

September/October 2009

\$9.95

www.StickyMinds.com

# BETTER SOFTWARE

**ROOM TO GROW**  
Cultivating test diversity

**STAKEHOLDER  
REQUIREMENTS**  
Think globally and locally

The Print Companion to

**StickyMinds.com**

**SOFTWARE  
LONGEVITY  
TESTING**  
PLANNING FOR  
THE LONG HAUL



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

# Cut Dev Costs... Without Cutting Heads.



*It's tough to look  
at your team and  
decide who should  
stay and who  
should go.*

Rally's customers are proven to be  
50% faster to market and  
25% more productive.

Avoid cutting heads with the only  
Application Lifecycle Management provider  
to ensure your Agile success.

**Learn More about our Guarantee Success Program  
at [rallydev.com](http://rallydev.com)**



Scaling Software Agility

© 2009 Rally Software Development Corp



# Save time while protecting software quality.



© 2009 Seapine Software, Inc. All rights reserved.

## Swiftcover.com cut their testing time in half with TestTrack Studio and QA Wizard Pro, while still providing the quality their customers expect.

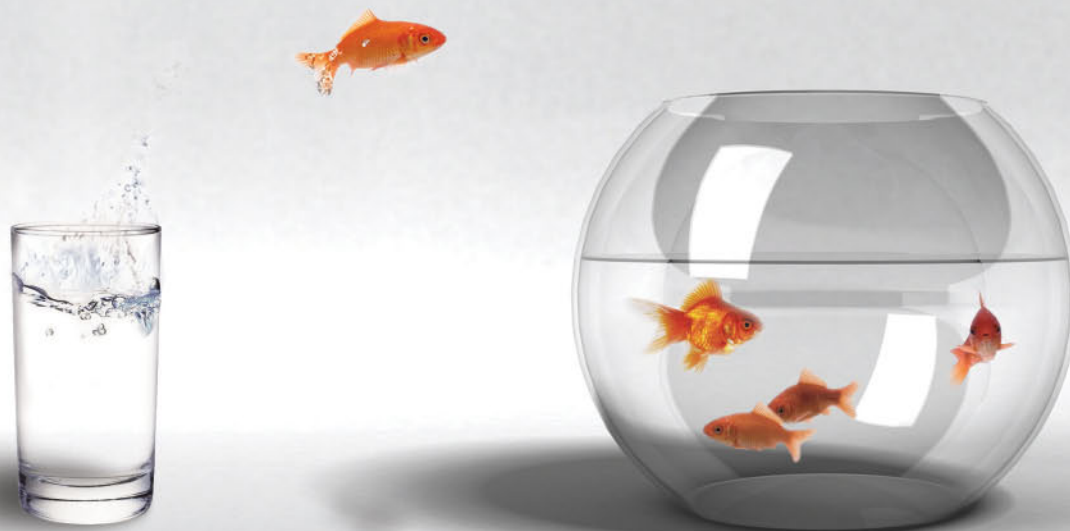
Seapine's end-to-end Software Quality Assurance (SQA) solutions help you deliver quality products faster. Start with **QA Wizard Pro** for automated testing and add **TestTrack Studio** for issue tracking and test case management—integrated quality assurance solutions that together reduce testing time, saving you money and improving customer satisfaction.

- Reduce quality assurance costs with automated functional and regression testing.
- Manage test case development, defect assignments, and QA verification with one application.
- Track which test cases have been automated, schedule script runs, and link test results with defects.

“*So much of success boils down to time. QA Wizard Pro and TestTrack Studio allow us to be more profitable because we do more in less time.* — Test Manager, Swiftcover”

Learn how to test faster while protecting quality. Visit **www.seapine.com/betterswift**

# Make the leap to Risk-based Testing and eliminate unnecessary compromises



Lengthy development cycles and greater time-to-market pressures, leave testing teams with little or no time to ensure complete test coverage. Cognizant's risk-based approach helps you optimize testing efforts while ensuring predictable quality and business readiness of applications.

Built on statistical models, our risk-based testing framework helps you categorize and prioritize testing based on business risk, thereby helping you to optimize cost and accelerate time-to-market.

With Cognizant, you get certified continuous business readiness, lower costs and accelerated results unmatched in the industry.



**Cognizant** | Testing Services  
Passion for building stronger businesses

#### World Headquarters

Cognizant Technology Solutions  
500 Frank W. Burr Boulevard  
Teaneck, NJ 07666  
Ph: +1 201 801 0233  
Fax: +1 201 801 0243  
Toll free: +1 888 937 3277  
Email: [inquiry@cognizant.com](mailto:inquiry@cognizant.com)



## Cover Story

### SOFTWARE LONGEVITY TESTING 18

How long do you let your software run during testing? An increasing number of software applications are intended to run indefinitely, in an always-on operating environment. And yet, few test plans include more than a brief memory leak test case. Learn how to test for problems due to the passing of time and problems due to cumulative usage. *by Steven Woody*

## Features



## Columns & Departments

### In Every Issue

Editor's Note 4

Contributors 6

Product Announcements 35

10 Things You Might  
Not Know About ... 41

Ad Index 44

**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 six issues.**

Visit [www.BetterSoftware.com](http://www.BetterSoftware.com)  
or call 800.450.7854.

### IT TAKES A VILLAGE 24

Scrum emphasizes the product owner role as a single voice managing a single backlog for an agile team. Yet for many organizations, a single-product-owner-per-team model fails to support the whole need in terms of defining value. Learn six ways to fill the product owner role, in some cases using a “village.” *by Ronica Roth*

### IDEs AND BUILD SCRIPTS 30

Teams benefit from using both IDEs like Eclipse and integration tools like Maven. Understanding the risks that can occur when IDEs and build scripts diverge and the guidelines for keeping the two consistent can increase team productivity. *by Steve Berczuk*

### TECHNICALLY SPEAKING 9

#### Scrumdamentalism • *by Lee Copeland*

It's been said that, over time, charismatic movements often evolve to become “bureaucratic”—focused on a set of standardized procedures that dictate the execution of the processes within the movement. Has Scrum evolved to this point, or is there still a place for agility in our processes?

### INSIDE ANALYSIS 10

#### The Whos and Wheres of Stakeholder Requirements • *by Mary Gorman*

Whether you're working on a collocated or a distributed team, it's important to take stakeholder requirements into account: Who are they and where are they located? These tips can help you narrow the gap between thinking and acting globally and locally.

### TEST CONNECTION 12

#### Food for Thought • *by Michael Bolton*

Ideas about testing can come from many different and unexpected sources, including reductionism, agronomy, cognitive psychology, mycology, and general systems. Michael feasts on Michael Pollan's *The Omnivore's Dilemma* and finds much to whet the tester's appetite for learning about how things work.

### MANAGEMENT CHRONICLES 14

#### Rescuing a Captive Project • *by Fiona Charles*

Allowing an individual to hold a project hostage to his knowledge and expertise is bad for the project and for the team. Fiona Charles describes one captive project and shows how it could have been remedied.

### THE LAST WORD 43

#### The State of the Practice • *by Ross Collard*

While software testing focuses on detection rather than prevention, we can argue that it has become a powerful counteroffensive against bugs. We can equally argue that many of today's software practices impede quality. Ross Collard compares these two positions and invites you to join the discussion.

### A LITTLE HOUSEKEEPING



This month, I'd like to call your attention to a change to our Fresh Ink eNewsletter and tell you about a couple of opportunities to take part in industry-specific surveys.

As we all know, the economy has been less than stellar this year, and we all are wondering how our industry—especially our jobs—have been affected. So, what better time to take the pulse of the industry with our annual Salary Survey?

I invite you to follow the link below that best describes your employment level, answer a few questions, and contribute to the collective knowledge of industry salaries and employment trends.

Survey results will be published in the November/December digital edition. Of course, your responses are confidential.

STAFF LEVEL: [www.stickyminds.com/2009salarysurveystaff](http://www.stickyminds.com/2009salarysurveystaff)

MANAGEMENT LEVEL: [www.stickyminds.com/2009salarysurveymanagement](http://www.stickyminds.com/2009salarysurveymanagement)

DIRECTOR LEVEL: [www.stickyminds.com/2009salarysurveydirector](http://www.stickyminds.com/2009salarysurveydirector)

Speaking of the digital edition, I want to let you know that starting with the July/August issue we streamlined some processes on our end, which changed the way you receive the digital edition. Fresh Ink, our eNewsletter that provides an overview of content from the latest issue of *Better Software* magazine, is now delivering the digital edition to your inbox.

And, finally, in this issue's The Last Word, Ross Collard presents his thoughts on the state of software testing and quality. He invites readers to participate in the discussion by completing a short survey at [www.stickyminds.com/testsurvey](http://www.stickyminds.com/testsurvey).

As always, I hope you enjoy this issue of *Better Software* magazine. Drop me a note to let me know how you've put this issue to work for you.

Happy Reading!

