basis for becoming more agile. Unfortunately, we see too many teams stop after this first success. They forget that part of being agile is continuously improving. Agile teams, like ancient knights, should strive to reach ideals—perfect, bug-free code every time—even though they know they can never succeed.

Teams must remain noble when trying to influence others in adopting agile. Forceful exertion of beliefs does not work. Self-awareness and empathy are the skills we must master. In order to influence others, you must begin by understanding how people feel, their perspectives on the topic, and what factors

are influencing their decisions. While empathizing, we must strive not to preach, bully, nag, or belabor a point. Influencing others occurs through a bond of respect to one another, a sharing of goals, and a strong sense of conviction toward achieving those goals. Attaining these skills requires constant effort.

Largesse

Be generous in so far as your resources allow; largesse used in this way counters gluttony. It also makes the path of mercy easier to discern when a difficult decision of justice is required.

Who hasn’t worked with the colleague who gave freely of his or her time, patiently answering our questions, and showing how things should be done? This type of mentor—whether senior to us in the organization or not—can have tremendous influence. Not only do we learn skills from such a colleague but we also learn the value of mentoring and repay that kindness by mentoring others ourselves.

Sadly, many of us have also worked with the colleague who refused to share his time or expertise—the gruff coworker who clings tightly to his knowledge of how to change the transaction date on processed data records, following his unstated rule that unshared knowledge leads to job security.

Which of these individuals did you prefer working with? How would you like your coworkers to think of you? Be generous as far as your knowledge and experience allow.

Defense

The ideal knight was sworn by oath to defend his liege lord and those who depended upon him. Seek always to defend your nation, your family, and those to whom you believe worthy of loyalty.

Though ancient knights may have willingly given their lives to defend their liege lords, we’re unwilling to suggest that modern-day software chivalry calls for a team member to be willing to die in defense of his CEO. In this modern age team members instead need to defend their applications vigilantly. Applica-

tions are frequently under siege from decaying code, schedule pressure that leads to sloppiness, and budget constraints. Rather than fight these modern-day dragons with lances, broadswords, and axes, today's chivalrous team member relies on automated testing, test-driven development, refactoring, working at a sustainable pace, and other agile weapons.

The products we build can in turn be used as weapons in defense of our users. When developing software, spare a thought for the end-users and how effective they can be at their jobs. What is the end result of the infamous suggestion of a "manual work around?" To achieve an effective working relationship with users, our perspectives and relationships with them must change.

Humility

Value first the contributions of others; do not boast of your own accomplishments, let others do this for you. Tell the deeds of others before your own, according them the renown rightfully earned through virtuous

deeds. In this way the office of knighthood is well done and glorified, helping not only the gentle spoken of but also all who call themselves knights.

Humility is a very strong element in any transition to agile. An effective team with a strong collaborative style does not have room for large egos. Team members seek satisfaction by experiencing how their achievements (however small) can combine into a continually improving product. Chivalrous team members frequently ask, "How has my accomplishment today enabled other members of my team to complete related tasks?" Regular releases to users provide another sense of satisfaction when users are able to do some business function they couldn't previously do.

Franchise

Seek to emulate everything I have spoken of as sincerely as possible, not for the reason of personal gain but because it is right. Do not restrict your exploration to a small world, but seek to infuse every aspect of your life with these qualities.

Should you succeed in even a tiny measure then you will be well remembered for your quality and virtue.

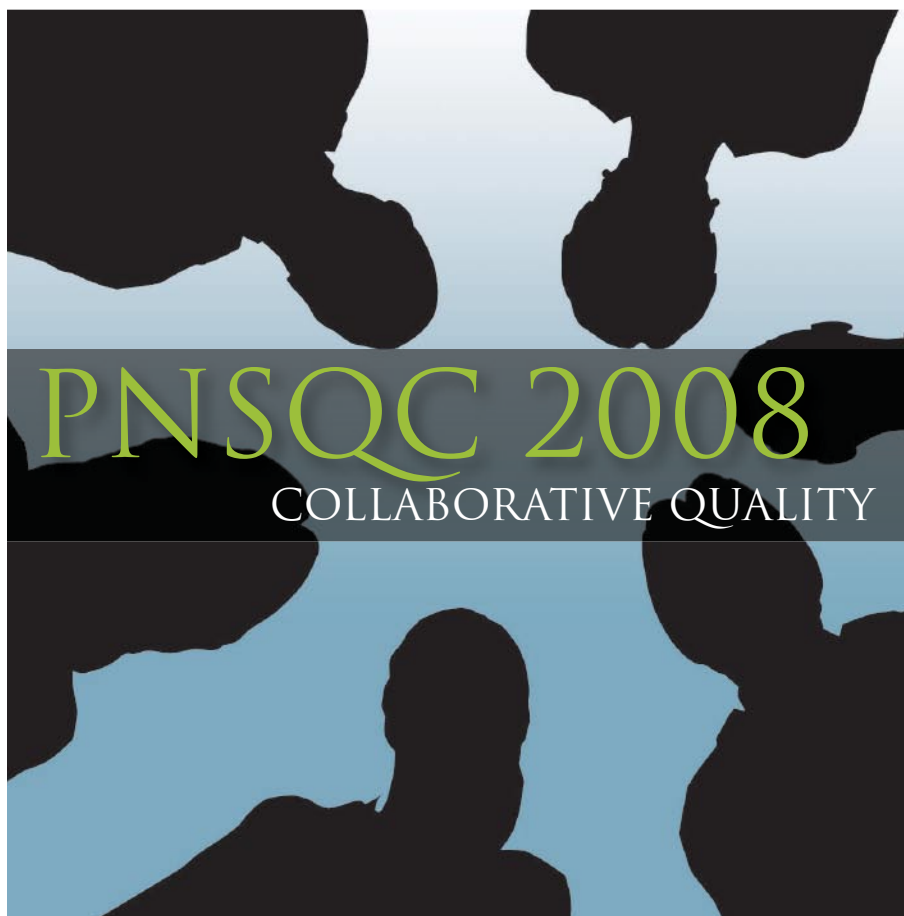
The benefits of adhering to the principles underlying agile software development can be applied in other walks of life. We are aware of or have used the principles of transparency, short time-boxes, measurable progress, rapid feedback, and increased communication in planning a wedding, renovating a restaurant, running a sports club, and managing a child's school work.