Heather Shanholtzer

HShanholtzer@sqa.com

## BETTER SOFTWARE

Publisher

**Wayne Middleton**

Vice President of Publishing

**Holly N. Bourquin**

Editor in Chief

**Heather Shanholtzer**

### Editorial

Managing Technical Editor

**Lee Copeland**

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

Circulation Coordinator

**Jamie Green-Gago**

Marketing Coordinator

**Stephanie Fender**

A PUBLICATION OF  
SOFTWARE QUALITY ENGINEERING



### CONTACT US

Editors: [editors@bettersoftware.com](mailto:editors@bettersoftware.com)

Subscriber Services: [info@better-software.com](mailto:info@better-software.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



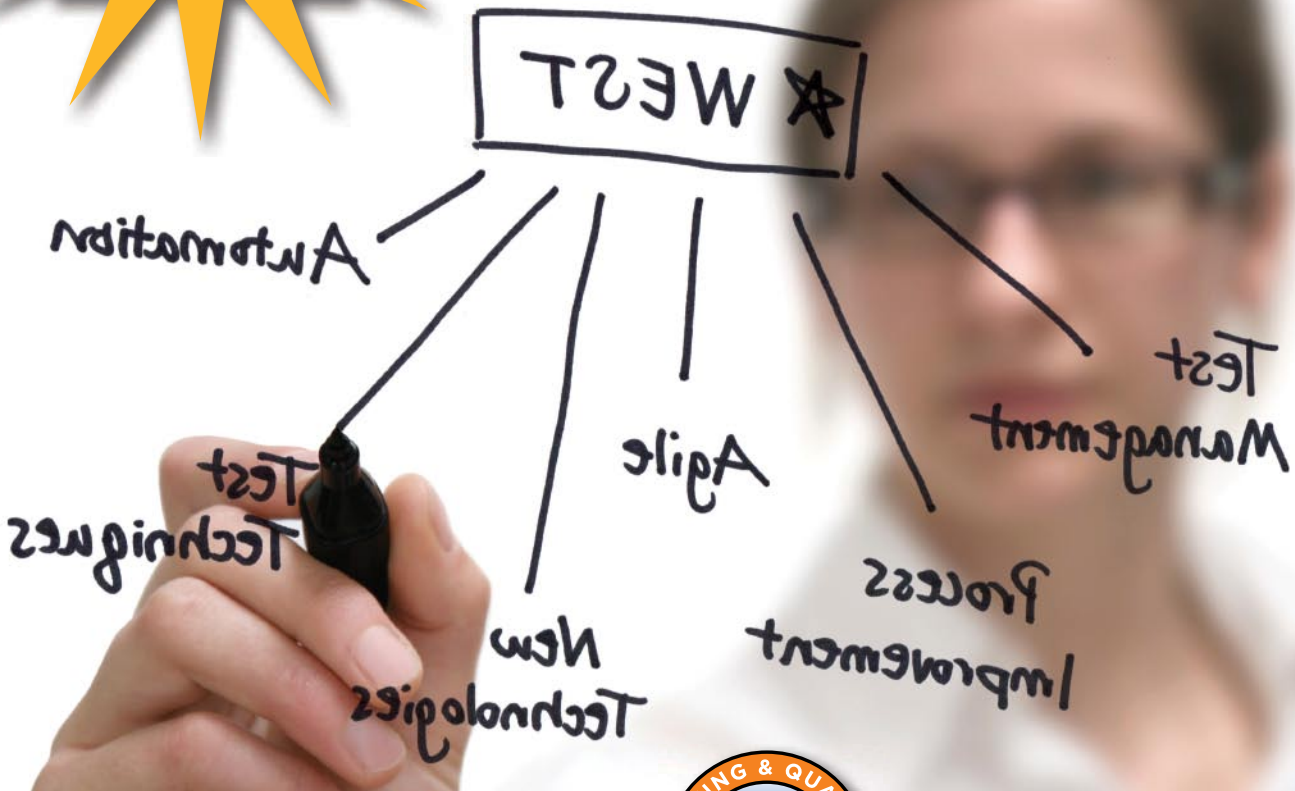
**STAR  
WEST**

# SOFTWARE TESTING

ANALYSIS & REVIEW

OCTOBER 5-9, 2009  
ANAHEIM, CALIFORNIA  
DISNEYLAND® HOTEL

*The Greatest Software Testing Conference on Earth*



**New for 2009!**

## Testing & Quality Leadership Summit

### You Can't Afford to Miss STARWEST 2009

- Participate in the broadest and deepest range of learning sessions available
- Cultivate relationships with peers and join the tester community
- Visit the largest testing EXPO anywhere
- Gain insight from top industry experts and speakers
- Network with peers and industry leaders

**99% OF 2008 ATTENDEES  
RECOMMEND STARWEST TO  
OTHERS IN THE INDUSTRY**



**[www.sqe.com/starwest](http://www.sqe.com/starwest)**

MAXIMIZE YOUR DISCOUNTS AND SAVE UP TO \$600.  
GROUPS OF 2 SAVE EVEN MORE!

#### Conference Sponsors:



#### Media Sponsors:



## Contributors



**STEVE BERCUK**, a Certified ScrumMaster and coauthor of the book *Software Configuration Management Patterns*, is an expert on helping agile teams use software configuration management effectively. Steve is an engineer at Health Insight Technologies in Boston. His Web site is [www.berczuk.com](http://www.berczuk.com), and you can email him at [steve@berczuk.com](mailto:steve@berczuk.com) or follow his blog at [www.berczuk.com/blog](http://www.berczuk.com/blog).



**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](mailto:mb@developsense.com).



**FIONA CHARLES** is a Toronto-based test consultant and manager with thirty years of experience in software development and integration projects. Through her company, Quality Intelligence, Inc., she works with clients in diverse industries to design and implement pragmatic test and test management practices that match their unique business challenges. Contact Fiona via her Web site: [www.quality-intelligence.com](http://www.quality-intelligence.com).



An experienced teacher, consultant, and author, **ROSS COLLARD** has advised senior managers of many well-known enterprises on software quality improvement and effective testing practices. He has trained more than 10,000 professionals in software testing and quality, and his materials have been used to train another 60,000. You can reach Ross at [ross@rosscollard.com](mailto:ross@rosscollard.com).



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



**MARY GORMAN**, CBAP<sup>TM</sup>, is senior associate at EBG Consulting and helps project teams explore, analyze, and build robust business and system requirements models. Mary serves on the Business Analysis Body of Knowledge committee of the International Institute of Business Analysis and is the leader of the elicitation subcommittee. She can be reached at [mary@ebgconsulting.com](mailto:mary@ebgconsulting.com) and [ebgconsulting.com](http://ebgconsulting.com).



**DALE PERRY** has been a programmer, analyst, database administrator, project manager, development manager, tester, and test manager. Dale's thirty years of project experience include large systems development and conversions, distributed systems, and online applications. He has been a professional instructor for more than fifteen years, including eleven years with Software Quality Engineering, and has presented at numerous industry conferences on development and testing.



**RONICA ROTH**, CST, is an agile coach and consultant with Rally Software. She has worked with dozens of teams and organizations to help them learn the agile practices that enable software development focused squarely on delivering real value and quality. Ronica's ten-plus-year software career has centered on facilitating clear communication and close collaboration across disciplines, helping people work together creatively and efficiently to develop high-value software.



**STEVEN WOODY** has more than eighteen years of experience testing embedded software for networking, telecommunications, global positioning system receivers, and communications satellite telemetry and command systems. He has worked at various companies ranging from the Fortune 500 to startups. Steven has presented at software testing peer workshops and has contributed to books on software testing. Contact Steve at [stevenwoody1@gmail.com](mailto:stevenwoody1@gmail.com).





# mingle<sup>®</sup>

Agile Project Management

**Your global  
project team  
on the same page**

Agile Project Management  
Software from the Pioneers of Agile



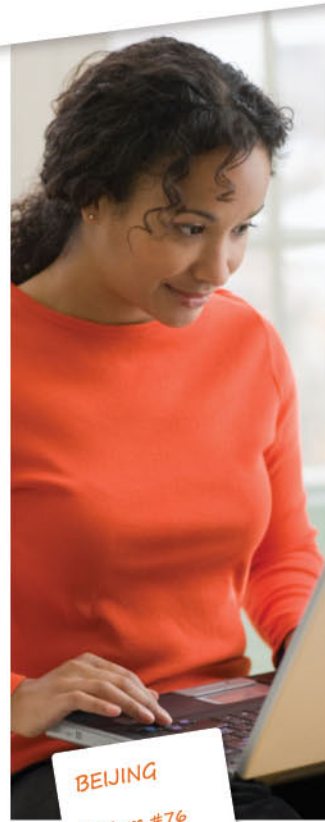
**BOSTON**

Card #54  
"Under  
Development"



**BANGALORE**

Bugfix #31  
"In Testing"



**BEIJING**

Feature #76  
"done, Done,  
DONE"

Provide a shared  
workspace for Developers,  
QAs, BAs, Customers & PMs

Capture & Visualize  
all project activity

Manage XP, Scrum,  
Lean & Agile Hybrid  
projects

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

Leverage the industry's  
best User Interface

**DOWNLOAD FREE TRIAL AT**  
[www.thoughtworks-studios.com](http://www.thoughtworks-studios.com)

 **ThoughtWorks  
studios**

Copyright © 2009 ThoughtWorks, Inc. All rights reserved.

# What is the **Future** of Your Testing Department?



**Off-shore** test engineers



**Automate** test execution



What's **next?**

In the next few years you are likely to see more product complexity, shorter product development cycles, and ever-present pressure for improved test quality.

Are your current methods for functional testing up to the task?

Off-shoring staff helped, but throwing more bodies at tests is still expensive. Automating test execution is

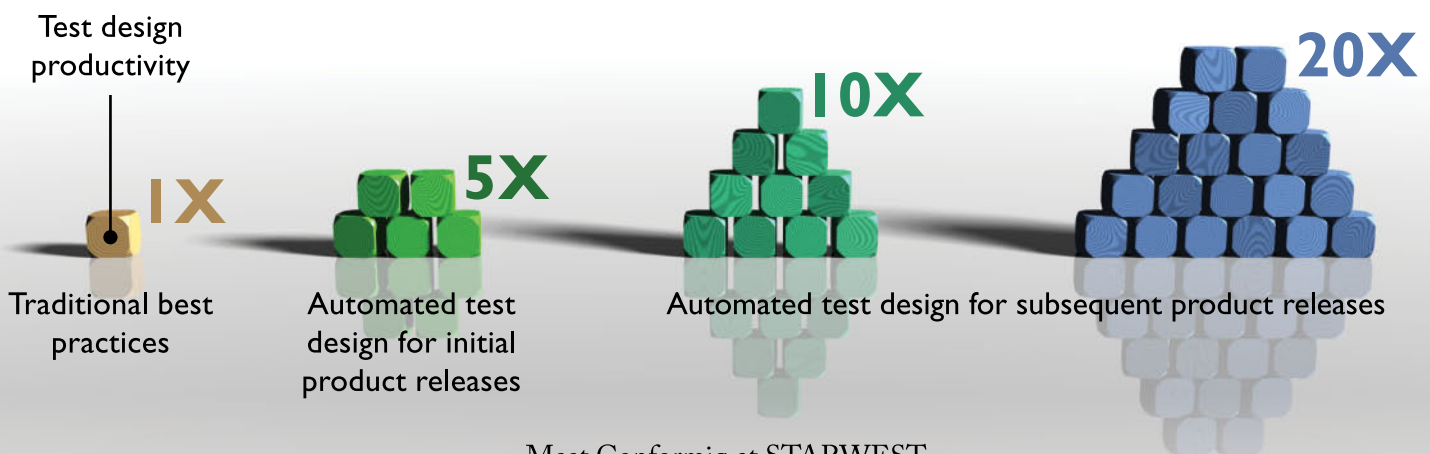
good, but now you have a test automation backlog. What's next?