Whether you have dragons to cleave, maidens to rescue, or simply deadlines and budgets to meet, working daily in ways that exhibit the ten knightly virtues is sure to help your team accomplish these once-impossible deeds. {end}

Sticky Notes

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

■ *A Code of Chivalry*, by Brian R. Price



26TH ANNUAL PACIFIC NW SOFTWARE QUALITY CONFERENCE

October 13-15,
2008
Portland, OR

www.pnsqc.org

World-class quality does not happen in a vacuum. Agile-inspired collaboration spans levels, disciplines, and industries.

KEYNOTES

Quality Dynamics of Agile SW Development
RON JEFFRIES & CHET HENDRICKSON

The Art of Building Consensus
SAM KANER

INVITED SPEAKERS

Playing Nice in the Sandbox
JANET GREGORY

Selling Your Ideas To Management
STEVE SMITH

*Expanding Trust and Collaboration
with Earned Value Tracking*
TAMARA SULIAMAN & HUBERT SMITS

*It's Not Just an Update: Using Status Reporting to
Expand Collaboration*
MIKE KELLY

WORKSHOPS

Collaboration using the Agile Testing Taxonomy
JANET GREGORY

Zeroing in the Right Problem
STEVE SMITH

SQL for Testers
KAREN N. JOHNSON

The Product Development Game
TAMARA SULIAMAN & HUBERT SMITS



Software Testing Certification

Certified Tester—Foundation Level Training



More than 80,000 Certified Testers Worldwide. Why Not You?

SUMMER/FALL 2008

Las Vegas, NV	June 8–10, 2008
Bethesda, MD	June 10–12, 2008
Ottawa, ON	June 17–19, 2008
Portland, OR	June 17–19, 2008
Milwaukee, WI	June 17–19, 2008
Boston, MA	August 26–28, 2008
New York/NJ Area	September 8–10, 2008
Minneapolis, MN	September 9–11, 2008
Salt Lake City, UT	September 9–11, 2008
Washington, DC	September 15–17, 2008
Philadelphia, PA	September 23–25, 2008
Atlanta, GA	September 23–25, 2008
Anaheim, CA	September 28–30, 2008
Indianapolis, IN	Sept. 30–Oct. 2, 2008
Jacksonville, FL	October 7–9, 2008

Toronto, ON	October 7–9, 2008
Rochester, NY	October 14–16, 2008
Kansas City, MO	October 14–16, 2008
San Francisco, CA	October 20–22, 2008
Sacramento, CA	October 21–23, 2008
Pittsburgh, PA	October 21–23, 2008
Omaha, NE	October 28–30, 2008
Charlotte, NC	October 28–30, 2008
Cincinnati, OH	October 28–30, 2008
Ft. Lauderdale, FL	November 4–6, 2008
Bethesda, MD	November 4–6, 2008
Tampa, FL	November 17–19, 2008
Sunnyvale, CA	November 18–20, 2008
Phoenix, AZ	December 2–4, 2008

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

PM Project
Management
R.E.P. Institute

SQE
TRAINING

www.sqetraining.com

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

Let's **TALK**

Learning to Communicate
in an

AGILE

Environment

by Ken Pugh





"To assume that another person is being recalcitrant because he wants to do something his way is not a useful step toward communicating."

Agile development employs more oral communication, feedback, and interaction than traditional development. Instead of creating written requirements specifications, a customer simply tells the developer what the requirements are. A customer and a developer may sit together to create a graphical user interface. A customer and a tester may sketch the acceptance tests for a user story on a whiteboard, rather than create a formal test document.

Developers and testers who have worked in a traditional environment may struggle to adapt to these more interactive relationships. To help with the transition, this article presents a few important topics regarding communication including personality types, communication styles, active listening, the Satir interaction model, exchange of tokens, and communication modes.