Maybe it's time for you to consider moving automation further up the chain and automating the actual test design. *Automated Test Design* (ATD) technology has advanced considerably and brand-name companies are using ATD today with great success—resulting in 5x to 20x productivity gains in test design. These numbers

are based on customer benchmarks against today's industry best practices in advanced verticals such as networking and telecommunications.

If you want to be prepared for significant test methodology improvements by 2011, then you need to start today.

Let us show you how other companies are implementing the future of test design now!



Meet Conformiq at STARWEST  
or call for a private consultation and demo:

Clark Cochran | Sales Director North America & Asia  
clark.cochran@conformiq.com | +1 (408) 205 5155

Per Nilsson | Sales Director Europe  
per.nilsson@conformiq.com | +46 (70) 884 2202

Automated Test Design™

**CONFORMIQ**

www.conformiq.com

# Scrumdamentalism

by Lee Copeland

*“Every Scrum sprint must finish by delivering new product functionality that is integrated, fully tested, and potentially shippable.” [1]*

*Fundamentalism is “clinging to a stubborn, entrenched position that defies reasoned argument or contradictory evidence.” [2]*

Recently, I received an email from my old friend Jake\* who wrote, “I have just come across a Certified Scrum Master who is pitching one of my very large clients that they should be doing ‘testing sprints’ after they do ‘development sprints.’ I can’t imagine a situation where this makes sense. It violates the Scrum notion of ‘done.’ I have seen this practiced by some of my clients, and it about killed them. Lee, are you guys pitching something like this, or is this just a misunderstanding?”

While we at Software Quality Engineering haven’t advocated this, several speakers at Software Quality Engineering conferences have proposed separating testing from development in some Scrum sprints under certain special conditions.

One speaker, Jared\*, has written, “In some Scrum projects, an iteration will be dedicated to hardening the codebase. A hardening iteration focuses on bug fixing. Hardening iterations can be used to explore how applications work on different platforms, in different development contexts, or when different co-resident third-party applications are running at the same time.”

Alex\*, another speaker, has written, “The agile intent is to perform all testing within the iteration—usually via automation. Unit, functional, and acceptance [testing] are the usual focus, but what about the other forms of testing? For example: full legacy regression, interoperability across sprints, performance, usability, etc. Thus the rub!” Alex suggests two strategies—“Extending the iterative model to accommodate testing via stabi-

lization (hardening) sprints” or “skewed testing sprints.” Alex continues, “Both strategies are typically used for domains with an existing large-scale legacy [manual] testing burden (or requiring broader-scale testing practices). In the latter strategy, the skew allows some development activity to progress while broader testing is performed.”

I shared Alex’s ideas with Jake who replied, “I think that this is enough to get him [Alex] de-certified. As a Certified Scrum Trainer, I have certain responsibilities regarding issues such as this.”

So, I wondered—do Certified Scrum Trainers have both a right and responsibility to seek out heretics and silence them, demanding that they change their views and their presentation materials? Or can Scrum accommodate different views on the proper goal and structure of sprints? Not being an expert on all things Scrum, I asked my friend Milo\*.

“This is nuts. Many people new to Scrum think we should have testing sprints. No Scrum trainer teaches that we should, and there are more than one hundred Scrum trainers, so this is one of the few things on which we all agree. Scrum gets its name from the idea of a cross-functional team, and this blows the cross-functional team apart. Also, [Alex] is wrong in what he says agile teams don’t do. Many do quite a bit of integration testing, and all focus heavily on regression testing—if the teams are any good at all. He really is quite wrong.”

That sounds definitive, but I asked my friend Johnny\* to comment. “I recommend the same sorts of things [as testing sprint advocates Jared and Alex]. For testing or stabilization sprints, everyone on the team tests. In fact, they found testing to be one of the most fun things after I taught them some basic exploratory testing skills. The project manager, product owner, business analyst, developers, testers—everyone tested. The programmers would take turns taking time off to bug fix while the rest of us tried to

find failures in their fixes. I also have done this to satisfy regulatory requirements. You do your regular testing during development; then, you have a regulatory-compliant, testing sprint where results are tracked for the regulator.”

It is not my intent to resolve this issue. In fact, I present this only as an illustration. Max Weber, the eminent German economist and sociologist, observed that, in their beginnings, many movements are “charismatic.” [3] The most effective ones begin with both a noble goal (eliminate polio, save the whales, feed the children) and a charismatic leader. The agile movement is no different. It has its noble goals of rapid delivery, collaboration, teamwork, and quality, and its own charismatic leaders.

Weber claims that, over time, charismatic movements often evolve to become “bureaucratic”—focused on a set of standardized procedures (rules) that dictate the execution of the processes within the movement. Often, formal division of powers, hierarchy, and relationships are created. The hierarchy of Scrum certifications is an example. This emphasis on following the rules often supersedes achieving the original goals. New and useful ideas that challenge the standardized procedures are summarily rejected.

It is curious that while the Agile Manifesto values “individuals and interactions over processes and tools” and “responding to change over following a plan,” Scrum seems to have become *less* agile in its emphasis on following the rules, regardless of development context or value achieved. {end}

## REFERENCES

- [1] Schwaber, Ken and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.
- [2] Dawkins, Richard. *The God Delusion*. Bantam Press, 2006.
- [3] Bendix, Reinhard. *Max Weber: An Intellectual Portrait*. University of California Press, 1977.

\*Not his real name



# The Whos and Wheres of Stakeholder Requirements

by Mary Gorman

Whoever said “Think globally, act locally” hasn’t done a project complicated by geographically dispersed stakeholders. Faced with multiple languages and cultures, you need to think and act globally and locally when you’re analyzing and delivering products. Here are some tips.

## Stakeholder Requirements: Who x Where = Complexity<sup>2</sup>

Recently, I’ve wrestled with complex stakeholder requirements on several projects. One required two organizations (in different countries) to merge two products into one. Another involved enhancing existing products, and yet another needed to develop training software for technicians across the globe. Complicated enough for you? And—oh yes—some of the project teams had outsourced the software development or business processes to other countries.

Did I mention that each project had thousands of stakeholders from numerous countries? And, of course, they had divergent needs.

Multiplying who (stakeholders) by where (locations) increases the complexity of any project exponentially. But you need to dive right in. Delaying discovery and analysis of the stakeholders’ requirements can lead to unanticipated and expensive consequences.

## On Your Mark

How do you start? First, clarify what “stakeholder” means on your project. A standard definition is anyone who affects or is affected by the product: sponsors, product champions, users, advisers, and providers (developers, project managers, testers, and sources of packaged solutions who produce or provide the software product based on your requirements). Essentially, you’re answering the who question: Who is involved in this effort?

At almost the same time, ask the where question: Where are stakeholders

located? If they’re spread across multiple locations, especially internationally, it will complicate analysis. Factoring in differences in language, customs, regulations, and time zones requires additional time and expertise.

Analyzing your stakeholders leads you to explore other requirements perspectives, such as how (processes), what (data), and why (rules). Here, I focus on tips for the who and where perspectives based on my recent work.

## The Foundation

Without a doubt, the most fundamental requirements tool is a glossary of key business terms, including definitions, synonyms, and acronyms. Concise, clear definitions help diverse stakeholders work with shared meanings. If, as in my projects, your stakeholders have different native tongues, provide definitions in all those languages. Your investment in creating and maintaining a glossary will quickly pay off in quality, consistency, and speed of communication and requirements development.

## A Picture Is Worth ...

Whenever possible, use visual models to elicit, organize, and communicate complex requirements. Models can be especially valuable when you’re communicating across languages. For example, a hierarchy model visually expresses what might take several pages of text to describe. Here are key techniques and models I use when analyzing stakeholders:

## MODELING USERS

When you’re working with product managers, a great model is personas—fictional archetype users that represent a composite of similar users. Personas



ISTOCKPHOTO

quickly help you identify and evaluate your product’s users. You could start top-down by modeling a “universal” persona and then look for regional-specific variations. Or, model them bottom-up by asking subject matter experts (SMEs) to identify local versions of personas, then work collectively with all the SMEs to identify their common characteristics and create an abstract universal persona.

Alternatively, consider user roles—people or things that interact with the software product to fulfill a task. Be sure to include direct users (who initiate actions) and indirect users (who get or respond to system behavior).

Producing a user role map reveals the visual organization of the various roles. As with personas, noting where these user roles are located can lead to identifying regional differences.

Whether you use personas or user roles, these models provide a valuable basis for eliciting, prioritizing, and validating requirements.

## MODELING USER ACTIONS

Scenarios or stories are valuable tools for stakeholder analysis. A story, such as “As a book buyer, I want to return the book I bought so I can get a credit on my credit card,” starts with who initiates the interaction. Scenarios, which are broader in scope than stories, also begin with who—such as “customer returns a purchase.”

Remember to dig to discover any indirect users who participate in producing

the outcome. Your product may interact with another system (in the sample story, the credit card company could be an indirect user) or may send information to another stakeholder. Often, analyzing where the action takes place leads to the discovery of new business rules and data requirements.

Prototyping a product or a user interface gives you a visual way to elicit and confirm a stakeholder's needs. Ask questions—Who is allowed to use the interface? What is their authority level? Ask which data and business rules are specific to each who. Learn where they will use the interface by studying their physical environment—a desk, a factory floor, a moving vehicle, outdoors (with weather challenges).

Include culture-specific requirements. For example, on a user interface for Western cultures, you position fields left to right. Rules about data may vary based on location. Leverage international standards wherever possible, such as adopting ISO 8601 for numeric dates and time.

## Regulators as Stakeholders

Regulators certainly affect the product! Identify whos such as regulatory bodies and legal authorities, and specify where their jurisdictions apply.

Don't wait. One of my clients documented its software requirements in one language and then later learned it had to translate the requirements into the language specified by a national regulatory commission. Ouch!

And be sure to analyze who must enforce the policies, rules, and regulations. You may uncover new user roles.

## Providers as Stakeholders

Providers have a vested interest in the success of the product. On a recent project, the product managers' responsibilities were expanded, and they found that specifying "fast" response time and a "user friendly" interface was not sufficient for providers to deliver the right product. The entire team realized that it needed to set realistic expectations for all parties by defining early on how much detail was needed and clari-

fying who would uncover and specify these details.

As with many distributed projects, the developers (the providers) were located in countries different from those stating the requirements. The solution was for the product managers to take the first crack at stating specific quality attributes and then to work with developers to finalize the details.

## It's All in the Numbers

Whether you're working with collocated resources or a distributed team, do the arithmetic. Make sure you factor in the time and talent you need to support fully the who and where requirements.

{end}

**What ways have you found to work with stakeholders on a distributed team?**

Follow the link on the [StickyMinds.com](http://StickyMinds.com) homepage to join the conversation.

**Get Noticed. Get Certified.**

Jonathan Keleher  
Certified Software Tester

**CERTIFIED  
TESTER**

**Receive  
20% Off!**

Register by Oct. 31  
using promo code:  
BSF20!

## FALL 2009

Indianapolis, IN	Sept. 29–Oct. 1, 2009
Anaheim, CA	Oct. 4–6, 2009
Colorado Springs, CO	Oct. 6–8, 2009
Baltimore, MD	Oct. 6–8, 2009
Toronto, ON	Oct. 6–8, 2009
Rochester, NY	Oct. 13–15, 2009
Raleigh, NC	Oct. 13–15, 2009
Kansas City, MO	Oct. 13–15, 2009
San Francisco, CA	Oct. 19–21, 2009
Austin, TX	Oct. 20–22, 2009
Pittsburgh, PA	Oct. 20–22, 2009
Omaha, NE	Oct. 27–29, 2009
Cincinnati, OH	Oct. 27–29, 2009
Ft. Lauderdale, FL	Nov. 3–5, 2009
Bethesda, MD	Nov. 3–5, 2009
Tampa, FL	Nov. 16–18, 2009
Sunnyvale, CA	Nov. 17–19, 2009
Phoenix, AZ	Dec. 1–3, 2009

[www.sqetraining.com/certification](http://www.sqetraining.com/certification)

**Empower Your Team**

**Software Tester Certification**  
*Certified Tester—Foundation Level Training*



# Food for Thought

by Michael Bolton

One can learn lessons about software testing and general systems thinking in surprising places. As a gift, I recently received the book *The Omnivore's Dilemma* [1] by Michael Pollan. This engaging book explores many dimensions of food and our relationships to it—how we produce it, how we obtain it, and how we consume it.

One section of the book tells a story of agriculture and *reductionism*, an approach to science that reduces complex things to much simpler components and interactions. In the nineteenth century, Baron Justus von Liebig determined that soil fertility and plant growth depended on just three chemicals: nitrogen, phosphorus, and potassium—abbreviated as N-P-K, from their initials on the periodic table. Nitrogen encourages chlorophyll production and growth, phosphorus helps in the growth of roots and blooms, and potassium is vital to the electrolyte balance that keeps the cell machinery working. Von Liebig's research led to the development of artificial fertilizers that made agriculture vastly more productive. With the N-P-K approach, plant nutrition and fertilization became a simple matter: Choose the effect you want, dump onto the soil more of whatever the plant is lacking chemically, and watch the crop yields go up.

In the twentieth century, agronomist Sir Albert Howard spent thirty years working with peasant farmers in India. He observed the interaction between the crops, the livestock that helped till and fertilize the soil, and the people who tended the whole system. Howard recognized agriculture as something far more complex than a chemical recipe, noting that “the health of soil, plant, animal and man is one and indivisible.” In his 1940 book, *An Agricultural Testament*, [2] Howard points out important weaknesses in von Liebig's model. Artificial fertilizers disrupt the balance of natural processes that produce humus, the complex by-product of the breakdown of



ISTOCKPHOTO

organic material. Humus provides minerals for plants and food for microorganisms that are symbiotic to plant growth. Humus also helps the soil retain water, prevent erosion, and keep toxic heavy metals from entering the food chain.

As the statistician George Box said, “All models are wrong; some are useful.” [3] Von Liebig's model of agriculture was very useful in that it led to explosive improvements in crop yields, yet it was also wrong in dismissing the role of humus and reducing biology to chemistry and living systems to mechanisms. Reductionist strategies can be very successful for some time, but eventually the system goes out of balance. Plants shoot up when given big doses of artificial fertilizers, but they also become vulnerable to disease and insects, which farmers attack with chemical pesticides that further disrupt the natural balance. Nitrate-based fertilizers run off into lakes and rivers, promoting the growth of algae, depleting oxygen, and killing fish. Increased productivity leads to lower food prices, so that, ironically, the more productive farmers are, the less money they can get for their crops. It's not at all clear that von Liebig anticipated these

effects. As Pollan says, “Once science has reduced a complex phenomenon to a couple of variables, however important they may be, the natural tendency is to overlook everything else, to assume that what you can measure is all there is, or at least all that really matters.” [4]

This pattern repeats itself in many approaches to testing and test management. No software product stands on its own; every product is a participant in a complex system of interactions between people, computers, and other software. For anything that we test, we face a dilemma. On the one hand, we need to reduce the complexity of the test space so that we can better understand what we're testing. On the other hand, we run the risk of oversimplifying the test space and missing important aspects of the product's relationships with other elements of the system.

Later in the book, Pollan goes on a hunt with a trio of mushroom aficionados. The quarry in this case was the “burn morel,” which flourishes in the first spring after a forest fire. These morels are hard to find; they're small, and they closely resemble the burned sapling stumps and pinecones that cover the



**“As testers, we have models, techniques, strategies, and biases. All of them are sometimes helpful, but each also is capable of misleading us.”**

landscape in which they grow. Pollan describes the patterns and techniques that the experts use for the search. One involved approaching the mushrooms from a low angle so that the heads would be visible; another involved looking near the dogwood plants that thrive in soil of the same temperature as morels; another was staying at the same altitude as previous finds that week. “I could see why you would want theories to organize your hunting,” Pollan says. “The theories told you when to intensify your attention, scrupulously combing the forest floor with your eyes, and when you could safely rest it. For the hunter-gatherer, high-quality attentiveness is a precious but limited resource, and theories, by encapsulating past experience, help you to deploy it most efficiently.”

That reminded me of focusing strategies that we use in testing: trigger error conditions, go backward and forward, force state transitions, try 'em all if you can, consider the opposites, overwhelm the inputs. But then the experts spoke again about an overriding heuristic: When hunting mushrooms (or hunting bugs) you should “be prepared to jettison all previous theories and go with whatever seems to be working in this particular place at this particular time. Mushrooms behave unpredictably, and theories can only go so far in pushing back their mystery.” [5]

Bug hunting, for me, works the same way. As testers, we have models, techniques, strategies, and biases. All of them are sometimes helpful, but each also is capable of misleading us. We never know which one is going to prove most fruitful in a given testing situation, because bugs by their nature are unpredictable and unpredictable.

Still on the mushroom hunt, Pollan describes learning how to spot the morels—a phenomenon known both to mushroom hunters and psychologists as the “pop-out” effect. “...when we fix in our mind some visual quality of the object we’re hoping to spot—whether

it’s color or pattern or shape—it will pop out of the visual field, almost as if on command.” Humans acquired this pattern-recognition capability through evolution; it’s essential to survival to be able to spot food in a complex and chaotic field. Can we take advantage of that evolutionary adaption to find bugs? I think to a great degree we can, and we can strengthen it by practice and priming. Emotional triggers like surprise, confusion, frustration, impatience, or fear provide important clues that point to the existence, meaning, or significance of problems. Reviewing, discussing, and celebrating bugs (as James Whittaker suggests in *How To Break Software* [6]) helps prime us to spot patterns of familiar problems. When we teach Rapid Software Testing, we recommend that people memorize, review, and practice using the Heuristic Test Strategy Model [7] to link test techniques, product elements, and quality criteria. Specific guideword heuristics in the model help us focus on aspects of the product that may contain bugs, while the entire list reminds us to de-focus from time to time. Moreover, we encourage people to develop and master their own heuristics and to share them with the community.

Pollan notes that learning about mushrooms is a life-and-death matter, and that not even the best field guides or photographs can convey the learning and confidence provided by direct experience and close collaboration with mentors. [8] The same can be said for testing, programming, and all sorts of other human endeavors.

For me, the most important testing lesson from *The Omnivore’s Dilemma* is a reminder: We can learn important things about testing from outside the field of testing. Ross Ashby was an influential member of the general systems movement in the early days of cybernetics. He coined the law of requisite variety, [9] which says for any pair of systems in which one is controlling the other, the controlling system must have more states

available to it than the system being controlled. Karl Weick applied this idea to people, saying, “Complicate yourself if you want to understand complicated environments.” [10]

It’s crucial for us as testers to be complicated—to cultivate diverse interests, to learn about the world, and to bring that knowledge back to our products and our processes. Learning is our job. Isn’t that wonderful? **{end}**

#### REFERENCES

- [1] Pollan, Michael. *The Omnivore’s Dilemma: A Natural History of Four Meals*. Penguin Books, 2006.
- [2] Howard, Albert. *An Agricultural Testament*. Rodale Press, 1972.
- [3] Box, George E. P. and Norman R. Draper. *Empirical Model-Building and Response Surfaces*. Wiley, 1987. p. 424.
- [4] Pollan. pp.147-148.
- [5] Pollan. p. 384.
- [6] Whittaker, James. *How to Break Software: A Practical Guide to Testing*. Addison-Wesley, 2002.
- [7] Bach, James and Michael Bolton. *Rapid Software Testing Course Notes*. 2009. [www.satisfice.com/rst.pdf](http://www.satisfice.com/rst.pdf)
- [8] Pollan. p. 372.
- [9] Ashby, W. Ross. *An Introduction to Cybernetics*. Chapman & Hall, 1956. [pespmc1.vub.ac.be/books/IntroCyb.pdf](http://pespmc1.vub.ac.be/books/IntroCyb.pdf)
- [10] Weick, Karl. *Sensemaking in Organizations (Foundations for Organizational Science)*. Sage Publications, 1995. p. 56.

**From what books or disciplines have you taken lessons about testing?**

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

# Rescuing a Captive Project

by Fiona Charles

“You’re kidding!”

It was Emma’s first day on the project and Fabio, the project manager, had just told her, “Management has made it absolutely clear that my top priority is to make Tom happy. We have to walk on eggshells or he’ll quit.”

Emma was impatient. “So let him quit. What’s the big deal?”