Personality types and communication styles

The Myers-Briggs Type Indicator (MBTI) is an instrument that describes individual preferences for using energy, processing information, making decisions, and relating to the external world. These preferences result in a four-letter "type" that can be helpful in understanding different personalities. The personality types exhibit different ways in which people communicate and how you might best communicate with those types (see the StickyNotes for a list of communication strategies for each type and a test to determine your own type).

You may have taken an MBTI test in the past. For those who haven't, here is a basic outline of the MBTI's four preferences. Each preference has two endpoints. As an introduction (or a review) let's see how each of these prefer-

ences might be reflected when a person is going to lunch.

The first preference describes the source of your energy—introvert or extrovert. An introvert draws energy internally, from his own thoughts and ideas. An extrovert draws energy from interactions with others. The extrovert might ask everybody if they want to go to lunch. The introvert may prefer having lunch by himself.

How one processes information is the next preference—sensing or intuitive. A sensing person is visual and fact oriented, while an intuitive person is open and instinctual. When an intuitive person looks at a menu, he tries to get a general idea of the type of food and the price range. The sensing person might read every line of the menu before deciding if he wants to eat there.

Decision making is the third preference—thinking or feeling. The thinking person uses logic and standards in making decisions. A feeling person is more concerned with feelings and personal relationships when making a decision. The thinking person might figure out which restaurant is the closest or which one has the cheapest food. The feeling person might suggest not going to a particular restaurant because someone in the group recently ended a relationship there.

The fourth preference—judging or perceiving—deals with how an individual relates to the external world. The judging individual is organized and structured. The perceiving person is spontaneous and flexible. The judging person has the lunch date in his Outlook calendar. The group will be leaving precisely at 12:00. The perceiving individual may make the lunch date on the spur of the moment. If it is around noon sometime, he's OK with that.

When you're communicating with another person, you will have an easier time if you know his MBTI type. But since most people do not wear their MBTI classification on their lapels (except for some particular groups), it's important to appreciate that each of us looks at the world differently.

So how do you communicate when you are a different type from the other person? The first thing to do is acknowledge those different types. To assume that another person is being recalcitrant because he wants to do something his way is not a useful step toward communicating. Adapting your communication processes to appeal to both styles would acknowledge the legitimacy of the other person's style.

Often group decision-making processes assume everyone will speak out. The extrovert asks, "So what does everybody think about this? Let's hear from you." The introvert may not give immediate answers. He may be pondering the ramifications of his answer before contributing it. You could try procedures that may appeal to introverts. For example, instead of asking for oral responses, you could have everyone write down ideas on index cards or post-it notes. Place the cards up on a board in clusters of related ideas. Now each person's voice has been heard.

As a side note to individual brainstorming, I'd like to interject the notion of "throwing the cards on the table." In communicating ideas, we often get hung up on our own ideas. We may feel that we need to promote or protect the ideas that we have suggested. One way to avoid this is to adopt the concept of "throwing the cards on the table." After ideas are generated in the individual brainstorming, the index cards are literally thrown on the table. The originator's



identity is lost (at least theoretically). Each idea then can be considered on its own merits, not on the relative status of the originator. The best ideas often are compilations of many ideas that have been thrown on the table.

Another example of introvert/extrovert interaction is in estimating the time needed to implement a requirements story. Often an extrovert will start by announcing a value. Introverts can appear to acquiesce to that value, since they may be internally analyzing that value as well as others. The Planning Poker game allows introverts to have the means to ponder their responses. In Planning Poker, each estimator has a deck of cards with various values on them. To estimate a story, each person selects a card with his estimate and places it on the table face down. When everyone has chosen a card, the cards are flipped over. If the individual estimates are far apart, the estimators briefly discuss their differences. The estimators listen to the discussion to see if it alters their estimate. Each person then selects a card with his new estimate, and the cards are again flipped over. The process continues for a few rounds until there is either convergence on an estimate or an agreement that the effort required to implement the story is difficult to estimate or unknown.

There are many other areas in which differences in styles may lead to problems in communication. For example, in many agile environments, story cards capture requirements. Each card includes a brief description of the requirement and an estimate of the effort to completely implement the requirement. Judges might want preprinted templates for these story cards and a check that each card has been completely filled in. Perceivers may be happy with blank cards. The brief description can satisfy

intuitive people, while sensing people may want more details recorded on the cards.

The differences between the styles of judging types who prefer exactness and perceiving types who are comfortable with inexactness often emerge with different ways to track time estimates and progress on the completion of the story card implementation. The time needed to implement a requirement story is commonly estimated in story points. Story points represent the *relative effort* to complete a story rather than absolute time. To emphasize that story points do not correspond to exact times, the values they can take are usually limited to those in a Fibonacci series (i.e., 1, 2, 3, 5, 8, 13, ...). The time to complete a story is estimated based on *velocity* (approximate number of story points that can be implemented per iteration). While perceiving types are comfortable with that inexactness, judging types may prefer actual day or hour values. During the planning for an iteration, the tasks required to implement each story are typically estimated in hours. Having two levels of estimates—story points for rough estimates and hours for detailed estimates on tasks—can help satisfy both personality types.