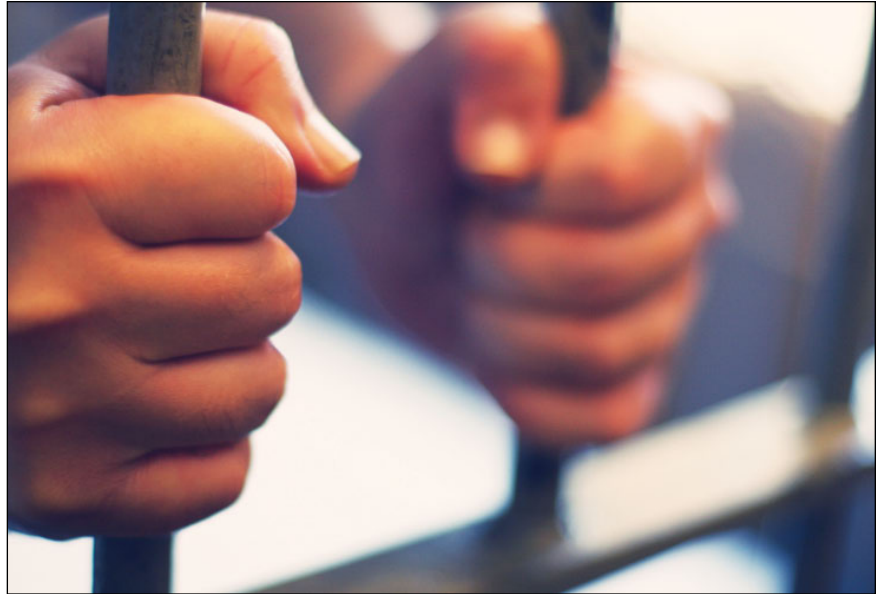
“The big deal is that Tom knows everything about what we’re building, and it’s all in his head. If he walks, we’re in even worse trouble than now!”

Fabio and Emma were the rescue team, parachuted in to replace a more junior project manager and test manager. Bill, the former project manager, had been demoted to technical lead and the test manager had been fired. Tom, the lead architect, was the only survivor from the original project leadership team.

Emma soon realized that Tom was the project’s biggest problem. Although he had a team of two senior people, he kept all critical information about the requirements and design to himself, doling it out to programmers and testers only as individual components were built. His architects, Amanda and Peter, could only run along behind, catching crumbs of information and struggling to contribute.

The application was half built, and the project had already run two months over its projected six. Only the changes in leadership had kept it alive. But, though expected to fix the project, Fabio, Emma, and Bill were handcuffed by their manager’s orders to humor Tom—orders that prevented them from doing everything needed to refocus all the project’s resources for optimal effectiveness.

Worse, as the person upon whom everything depended, Tom was stressed out. He reacted by blaming everyone but himself for the project’s problems. According to Tom, the programmers and testers were incompetent. He shared his contempt with the team constantly,



ISTOCKPHOTO

and most people felt helpless to fight back. He publicly praised his architects and privately abused them. The whole project team knew that resisting Tom brought vindictive retaliation. In this toxic atmosphere, morale was dismal and the application quality was equally low, reflecting the team’s misery and the counter-productive processes.

The project was Tom’s hostage—and management had let it happen.

## How the Project Ended

The project limped agonizingly along and finally delivered a buggy application to the business a year late. Poor quality was the primary cause of the schedule overrun. The programmers in particular had not understood requirements. There were long delays as testers found critical problems and programmers tried to fix them, queuing for Tom’s time as the sole arbiter for requirements and design.

Management decreed overtime to appease the business sponsor. The whole team worked many successive six-day weeks, and the programmers were forced to work even more.

As project manager, then as technical lead, Bill had been beaten down too

often. He did little to shield his programmers or help them get essential information. Programmer morale continued to decline, and team members were frequently ill. Management eventually fired Bill and replaced him with a stronger technical lead—too late for most of the programmers. Three resigned mid-project and the remaining six followed at project end.

Amanda and Peter both quit long before the end. Each had requested transfer to another project. Neither could tolerate management’s lack of support and refusal to move them, and the company lost their valuable skills.

Only the test team survived essentially intact. Coming into an already troubled project, Emma managed to keep a relatively clear perspective. She encouraged her testers to hound Tom for information and defended them from his verbal assaults. Emboldened by support and their own increasing expert status, the testers grew stronger. Eventually, with agreement from Fabio and the new technical lead, Emma paired her testers with programmers to complete the application.

Senior management never addressed the Tom problem, though they directed

## STORY LINES

- **Never allow individuals to hold projects hostage to their expertise or knowledge.**
- **If a hostage situation does occur, remove the problem resource.**
- **Contact the customer immediately after the hostage taker has been removed.**
- **Totally commit skilled people and resources to work through the transition issues.**
- **Recognize that removal of a key resource may slow a project down at first, but overall, a well-managed team will recover and produce a better result.**

him to document the requirements as acceptance criteria for business sponsor agreement. He completed the document too late to help the project, but it was instrumental in securing business acceptance of the application.

The sad conclusion: a bug-ridden application, loss of many valuable company resources, and unhappiness for the remaining people.

If management had supported Fabio and his team leaders in addressing the hostage situation instead of capitulating to it, the conclusion could have been much more positive.

### How It Could Have Ended

Fabio called a team leaders' planning session. To everyone's surprise, Amanda turned up in Tom's place.

Fabio announced, "Tom is on leave for a week. He won't be returning to this project. Amanda has agreed to take on the architecture lead role. She'll need help. Tom's absence is going to leave us with a big knowledge gap, and we have to develop a plan to fill it."

Emma asked, "Will we have access to

Tom when he comes back from leave?"

"We can arrange some limited access, but Tom is moving to a new project. In no time, he'll be too busy for us."

Fabio continued, "Let's start by identifying all the big things that only Tom knows. We probably won't get them all, but we should know the really critical ones."

They brainstormed a list. Most items were requirements-related, with a few associated with the design. Luckily, the fundamental components were already in progress, but some details existed only in Tom's head.

Next, they explored solutions. They agreed it was essential to share requirements knowledge across the project, and that the acceptance criteria management had pressured Tom to document would be a good vehicle. They planned a five-day effort during which Amanda, assisted by Peter, would lead intensive sessions with Tom plus a cross-functional team consisting of the tester and programmer responsible for each functional area.

The remaining big gap was Tom's working relationship with Rebecca, the project's business sponsor. Though Tom had alienated the project team, he had carefully cultivated Rebecca's trust. Fabio had Rebecca's attention for project progress and status, but she routinely took concerns about the application to Tom. Tom's removal could cause the business to lose confidence in the project.


Amanda suggested scheduling time with Rebecca and her new requirements team to review the acceptance criteria, once developed. Along with the obvious benefits of sponsor review, this would help build new relationships and assure Rebecca that the team remained strong.

Fabio and his team began executing their plan. At first, it wasn't easy for Amanda and her cross-functional team to extract knowledge from Tom. But the week's enforced break had changed his attitude. When the initial shock and anger faded, he realized he no longer felt overwhelmed. Relieved, Tom began to feel eager for a new project challenge and strove to cooperate.

Did everything go smoothly from there? Of course not. The project was


still late and over budget. Team members still had to grapple with existing quality problems, and they lost time falling into unexpected holes left by Tom's departure. But the atmosphere was no longer toxic and morale steadily improved. Having shared knowledge across the project, the team members were in a much better position to work efficiently. They learned to trust each other and work together to solve the inevitable problems as they arose.

Best of all, they finished with a solid application that the business was pleased to put into production. {end}



**Have you ever worked on a project held captive by one individual? Could the situation have been improved by removing the person?**

Follow the link on the [StickyMinds.com](http://StickyMinds.com) homepage to join the conversation.



Better Software magazine  
StickyMinds.com

**2009**

Don't miss your chance to contribute to the collective knowledge of industry salaries and employment trends.


Follow the link that best describes your employment level, answer a few questions, and enter to win a \$50 Amazon.com gift certificate!

**STAFF LEVEL:**  
[www.stickyminds.com/2009salarysurveystaff](http://www.stickyminds.com/2009salarysurveystaff)

**MANAGEMENT LEVEL:**  
[www.stickyminds.com/2009salarysurveymanagement](http://www.stickyminds.com/2009salarysurveymanagement)

**DIRECTOR LEVEL:**  
[www.stickyminds.com/2009salarysurveydirector](http://www.stickyminds.com/2009salarysurveydirector)

Survey closes Sept. 25, 2009. Results will be published in the Nov./Dec. 2009 **DIGITAL EDITION.**





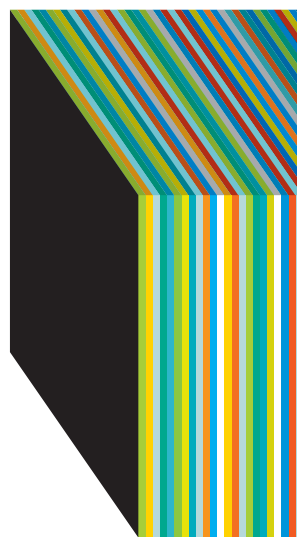
Smarter technology for a Smarter Planet:

## Building the extraordinary into everyday things.

By next year, the average car will require over 100 million lines of software code, and a commercial airplane, over 1 billion. It's approaching the point where a car or a plane isn't simply a car or a plane anymore. What makes them truly unique is the underlying software—the invisible thread—that infuses them with intelligence. It's no wonder then that fully 66% of the products developed in the past year included some kind of embedded software.

Today, every company needs to consider software a core strategic business asset. Unfortunately, 41% of software projects fail to deliver the expected business value and ROI. Only IBM has the experience, the resources and the solutions to build more effective software design and delivery processes for the world's leading businesses. So they can foster collaboration, automate workflows, report progress against goals and improve on results.

A smarter business needs smarter software, systems and services. Let's build a smarter planet. [ibm.com/delivery](http://ibm.com/delivery)







# SOFTWARE LONGEVITY TESTING

PLANNING FOR  
THE LONG HAUL

BY STEVEN WOODY



## How long can your software operate continuously? Is your answer measured in hours, days, months, or years? More to the point, how long do you let the software run during your test cycle?

Longevity testing is an often-overlooked part of software robustness testing for many projects—from networking routers to Web servers and even planetary rovers. Many software applications are intended to run indefinitely, in an always-on operating environment. And yet, few test plans include more than a memory leak test case or let the software run for longer than a few days.