Progress of story completion can be communicated in ways that suit both intuitive and sensing types. Agile teams commonly track progress of stories on a large board called the storyboard. The movement and location of the requirement story cards on the storyboard demonstrates the progress. An intuitive type can get a picture of progress with just a quick glance at the board. Sensing types typically prefer seeing a numerical tracking mechanism, such as a spreadsheet. Using the spreadsheet, they may create intricate measures of progress using graphs or formulas. Updating a storyboard and entering the details into a data-manipulation program usually can satisfy the communication needs of both types.

active listening

When you gather information that is presented orally, you can practice active listening. Active listening focuses on understanding what the other person is

saying and giving feedback on how well you understand. Clear your mind and focus on what the speaker is saying, not on what you are going to say next. You are not trying to have a debate; you are trying to understand.

If you follow what the other person is saying, periodically signal your understanding with either an oral response such as, “I understand” or a body language response such as nodding. Never pretend to follow if you really don’t comprehend what the speaker is saying. If you don’t understand, ask the speaker to expand or clarify. Simply make a request, such as “Can you give me an example?” when you feel the need for more clarity, or simply say, “I’m not following you.”

In gathering requirements or performing root-cause analysis, you might employ the “Five Whys.” The Five Whys suggests you ask a speaker “Why?” five times in a row. For example, if he states, “I like the Dallas Cowboys,” you ask, “Why?” When he responds, “Because they’re a great football team,” you ask, “Why?” And repeat this process until he gets down to the real reason, “They have the best cheerleaders.” However, asking “Why?” can put the speaker in a defensive mode. What you want is for the speaker to reflect and expand on his thoughts. So when he says, “I like the Dallas Cowboys,” you could respond, “Tell me more.” Once you have the information gained from active listening, then you may want to ask the “Whys” to help you better understand the requirement and its purpose.

An accompaniment to active listening is active writing. When taking notes, many people just write them on a pad of paper. The speaker does not get any feedback from you as to what you are capturing. You could be writing down the groceries that you want to buy at the store later.

With active writing, you use a whiteboard or a flip chart to record your notes. Those who want to save paper can write on a whiteboard and later capture its contents with a digital camera. Having your writing visible to the speaker makes apparent your understanding of his words. It also helps control the pace of the discussion. If you

Virginia Satir was a family therapist. As she became involved with other larger systems—organizations, communities, and countries—she realized that, in many cases, these systems were like big dysfunctional families, so she started to apply family therapy to them.

Virginia Satir also created the “Change Process Model” to help families process change. This model has been applied to organizations to help them deal with change, such as a traditional development group switching to agile development. The change model specifies a series of stages in how people respond to a change. In initial status quo stage, people are in the current familiar situation. When a foreign element (the change) occurs, such as the introduction of agility, then a state of chaos ensues. Chaos reigns until a transforming idea or ideas emerge, such as those coming from retrospectives. Then the integration stage begins as the ideas become embedded into the process. When the ideas are assimilated, the group is in the new status quo stage. See the StickyNotes for more information on the Satir Change Process Model.

haven’t finished capturing a statement, the speaker can see that, pause, and wait for you to catch up. The speaker also gets immediate feedback that you have captured his ideas correctly.

satir interaction model

Often we do our best to communicate with each other, and yet misunderstanding remains. Virginia Satir’s Interaction Model helps to explain why misunderstandings occur.

The Satir Interaction Model has four phases. The first is observation—what you see and hear. For example, my wife sees a pile of dirty clothes next to the laundry basket. The second phase is interpretation—the meaning you apply to your observation. She interprets the pile of clothes as my being inconsiderate, as the family rule is to put clothes to be washed in the basket. The third phase is significance—the “weight” you assign to the interpretation. She assigns a rather heavy weight to my apparently inconsiderate action. The final stage is the response—what you do about the significance of the interpretation. She directs a well-deserved (in her mind) remark to me about my inconsiderate action.

Now, reflect on an alternative sequence. She observes the pile of clothes. Her interpretation is that she is curious why I have violated a family rule that is so important to her. She assigns it a high significance and asks me for an explanation. I explain that I had been working in the woods and that the clothes have poison ivy on them. I did not put them in the laundry basket because I was going to wash them separately. With that information, the significance and her response change considerably.

When you observe a team member doing something that has a negative impact on you, start your conversation with him by stating in neutral terms your observation. Often the person may not even be aware of the action you observed. Then, ask for his interpretation. The person may have been aware of his action and done it for reasons that seemed reasonable to him. You can then describe your own interpretation and the significance you place on your observation.

For example, you may observe your



colleague not checking his code into the repository at the end of the day. You could interpret that inaction as your colleague does not care for the team, attach significance to that interpretation, and make a heated response to him. Instead start by simply stating your observation—code not checked in. Then ask for his explanation. You may find that he got an emergency call to pick up his kids from school or some other crisis.

exchange of Tokens

Communication is an exchange of tokens. The tokens or words we use in either written or oral communication have subtle differences in meanings. For example, how often is the ball kicked in football? It depends on your frame of reference. People with a background in American football might respond the ball is kicked only a few times per game. Others might say, all the time, because that’s how one moves the ball in the sport that Americans call soccer.

There also are greater differences in meaning. For example, the connotations of “done” and “I’ll have that for you tomorrow” vary widely. When you hear a developer say a feature is “done,” do you interpret that as a) the code compiles, b) the feature passes the programmer’s tests, c) the feature passes the tester’s test, or d) the customer has accepted the feature? Ask this question of your development team and see if all members respond with the same answer.

If someone says “I’ll have that for you tomorrow,” does that mean a) first thing in the morning, b) sometime during the

"some people prefer passive communications so they have time to absorb and reflect on the information. others favor passive communication because they work best in silent contemplation."

day, c) by the time he leaves work, or d) at least one second before midnight?

When someone says "I'll try," what are your expectations of how hard? Will he try a) until death takes over, b) until he is exhausted, or c) until his fingers are cramped? What are your expectations of how long? Will he keep it up a) until the task is complete, b) until something more pressing comes along, or c) until he gets bored?

With all of these different meanings, how do you avoid misunderstanding "I'll try to have that done for you tomorrow?" Ask about, and then agree on, the interpretation of these communication tokens. If the meanings are too embedded in people's minds and not easily subject to change (for example, the connotation of "done"), then create some new shared tokens with different meanings (such as "code done," "tests done," and "customer done"; or "done," "done-done," and "done-done-done").

communication modes

Active communication involves talking—usually face-to-face. Passive communication employs documents. Active communication involves two or more people working on the same task, such as discussing at a whiteboard or programming as a pair. Active communication is usually oral but also includes body movements and hand gestures. As much as we like to communicate orally, it is not necessarily the most effective means of communication for everyone. Some people read more effectively than they listen. For them, creating a docu-

ment is not done for documentation's sake. It's using the document to enhance communication.

If someone has trouble receiving oral communication, write the information down on a whiteboard and take a picture. Sensing/judging people often need more detailed information that follows a particular format. For example, a written use case that details a requirement story can help satisfy these needs.

In communicating, keep it simple. Developers often like to draw diagrams to explain a design. A diagram can illuminate how code modules relate to each other more easily than text. When many symbols are added, simplicity can be lost. Some people fall in love with all the potential symbols contained in a particular notation, such as the Unified Modeling Language. I call them the symbolologists. Instead of using a small set of symbols and adding clarifying words, the symbolologists want to use all of the correct symbols. The result is a diagram that they understand. Other people may need to decipher it by constantly referring to symbol descriptions.

Some people prefer passive communications so they have time to absorb and reflect on the information. Others favor passive communication because they work best in silent contemplation. Their productivity decreases when they have to work in an environment filled with sound. Many agile teams work in bullpens, which often have a noisy atmosphere. Overhearing conversations, which happens in many XP environments, can be counterproductive for

some people. If your team has individual cubicles, as well as a team space, those who work better in silence could have "In/Out" labels on their doors. If they are in their cubicles, they may put up the "In" label if they are available for interruption and put up the "Out" label when they are performing work that requires concentration.

summary

To ensure good communication, communicate about your communication. Take the time to understand each other's communication style. Talk about how you like to receive information, how much exactness and detail you require to be comfortable, and the meanings of the words you use. When gathering information and having discussions, practice active listening and active writing. Use the Satir interaction model to help you analyze why misunderstandings occur. Be proactive about communications to avoid a "failure to communicate." {end}

Sticky Notes

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

- Communication strategies
- MBTI type test
- Satir Change Process Model

Product Announcements

Web 2.0 BPM Suite

SUNNYVALE, CA—Vitria Technology, Inc. announces the release of M₃O, the industry's first Web 2.0 BPM suite that empowers business users to directly model, manage, monitor, and optimize their business processes.

Vitria also is introducing M₃O's Exception Manager, which utilizes all of M₃O's capabilities to automatically resolve process exceptions across the enterprise. Exception Manager is built on three generations of product knowledge in exception management and leverages the benefits of business-level modeling, policy-driven automated resolution, and standards-based technologies in a high performance, enterprise-class solution.

Visit www.vitria.com for more information.

LISA 4

DALLAS—iTKO, Inc. announces enhanced native support for enterprise IT monitoring frameworks within the latest release of its LISA 4 testing and validation solution. The new, advanced test execution capabilities of LISA allow IT

teams to generate events and transactions that appear on the monitoring framework without the need to code a test harness. LISA can also become an agent that listens to the resulting metrics, making it an integral part of the enterprise's application management process.

LISA already contains the ability to test and accept data for most common enterprise server and application metrics, such as JMX providers, SNMP data, and Windows Perfmon. LISA now ships with several out-of-the-box configurations for most major metrics providers and application servers, such as BEA WebLogic, IBM WebSphere, TIBCO, and JBoss. LISA also allows extensions to capture proprietary metrics into test execution context.

Visit www.itko.com/lisa for more information.

SaaS-Enabled Products

SAN MATEO, CA—Serena Software will offer three solutions via Software-as-a-Service (SaaS): Serena Mariner (project and portfolio management), Serena business mashups, and Serena agile lifecycle

management tools. SaaS provides significant benefits for companies—no infrastructure to manage, no software to install, faster time to value, and simple subscription pricing.