Longevity testing (also known as soak testing or endurance testing) looks for software problems that appear only after an extended operational time. These problems fall into two broad categories: problems due to the passing of time and problems due to cumulative usage.

### The Passing of Time

#### ACCURACY DRIFT

The passing of time can cause software accuracy to drift. A tragic example is the Patriot missile failure in February 1991. The software uptime was stored as an integer and converted to a 24-bit floating-point number, causing the software to have noticeable inaccuracy in the target-velocity tracking after only eight hours of operation and to be inaccurate to the point of failing to track the target after only twenty hours of continuous operation.

The field-mobile Patriot missile launcher was intended to operate for only a few hours at a time. However, nothing prevented the system from being operated for much longer periods. At the time of the failure, the system had been running continuously for one hundred hours. [1]

Don't rely on the software users to reboot their systems periodically. In fact, assume the opposite—that most users will operate the software continuously with never a reboot. This continuous usage goes against the practice in most software test labs, where the software is restarted every few days with a new build if not rebooted hourly between test suites.

#### TIMEOUTS, EXPIRATIONS, AND SCHEDULERS

Most software is populated with timers and schedulers, which start or

stop activities minutes, hours, or days into the future. Sufficient time must be allowed during testing to exercise these timers. Timeouts and expirations to be tested commonly include user session timeouts, firewall session timeouts, ARP tables and MAC address tables aging out, routing tables expiring, SIP registrations renewing, DHCP leases renewing, DNS mappings expiring [2], and age-based passwords expiring. Software scheduler testing should include automatic maintenance activities such as daily virus scans, nightly database resynchronizations, weekly database backups, and monthly software patches or upgrades.

A software license is a type of software scheduler that commonly expires after six or twelve months. Forgotten software licenses tend to expire at the worst possible time. Even worse, a bug in the licensing software can have dramatic consequences, as demonstrated by VMware virtualized servers, which were prevented from starting up after August 12, 2008. [3] Gentle warnings should be given weeks in advance of a software license's expiring. When the system boots up with an expired license, the user should be presented with a clear path to update the license key. Software applications that depend on the software license need to handle the expiration gracefully. Avoid a total crippling of all applications when just one application has a dependency on an expired license.

Some activities are postponed when the software is very busy, and other activities are only activated when the software is sufficiently idle. Longevity testing should include tests of long busy periods (soak testing) as well as long idle periods (quiescent testing).

**Soak Testing:** Testing the system under heavy load for an extended period is known as soak testing or endurance testing. This testing differs from typical stress tests, which apply a burst of heavy load or a volume spike for a matter of minutes. Soak testing verifies that data buffers and message queues operate correctly over periods measured in hours or days.

Place the system under increasing amounts of load (increasing users, trans-

actions, traffic, incoming data, or dataset size). Increase the load to three times, five times, then ten times normal, until any load-regulating, rate-limiting, or discard mechanisms become active. Then, maintain that load and don't let the system come up for air. On embedded systems, incorrect hardware watchdog/heartbeat timeouts can occur when the software is busy for extended periods of time.

Keep the software busy for an extended period so that software postpones or preempts nonessential activities in order to perform time-sensitive activities. Make sure that scheduled maintenance activities—such as database backups, database resynchronizations, virus scans, and in-service software patches or upgrades—do not crash an already busy system if the maintenance window inadvertently overlaps with a period of heavy system use.

**Quiescent testing:** Testing the system with no load is known as *quiescent testing*. Place the system in an idle mode with the minimum amount of activity. Allow sufficient idle time for any timers or leases to expire and any sleep, hibernate, or suspend modes to activate. After a sufficiently long idle period, resume activity on the system, verifying correct operation. This test then can be repeated for longer and longer quiescent periods.

#### CALENDAR DATE AND TIME ROLLOVERS

The passing of time eventually causes clock and calendar rollovers for any software that uses a real-time clock. The Y2K rollover is the most well-known example of this type of bug [4], but there are many standard date and time rollovers that need to be tested [5], depending on the longevity of your software.

All date and time software should be tested for near-term rollover dates and times such as the one-hour adjustment for daylight saving time (in fall and spring) and February 29 (Leap Day, occurring mostly every four years). Systems using UTC or NTP need to contend with an extra leap second being added (23:59:59, 23:59:60, 00:00:00) yearly in June or December, as needed.

Longer-term rollover dates include the GPS system seconds reaching 999,999,999 on September 14, 2011, and the GPS week number rollover on April 7, 2019. Operating systems and compiler-date and time-library routines have rollover dates as well. [5] The most well-known operating system calendar rollover is the Unix 32-bit time overflow that will occur on January 19, 2038. [6]

Some calendar rollover bugs occur on non-standard rollover test dates, as demonstrated by the Tandem CLX fault-tolerant workstations, which stopped working at 3:00 p.m. on November 1, 1992. [1] A bit of research and code inspection will be needed to determine the key rollover dates to be tested for your particular software. Software testers should not hesitate to “look under the hood” at the code, as advocated by Len DiMaggio. [7]

For each rollover date and time test, manually set the date and time of your system and then verify that your software operates correctly during and after each rollover. Verify that billing cycles, reservation dates, alarm clocks, and other date-based calculations operate correctly. Verify that age-based calculations work equally well for someone born today as for someone born 122 years ago. Set the date years into the future to verify that your software does not have an unexpected date or time limitation over its expected product lifespan. Reboot the software after each rollover to verify that the system will boot up, initialize, and then operate correctly.

## SYSTEM UPTIME AND CLOCK TICKS

System-uptime counters and any algorithms that use the system uptime are particularly prone to failure when the counter reaches its maximum value and overflows, restarting the uptime count to zero. These uptime counters are incremented every few milliseconds by a hardware clock, making the rollover occur quite predictably after a specific number of days, depending on the period of the clock tick, as shown in table 1.

Examples of software problems caused in 2008 by system-uptime rollovers included crashes of the Cisco Unified Communications Manager after 248 days [8] and the Cisco MDS 9000 Series

Clock tick period	Uptime counter size	Days until rollover
1 mS	signed 32 bit	24.9
2 mS	signed 32 bit	49.7
2 mS	unsigned 32 bit	99.4
5 mS	signed 32 bit	124.3
10 mS	signed 32 bit	248.6
10 mS	unsigned 32 bit	497.1

Table 1

Multilayer SAN Switches restarting after 497 days. [9]

Even longer uptime problems are possible, of course, as demonstrated in 2006 by the Sun StorEdge RAID Manager resetting after 828 days. [10] For accelerated testing of system-uptime rollovers, it is quite helpful to have a debug command allowing manual setting of the system-uptime counter.

## Cumulative Usage

### RESOURCE EXHAUSTION

The cumulative usage of software tends to create more and more intentionally stored data. If storage resources are not managed carefully, this stored data causes file systems to fill up or free memory to be depleted, a problem known as resource exhaustion.

A dramatic example of resource exhaustion occurred on NASA’s Spirit rover, which stopped communicating with Earth on January 21, 2004, after having landed on Mars just seventeen days earlier. Suspecting a problem with the flash memory, JPL engineers commanded the rover to boot up without reading the flash, and then deleted hundreds of unneeded files on the flash memory, which quickly addressed the problem. [11] The rover has now been running for more than five years, well surpassing its longevity design goal of ninety days of operation.

Resource exhaustion also can occur due to unintentional consumption of resources such as memory—commonly known as memory leaks. A simple bug that neglects to free up a small block of memory after it has finished using it can eventually cause the whole system to crash when no free memory is left. Unix daemon programs must be particularly

well tested for memory leaks, as these programs are intended to run indefinitely. [12]

A recent example of a memory leak problem was discovered in the Cisco MDS 9000 Family SAN-OS Release 3.0 in February 2008, which caused the switches to reload after running out of memory after about 233 days. [13]

When testing longevity, verify that the software gracefully handles commonly occurring resource exhaustion scenarios. According to Jim Gray’s often-cited study of Tandem computer outages, “The most common procedural mistake is letting the system fill up: either letting some file get so big that there is no more disc space for it, or letting the transaction audit trail get so large that no new log records can be written.” [14] At the very least, software should monitor the resources and give clear warnings as critical resources are consumed past 90 percent, 95 percent, and 99 percent levels.

Resource monitoring is a vital tool for quick detection of resource management bugs during testing and operation, but resource exhaustion problems often creep in on resources that are not monitored, including inodes, file handles, sockets, process threads, and data buffers and queues. Longevity testing attempts to find resource exhaustion bugs in as short a time as possible.

### OVERFLOW AND WRAPAROUND

Integer overflows happen as the cumulative use of the system causes integer counters in the software to reach their maximum values.

After reaching the maximum value, the counters then go negative (for signed numbers) or rollover to zero (for unsigned numbers). These unexpected jumps can cause catastrophic problems for the software.

Transaction-based protocols that increment the transaction ID or session ID with each new transaction must handle increasingly larger session identifier numbers as time goes on. Database applications must handle increasingly larger database record ID numbers. These overflow issues can occur more rapidly with larger scale systems, with larger dataset sizes, and with higher transaction rates, all of which can be used to accelerate longevity testing.

## REENTRANT CODE

Some bugs don't show up until the same code has been run more than once. A subroutine that works flawlessly the first time may crash the software when it runs later, due to an improperly re-initialized variable affecting the results the second, third, or nth time around.

Just such a reentrant problem occurred during simulator testing just prior to the NASA shuttle flight on the STS-2 mission in 1981, causing all four of the flight computers to lock up. The subsequent investigation revealed that a reentrant subroutine for fuel control was not properly initializing a variable during subsequent passes. [15]

Similarly, a reentrant bug in Microsoft's Internet Explorer was found in December 2008, requiring a security patch to address it. [16] In order to catch reentrant code bugs, software features must be tested repetitively during longevity testing—without any reboots or restarts of the software.

## Accelerated Testing Techniques

Scaling up the system to the maximum that resources allow will help accelerate the occurrence of many longevity bugs. Increase the number of users, the number of transactions per second, the interface traffic rate, the incoming data rate, or the dataset size as your system and testing budget allow.

Another technique to accelerate longevity testing is to preset any counters, session IDs, or record numbers to within a few counts prior to the rollover point. Then, start normal operation and observe that the integer rollovers are handled correctly in the software.

A third acceleration technique is to

pre-consume the software resources. Nearly fill each storage type, including flash memory, RAM, and hard disks. When creating test files to fill storage space, perform two types of accelerated tests: the first using one or two extremely large files, and the second creating thousands of minimum size files in multiple subdirectories. For both tests, repeatedly create and delete files and directories to exercise the file system data structure.