Serena Mariner provides total visibility into project status and metrics to ensure the right people are on the right projects at the right time, ultimately delivering more value to the business. Serena Mariner gives business analysts a way to manage their entire portfolio quickly and easily without creating unwieldy Excel spreadsheets. All Serena products delivered via SaaS will have published pricing and discount schedules.

Serena Mashup Composer brings drag-and-drop ease of use to lower the barrier of entry to mashup development. Soon, for a monthly subscription fee, these mashups can be deployed and run on Serena's on-demand mashup platform.

Visit www.serena.com for additional information.

Expanding Agile Horizons

LEARN HOW TO DELIVER BUSINESS VALUE WITH AGILE

Join us in Toronto for the biggest ever gathering of agile practitioners and thought leaders. This conference expands Agile from software development to delivering business value. At Agile 2008 you will discover how to put agile principles to work!

Our program is organized like a music festival that provides different stages to attract audiences with common interests, such as tools, culture, leadership & teams and user experience. Each stage will have a feel of a smaller, focused mini-conference whilst providing you with wide choice of topics to choose from. At Agile2008 we provide a program suitable for all experience levels from novices to experts.

We are pleased to announce opening keynote speaker Jim Surowiecki, author of the book "Wisdom of Crowds". Bob Martin, author of "Agile Software Development, Principles, Patterns, and Practices," speaks at our last night conference banquet open to all attendees. Our closing keynote is Alan Cooper "Father of Visual Basic" and the inventor of using personas in Interaction Design.

Early bird registration is open now! Details at www.agile2008.org



Agile2008
Conference

Toronto August 4 to 8, 2008

Rally End-to-End Solutions

BOULDER, CO—Rally Software Development Corp. announces enhancements to its products, online training center, and Web 2.0 portal, which enable customers to adopt agile practices from product idea to delivery. In addition, new enterprise-ready integrations with products from traditional application lifecycle management (ALM) vendors—including IBM, Microsoft, and HP—help drive incremental agile adoption for organizations of all sizes.

- Rally Release 2008.1 will include drag-and-drop ranking, task reordering, defect suite roll-up status, enhanced release burndown charts, expanded revision history, acceptance test authoring and tracking, and international Web caches to further enhance performance and server response times.
- Agile University, an online center for public agile training courses, now has a new administrative system for agile trainers. New administrative functions allow a worldwide network of trainers to

post and manage courses while taking advantage of full course coordination services.

- Agile Commons, a rapidly growing Web 2.0 community for agile practitioners, includes feature requests that are prioritized and voted on by customers to drive collaboration, responsiveness, and innovation. Enhancements to the Agile Commons platform include allowing read-only status on feature requests, sorting by category, and allowing members to change their votes.

Visit www.rallydev.com for more information.

Squish 3.3

GERMANY—froglogic GmbH unveils version 3.3 of the automated GUI testing tool Squish. Squish supports creating and running automated GUI tests of applications based on a variety of user interface technologies including Trolltech's Qt toolkit, Java AWT/Swing/NetBeans, Java SWT/Eclipse RCP/JFaces, Web/

HTML/AJAX, and Mac OS X Carbon/Cocoa.

One of the new features of version 3.3 is cross-technology support. While previous Squish versions worked exclusively on one specific GUI technology, the new release enables the testing of hybrid GUI applications. For example, Squish tests can automate Win32 ActiveX controls embedded in Qt applications, Java applets, Flash, or Active-X components embedded in a Web application, or mixed Java Swing/AWT and SWT/Eclipse RCP applications.

Other new features include dedicated support for interacting with Eclipse GEF elements (Squish for Java) and QGraphicsView items (Squish for Qt), support for testing Mac OS X Carbon/Cocoa GUI applications, support for testing Java applications started via Java Web Start, an integration into the Eclipse TPTP Framework, and several improvements to enhance the tool's ease of use and test maintenance.

Visit www.froglogic.com for additional information.

Skytap™ VIRTUAL LAB

» Experience the freedom of cloud-based virtual labs

Skytap Virtual Lab is a revolutionary virtual lab automation solution available as an on-demand service over the Web. It gives you the freedom to:



» Rapidly build and configure test environments using Skytap's pre-built virtual image library



» Access lab hardware over the web, in minutes



» Collaborate with other team members to resolve defects, wherever they are in the world

Learn more at www.skytap.com, or visit booth #52 at STAREAST, May 5-9, 2008 • Orlando, Florida

Skytap, Inc. TOLL-FREE: +1-888-SKY-TAP8 • DIRECT: +1-206-866-1162

 www.skytap.com

1 Things You Might Not Know About

Being a Successful Test Manager

by Rick Craig

1

MOTIVATE YOUR STAFF. Testers who are excited and committed to the testing effort will be far more productive and effective than those who are not. New test managers should motivate their employees by showing them respect, asking their advice, giving them challenging assignments, providing meaningful training, rewarding their efforts, and banishing “zero defect” mentality.