## Which Longevity Test to Run?

When running a longevity test, should the software be idle, as busy as possible, or somewhere in between? Each type of environment has its advantages and disadvantages. An extended period of heavy usage followed by an extended period of normal usage will find many longevity problems. Try to compress at least one year of expected activities (logins, refreshes, transactions, calls, packets) into the first seventy-two hours of testing.

During any longevity testing, increase the sensitivity on system alarms and logs, and then periodically check for any unexpected error messages. If supported, start a run-forever command such as a continuous ping. Monitor critical software resources for leaks. Continuously verify the software operation for correctness, accuracy, and performance.

## How Long to Run Longevity Tests?

Testing for the full operational life of the system is only practical on software with the maximum run time measured in just hours or days. Most software will be in operation for months or years or will be expected to run indefinitely. Even with automated testing tools and accelerated testing techniques, how long should you test software longevity?

If the test overhead or equipment costs are prohibitively high for a product, a full suite of longevity tests might only be done once. Communications satellites and space exploration probes are operated continuously for weeks at a time during system testing prior to launch. [17]

Other products may benefit from a regular verification of longevity test cases, if only for major releases. High-

availability, continuous-uptime, and nonstop software should be endurance tested with a continuous, heavy load for a full eight days, as some problems will not appear until after a full seven days of continuous stress testing.

Keep track of how long the software runs continuously during testing. Document the maximum uptimes tested for idle, normal, and stress operation for each version of software. A software feature useful for longevity testing is a log message giving the system uptime at the point when any reboot, restart, or reload command is issued.

To determine how long to test longevity, perform a code review and make a list of the integer counters. Calculate how long it will take for each counter to overflow. Your longevity testing should run at least long enough to verify that counter rollovers are handled correctly.

Start longevity testing early with the available software version under development, well before the planned release date. One approach is to set up six longevity test systems (powered via an uninterruptible power supply). Once a month, verify correct operation on all six systems. Once a quarter, update the shortest-running system to the latest software version under test. Do this every three months until you have six systems, one each running 90, 180, 270, 360, 450, and 540 days. When the oldest system reaches 540 days of uptime, update the software on that system to the current version.

To summarize the steps for longevity testing, start early, put the system under heavy load, continuously monitor the performance, and let it run for days, months, or years. Remember that a few simple longevity tests can prevent major problems from eventually occurring—it's just a matter of time. **{end}**

## REFERENCES

- [1] Neumann, Peter G. *Computer Related Risks*. Addison-Wesley Professional, 1994.
- [2] Brewer, Eric A. "Lessons from Giant-Scale Services." *IEEE Internet Computing*. July/August 2001. [www.cs.berkeley.edu/~brewer/Giant.pdf](http://www.cs.berkeley.edu/~brewer/Giant.pdf)
- [3] Keizer, Gregg. "VMware licensing bug blacks out virtual servers." *ComputerWorld*. August, 2008. [www.infoworld.com/article/08/08/12/VMware\\_licensing\\_bug\\_blacks\\_out\\_virtual\\_servers\\_1.html](http://www.infoworld.com/article/08/08/12/VMware_licensing_bug_blacks_out_virtual_servers_1.html)



# The **Seven** Habits of Highly Effective Testing Organizations

A New White Paper by

**Testing Expert Lee Copeland**

How are you tackling quality issues?

Is your testing organization regarded as trusted advisor – or a perceived bottleneck to getting releases out the door?

**From this white paper you will learn:**

- . How Agile is changing the way we test
- . Why the testing organization no longer needs to be a distinct group
- . The importance of risk-based testing and its main elements
- . Change management's importance for producing quality software

Lee Copeland brings us a look at the habits that we should adopt to take testing to a new level and reap greater value from QA efforts.

Get the free white paper, compliments of MKS:

[www.mks.com/seven\\_habits](http://www.mks.com/seven_habits)

Contact us: 1 800 613 7535 | [info@mks.com](mailto:info@mks.com)

**MKS**

- [4] Collard, Ross. "The Y2K Bust." StickyMinds.com [www.stickyminds.com/s.asp?F=S3241\\_ART\\_2](http://www.stickyminds.com/s.asp?F=S3241_ART_2)
- [5] "Potential problem dates for computers." The Institution of Engineering and Technology, 2001. [www.theiet.org/factfiles/it/index.cfm](http://www.theiet.org/factfiles/it/index.cfm)
- [6] "RFC2550 – Y10K and Beyond." The Internet Society, 1999. [www.faqs.org/rfcs/rfc2550.html](http://www.faqs.org/rfcs/rfc2550.html)
- [7] DiMaggio, Len. "Looking Under the Hood—An Investigative Approach to Software Testing." *Software Testing and Quality Engineering*, January 2000. [www.stickyminds.com/getfile.asp?ot=XML&id=5023&fn=XDD5023filelistfilename1%2Epdf](http://www.stickyminds.com/getfile.asp?ot=XML&id=5023&fn=XDD5023filelistfilename1%2Epdf)
- [8] Field Notice: FN – 63174- Cisco Unified Communications Manager. Cisco, 2008. [www.cisco.com/en/US/ts/fn/631/fn63174.html](http://www.cisco.com/en/US/ts/fn/631/fn63174.html)
- [9] Field Notice: FN - 63178 - MDS Fabric and Blade Switches May Reload After 497 Days of Uptime. Cisco, 2008. [www.cisco.com/en/US/ts/fn/631/fn63178.html](http://www.cisco.com/en/US/ts/fn/631/fn63178.html)
- [10] Sun StorEdge RAID Manager. Sun Microsystems, 2006. [sunsolve.sun.com/search/document.do?assetkey=1-66-201560-1](http://sunsolve.sun.com/search/document.do?assetkey=1-66-201560-1)
- [11] Reeves, Glenn and Tracy Neilson. "The Mars Rover Spirit FLASH Anomaly." Jet Propulsion Laboratory, 2005. [trs-new.jpl.nasa.gov/dspace/bitstream/2014/39361/1/04-3354.pdf](http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/39361/1/04-3354.pdf)
- [12] DiMaggio, Len. "Testing Unix Daemons." Dr. Dobb's Journal, March 2000. [www.ddj.com/184404033](http://www.ddj.com/184404033)
- [13] Cisco MDS 9000 Family SAN-OS Release 3.0. Cisco, 2008. [www.cisco.com/en/US/ts/fn/620/fn62818.html](http://www.cisco.com/en/US/ts/fn/620/fn62818.html)
- [14] Gray, Jim. "A Census of Tandem System Availability between 1985 and 1990." HP Technical Reports, 1990. [www.hpl.hp.com/techreports/tandem/TR-90.1.html](http://www.hpl.hp.com/techreports/tandem/TR-90.1.html)
- [15] "Excerpt from the Case Study of the Space Shuttle Primary Control System." Communications of the ACM, 1984. [www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Ariane/shuttle.html](http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Ariane/shuttle.html)
- [16] Howard, Michael. "MS08-078 and the SDL." [blogs.msdn.com/sdl/archive/2008/12/18/ms08-078-and-the-sdl.aspx](http://blogs.msdn.com/sdl/archive/2008/12/18/ms08-078-and-the-sdl.aspx)
- [17] Academy of Program / Project & Engineering Leadership. "Redesigning the Cosmic Background Explorer (COBE)." National Aeronautics and Space Administration. [www.nasa.gov/pdf/293139main\\_COBE\\_case\\_study.pdf](http://www.nasa.gov/pdf/293139main_COBE_case_study.pdf)

## Sticky Notes

For more on the following topics go to [www.StickyMinds.com/bettersoftware](http://www.StickyMinds.com/bettersoftware).

- Longevity bug examples
- More reading on longevity testing

# AGILE SOFTWARE DEVELOPMENT TRAINING

*Accelerate Your Career & Empower Your Team*



## 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 \$500. For a complete list of courses available, visit [www.sqetraining.com](http://www.sqetraining.com) or call 888.268.8770 or 904.278.0524 for pairing discount options.



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.

## AGILE SOFTWARE DEVELOPMENT TRAINING WEEK LOCATIONS

September 24-25, 2009	<b>Boston, MA*</b>
October 12-16, 2009	<b>San Diego, CA</b>
November 8-10, 2009	<b>Orlando, FL**</b>

\*Only User Stories and Estimation in Agile Development is available in Boston, MA.

\*\*Only Scrum Master Certification is available in Orlando, FL.

## Choose from 4 specialized training courses:

### THREE-DAY COURSES (Monday - Wednesday)

- Scrum Master Certification
- Design Patterns Explained

### TWO-DAY COURSES (Thursday - Friday)

- Practical Test-Driven Development
- User Stories and Estimation in Agile Development



Let SQE Training come to you. For more information about on-site training courses, contact SQE Training at 904.278.0524 x212 or 233 or 888.268.8770 or email [onsitetraining@sqe.com](mailto:onsitetraining@sqe.com).

# IT TAKES A VILLAGE

## 6 Ways to Fill the Product Owner Role

by Ronica Roth



“It takes a village to raise a child,” claims the old Nigerian proverb. That’s a far cry from “A mother is the single, wringable neck responsible for turning out a good kid.”

The folks at Yahoo! coined the phrase “single, wringable neck” to emphasize that every software delivery team needs to be guided by decisiveness, responsibility, and accountability.

Scrum emphasizes the importance of the product owner role, because the business—rather than engineering—should define how the business will derive value from software.

## Scrum 101

In the original Scrum book [1], Ken Schwaber and Mike Beedle describe a Scrum team as including a product owner—a single person who owns the vision and backlog, accepts completed work from the team, and calls for releases.

Yet, in dozens of companies I’ve visited—those that are agile and those that are striving toward agility—the single, accountable product owner is not enough to define value for their software projects and products. The one-product-owner/one-team model just doesn’t cut it for larger organizations.

Different companies have different needs in terms of how to define value for their software, who does that work, and by what processes. For example, some companies struggle with scale—how can I staff every Scrum team and also watch the big picture? Some companies seek a solution that addresses complexity, such as a product of tightly integrated components or one that requires the input of many specialties. Other companies adopt a Scrum framework only to have it make visible an under-investment in product management; they need to find people with the right skills to support delivery teams.