2

FIND A SENIOR MANAGEMENT MENTOR. A senior manager acting as a testing advocate for both you and your testing group can help obtain buy-in from other senior managers on the value of testing and its role in the entire organization. Additionally, a senior management mentor can help grease the skids when test managers need assistance in budgeting, training, conflict resolution, etc.

3

TREAT THE DEVELOPERS LIKE YOUR CUSTOMERS. Test managers need to forge relationships with the developers, requirements specifiers, etc. Since we are measuring the quality of the product created by the developers, they, too, are interested in our results and should be treated as customers. Remember, when all is said and done, the developers and testers usually have the same goal: shipping a useful, high-quality product in a timely fashion.

4

DEVELOP A STRATEGY. Most people immediately think of a formal document when the word strategy is mentioned, and indeed that may be required. In other cases, however, the strategy might be notes on a whiteboard or simply a conversation. In strategy decisions, test managers should include all stakeholders—developers, users, testers, etc. Examples of strategy topics to discuss include: how to choose tests, how to determine relative risk, contingencies, scheduling, staffing, CM, metrics used to measure testing status, effectiveness, etc.

5

CHOOSE THE METRICS YOU NEED TO MANAGE TESTING, BUT DON'T GO OVERBOARD. Test managers need basic metrics to measure test effectiveness, status, resources, and the quality of the product being tested. Managers may very well determine that they need more metrics than these, but it is usually better to start simple and add metrics when their need is actually identified. Discontinue metrics that are currently being collected but are unused.

6

AUTOMATE WHEN POSSIBLE, BUT DON'T FALL IN LOVE WITH YOUR TOOLS. Tools have the ability to make testing groups more effective and efficient and should be used when the need for these tools has been justified. The test manager should ensure that the tool helps him implement his strategy. The tendency to use a tool just because you bought it is not normally fruitful.

7

MEASURE THE EFFECTIVENESS OF YOUR TESTING. If a test manager complains of lack of time and/or resources for the testing effort, my first question is always “How effective is your testing?” New test managers should choose at least two different metrics—e.g., functional coverage, code coverage, DDP, and defect age—that help measure the effectiveness of the testing effort.

8

INVEST IN YOUR TEST ENVIRONMENTS. Many new test managers will learn quickly that they spend an inordinate amount of time lobbying for resources for test environments and determining how to create and populate them, refresh them, etc. At the very least, this is inefficient. Some managers may even discover that the inaccuracies in their environment invalidate some of their testing or at least make the results suspect.

9

GET A “SEAT” ON THE CHANGE CONTROL BOARD (CCB). All too often, testers are omitted from the process of determining the relative effort to implement fixes. While testing, testers learn a great deal about the systems, and this insight is useful in determining the priority of fixes and enhancements. If your company has a formal CCB, the test manager should be a participant on the board. If there is no formal CCB, then the test manager should take part in the informal decision-making process of which bugs and enhancements should be given the highest priority.

10

CONTINUOUSLY IMPROVE THE WAY YOU TEST. Organizations are rarely static, so test managers should be committed to a continuous process improvement initiative. Any improvement initiatives should include participation by as many team members as possible. Needed improvements can be identified during post-project reviews or by using formal techniques like test point insertion.

When to Step Up, When to Step Back

by Pollyanna Pixton

From day one as a developer, I was asked by leaders to do things that I knew were a waste of time—keeping me from getting the “real” work done. When I became a project manager, those leaders continued wasting my time with questions such as: Where are we? When will we finish? How many errors are there? How much will it cost? But when I became the dreaded leader, it all became clear. When developing products, leaders need to know things to meet customer needs in the optimal market window so the company can continue to exist and hopefully prosper.

Leaders can stifle progress when they interfere with team processes. But as a leader, you don’t want an on-track project to go over the cliff and deliver the wrong results. There are times when leaders should stand back and let the team work—and times when leaders should step up and lead. How do we decide which is which?

Let’s start with stepping back. You hired your staff members for their abilities to address issues, solve problems, and create innovative and competitive products. Put your people to work improving operations, increasing workflow, and removing bottlenecks. They operate closest to the problems and have the best chance of finding workable solutions. You hired these people to deliver new, exciting, and competitive products to the marketplace before your competition can.

As their leader, you must unleash this talent and allow the team to succeed. Create an environment based on trust, bring together people who have the right knowledge, ensure they understand the objectives, purpose, and constraints of the project—and then step back.

As a contributing team member, I want to take pride in my work. I want to use my knowledge and experience to develop and deliver my part of the product. I want to create the best and most efficient solution, and I want to collaborate with my teammates to make sure we all deliver by assisting each other when we need help. I am not happy when someone begins to tell me how I should do my job or when I don’t get the opportunity to find and correct my own mistakes. That’s how I learn and gain more experience and knowledge. So, I need leaders to stand back and let me get the work done. What about accountability? I want to be held accountable by the people who understand what I am doing and how I think and who share in the challenges of the team. In other words, I want my teammates to hold me accountable—and they do.

But there are times when leaders need to step up and lead. As leaders, we want our projects to deliver results, but not just any results, the *right* results—results that are inline with the corporate strategy. You could just lecture team members on this and hope

“Another key to success: Don’t rescue your teams—but don’t let them flounder for too long, either.”

they get it. More effective than lecturing, though, is continuously asking questions to help them discover the answers. Questions like: How does this feature/objective fit with our company strategy? If it does not, do we need to modify our strategy? Does our prioritization scheme match our business priorities? Can we reach our market window?

Another key to success: Don’t rescue your teams—but don’t let them flounder for too long, either. It can be difficult to know when the team needs more time to struggle toward a solution or needs you to add some leadership (not step in!). How do you know when to step up? Through your own experience and intuition. If you ask inquiring questions rather than tell the team what to do, you will maintain the integrity of its own problem-solving process. So, when you get a sense that team members are thrashing, bring them together. Do not ask what’s wrong (this may pass

as judgment that they cannot overcome). Do not ask them where they are stuck (they might not be; they might only need a new view). Ask team members to tell you about the project, their approach, their ideas, and their solutions so far. Help them take a new and fresh look. Often, this is not easy. When leaders hear a problem and know the answer, they will want to give the answer. Once you do this, you are cooked. Your effectiveness as a leader diminishes. Suddenly, a steady stream of people will be in your office asking you to fix their project and give them the solutions to all their problems. You have taken away the team’s ownership and, in effect, told team members they are incapable of solving their own problems. They will lose pride in their work and their productivity will drop off dramatically.

Allow your teams to manage their workload, find solutions, and deliver. Make sure all team members understand that their solutions, objectives, and goals must be in line with the company strategy. Help your teams find their own solutions—but only after they have tried on their own. Don’t be too quick to give them solutions. Ask questions without giving answers. Your questions will help your team members discover their solutions. Use your questions to unleash the talent and creativity in your organization. Then stand back, get out of the way, and let them get the “real” work done. **{end}**



StickyMinds.com

*Map out your project success using
StickyMinds.com as your resource guide*

Benefits of joining StickyMinds.com:

- Access daily news articles geared to our industry
- Have any of our six eNewsletters delivered straight to your inbox
- Enjoy Podcasts and Videocasts featuring industry experts
- Find software solutions in the comprehensive Tools Guide
- Search and read book reviews
- Post a question to your colleagues on our Discussion Board
- Find the resources you need for building better software
- Submit articles or technical papers for others to download

StickyMinds.com is the Web's first and most popular interactive community exclusively engaged in improving software quality throughout software development.

Membership is free, so sign up today and start your journey to building better software! www.StickyMinds.com/join

StickyMinds.com

Index to Advertisers

Aculis	www.aculis.com	30
Agile 2008 Conference	www.Agile2008.org	44
AutomatedQA	www.testcomplete.com/try	Inside Back Cover
Blackbaud, Inc.	www.blackbaud.com	21
Empirix	www.empirix.com	1
Hewlett-Packard	www.hp.com/go/software	24
Hewlett-Packard	www.hp.com/go/software	Back Cover
IBM	www.ibm.com/rational	14
Parasoft	www.parasoft.com/soaqualitysolution	Inside Front Cover
PNSQC	www.pnsqc.org	36
Pragmatic Software	www.SoftwarePlanner.com	23
Rally Software	www.rallydev.com/bsm	11
Seapine	www.seapine.com	2
Skytap	www.skytap.com	45
SmartBear	www.codecollab.com	31
SQE Better Software Conference	www.sqe.com/BetterSoftwareConf	5
SQE Certification Training	www.sqetraining.com/Certification	37
SQE Testing Training	www.sqetraining.com/Testing	12
STARWEST 2008	www.sqe.com/StarWest	16
StickyMinds.com	www.StickyMinds.com	48
Web Performance, Inc.	www.webperformance.com	19

Display Advertising Shae Young young@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.

Checking Automated Testing Prices?



TestComplete™

Sensible price. Superior automated testing.

Get superior automated testing software for a fraction of what the big guys want to charge you. TestComplete is an award-winning testing solution that gives you functional, regression, unit, load, client server testing and more in one low priced package. It tests any of your Windows applications: Desktop or Web, C++ or VB, .NET or Java, Flash or WPF. Don't throw your money away on overpriced software that does less than TestComplete. Download the free trial of TestComplete now!

- ✓ Award-Winning Features
- ✓ Easy Record & Playback
- ✓ Great Object Recognition
- ✓ Unlimited Extensibility



FREE TRIAL - DOWNLOAD NOW
www.testcomplete.com/mad

AutomatedQA
test, debug, deliver!
(978) 236-7900



ALTERNATIVE THINKING ABOUT APPLICATION SECURITY:

Hone Your Threat Detection (To A Telepathic Level).

Alternative thinking is attacking your own Web applications, finding vulnerabilities and destroying them with precision and vengeance—throughout the life of the application.

It's looking at application security through the eyes of a hacker to identify threats to your system and risks to your business.

It's harnessing the power of SPI Dynamics, recently acquired by HP, to redefine and expand your security abilities. (Please note: positive effects on your bottom line.)

It's assessing security the right way, from development to QA to operations—without slowing down the business.
(Cue elated cheers.)

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