My challenge has been to help these teams find a structure that meets their needs while also following the principles of agile software development—that is, a structure that emphasizes collaboration and that focuses development squarely on providing value with quality.

In all the permutations I’ve encountered, I’ve discovered patterns used by companies to fill the product owner role

successfully. Each pattern addresses different challenges regarding scale, complexity, or skill sets.

A common situation is the case of one product owner who owns a single backlog for a single product (or project) that is being worked by two teams. It’s usually a simple case of the work’s needing two teams to meet delivery goals.

Although this scenario breaks with the idea of one owner per team, it does have the advantage of providing the product with a single owner. Thus, that owner has the vision and owns the entire backlog. Depending on the skills of the teams, she may have maximum flexibility to move work between teams to deliver value.

If the owner is new to agile software development, I recommend you ease her into this arrangement. Let a newly minted product owner learn the rhythms and cadences of iterative development in a Scrum 101 arrangement before you ask her to support two teams at once.

That said, this scenario can be successful as long as that one product owner is truly dedicated. There’s no time here for her to be assigned to any other work. Her time will be filled with grooming enough backlog for two teams’ worth of work each iteration (see sidebar: Grooming the Backlog) and with accepting done work.

And, therein lies the challenge. That’s a lot of work, especially if the product owner has to spend much time with stakeholders to groom the backlog effectively. The Scrum 101x2 product owner may fall behind—and burnout—quickly.

Instead, try ...

## Scrum 101<sup>2</sup>

Or, you could call it “Scaling Scrum 101.”

In this pattern, we assign a product owner to each team, just as the authors of Scrum originally recommended. This means each team gets maximum access for collaborative planning, prototype reviews, questions during the iteration, and story acceptance. This highly collaborative work and intense involvement of the product owner ensures common vision and completed stories that more often deliver value.

If we’re describing two teams working

## GROOMING THE BACKLOG

A core piece of the product owner’s job is to groom the backlog. This work is a significant shift from the product manager’s or business analyst’s previous job of developing a requirements document up front and handing it off. Instead, an agile product backlog needs constant attention:

- The backlog tends to have small, clear items near the top and larger, fuzzier items near the bottom. The product owner must constantly look ahead to what’s coming and gather more information, breaking larger items into smaller ones that fit into an iteration.
- The product owner adds acceptance criteria and other detail just in time for planning.
- When iteration demos and frequent releases result in feedback, the product owner turns that feedback into new items for the backlog.
- The backlog should be stack-ranked. As new items are added to the backlog—via bugs or feedback or suggestions from the team—the product owner must decide where those items fit in the rankings.

on a single product or on closely related work, the challenge is for the two teams and two product owners to coordinate and collaborate effectively. Scrum doesn't specifically prescribe ways for these product owners to collaborate the way it helps a delivery team know how to collaborate via daily stand-ups and planning sessions.

In my experience, product owners need formal, cyclical opportunities to collaborate on backlog grooming, planning, and feedback.

For the teams, the more closely connected the work, the more important it is for them to conduct some kind of release planning (plan the work that will go into the next release) or midrange planning (plan the next three to six iterations, regardless of whether they result in one release or many). That planning session is the opportunity to coordinate feature delivery and identify and mitigate dependencies. At this level of planning, we are talking about backlog items across iterations, not detailed tasks within an iteration. The idea is to determine at a midlevel which work will be done when.

In addition, the product owners must meet periodically to ensure their backlogs remain in sync. If the work is more separate, they can meet relatively infrequently, perhaps once an iteration. If the work requires more coordination, then they need more frequent meetings. I've known at least one group of product owners who met daily.

The product owners and their teams should also attend each other's demos to get a good sense of each team's progress. Finally, I recommend a specific release or midrange retrospective that is focused not on individual teams but on how the multiple teams and product owners are coordinating.

In this scenario of multiple related teams, each with a product owner, the biggest pitfall I've seen is a failure to build consensus on a single direction for the software. In that case, the group might benefit from an über product owner.

## The Product Manager, aka Über Product Owner

The problem of clear, big-picture vision can be solved by the über product

owner pattern. In this scenario, a so-called über product owner (often someone with the title product manager) "owns" the overall value of the product or project and is assisted in execution by team-level product owners (often people with business analyst titles).

The über product owner owns the big picture and the release and ensures related teams are coordinating to a shared vision and definition of value. The team-level product owners then own the backlogs and must help detail user stories with acceptance criteria that will help build toward the release goals.

The key to success in this model is that the über product owner and product owners work very closely to ensure they are working toward the same goals. For example, I worked with at least one company in which the über product owner had weekly meetings with his team-level product owners. This ensured that the product owners had an opportunity to voice any concerns or discuss any obstacles. Another purpose of the meetings was to synchronize the future backlog, to ensure all backlog items were prioritized to meet release goals.

I worked with another group of product owners who met with their über product owner in a daily stand-up. The idea was to report on progress in backlog grooming and make visible any impediments to getting clarity on upcoming backlog items.

I have seen the über product owner approach fail when the person in the role of über product owner did not really understand the evolutionary, iterative, and value-oriented nature of agile software development. For example, in one company, the über product owner expected to be involved up front and again just before release but did not expect or want to weigh in on feedback and direction throughout the development cycle.

In another example, the product owners were trained in agile but were led by an über product owner who was not trained. That über product owner measured progress according to his original sense of the complete scope of the project. He did not understand that, instead, he and his product owners should constantly evaluate the delivered features to determine what the minimal

marketable feature set might be. As a result, his product owners felt pressured to deliver everything by the original date, regardless of new learnings. In that disconnect, the über product owner offered more pressure than direction. Thus, it is critical that the über product owner be as versed in agile software principles as anyone.

In this work of tradeoffs, the agile product owner—über or otherwise—can be helped by an architect partner.

## Dynamic Duo

A colleague of mine worked at a consulting company that was building Web sites for clients (and also building common code). The product owner role was filled by a business analyst/architect tandem from the client. This tandem served a user community and also a technical steering committee. Together, they were able to ensure that the work of the software development teams served both user goals and technical goals.

For companies that use an architect role (see sidebar: The Role of the Architect), it's critical that product owners have a close working relationship with architects. Luke Hohmann of Enthiosys describes agile product roadmaps as containing an architectural element. Just as we need to plan a product direction strategically, we want to think strategically about evolving the system to support that plan. Dean Leffingwell talks about laying architectural runway that serves the product direction. [2] These efforts will be most successful when product owner and architect are collaborating, rather than competing.

Whether a true tandem or simply a productive partnership, the dynamic duo of product owner and architect is ultimately working to serve its core constituencies: users and customers. Yet how does a product owner—or über product owner—make sure she really is representing all those constituents? Enter the product council.

## Product Council

Another village that might support a product owner is the product council that helps define priorities at the project, or epic, level.

I was product owner for an internal

enterprise application that served nearly every company department. Each department had its own wish list of enhancements and defect fixes for that application. Some projects crossed departments. My product council (actually called an IT steering committee) met once an iteration. At those meetings, department heads lobbied for their projects and the chief operating officer provided information about company-level strategies. I guided the group to reach consensus about project priorities and the group provided a ranked list of projects. Note that projects in this case were anywhere from three iterations to less than one iteration.

The product council did not determine which stories took priority within a project. I worked with the department heads and stakeholders to determine which stories of a given project would make the cut for a given iteration.

The challenge in my case was gaining consensus among people who were competing for limited resources. The key was strong facilitation and the payoff was that decisions were stickier—department heads stopped trying to go around the system to get their work prioritized in the queue.

The added bonus was that the department heads stopped asking IT to make trade-offs between projects. Instead, they did their own negotiation to de-scope projects and intertwine priorities. I started hearing sentences like “What if we do the highest-value part of your project in two iterations, then deliver the highest-value part of mine?”

Years ago at Rally when we started with Scrum, we used to do a design meeting every few weeks with a couple of key stakeholders to talk about what was coming up and prepare the backlog. This worked relatively well, but as we added more discipline to our release cycle, the ad hoc backlog planning to which our product owners were accustomed started to break.

We found that if you want your team to be able to make a commitment around eight weeks of backlog, you need to do somewhat more preparation. And, if you want your team to meet that commitment, you need a mechanism to manage your stakeholders to minimize backlog churn.

About a year ago, Rally chartered a product council to solve this problem. The product council is led by the product line manager (über product owner) for each product and consists of stakeholder representatives from all interested departments. This group operates as a lightweight Scrum team that grooms the backlog for the next release, meeting every two weeks and working through a sort of kanban of features, as shown in figure 1.

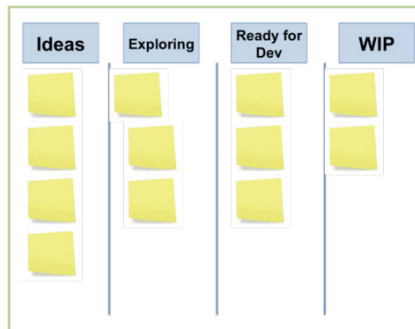


Figure 1: Product council kanban

## PRODUCT COUNCIL KANBAN

- “Ideas” are things we’ve talked about but that have never been evaluated in depth for size or feasibility.
- “Exploring” items are currently being researched by product owners and others but are not quite ready to be developed.
- “Ready For Development” includes items that the product owners feel are understood well enough to be pulled into release planning.
- “Work In Progress” (WIP) items are pieces that we’ve started but not yet finished.

The advantage to our product owners is clear guidance for the product, which includes many viewpoints. The council helps right-size our investment on research, leading, we hope, to the biggest bang for our buck as product owners look ahead.

But what if there are many constituents who need to be involved in more than an advisory role? What if you really need the input of many stakeholders at a detailed level throughout every iteration? Enter the product owner team.

## THE ROLE OF THE ARCHITECT

“Architect” is an overloaded word in the software world. In some organizations, the title simply means highly skilled developer. In other organizations, the architects are the gatekeepers, preventing delivery teams from “bad design.” Some architects do all the designing and none of the coding—a decidedly un-agile approach.

An approach I’ve seen that seems more agile is for the architect to function like one half of a programming pair, looking at the big picture while the teams focus on the detail. The best architects work closely with the delivery teams, writing code, mentoring developers, and using their skills and experience to look for ways to improve quality across all teams.

Wherever architects are responsible for big-picture design, they need to be working closely with product owners.



## The Product Owner Team

I introduce this last pattern with some trepidation. I think the product owner team pattern is fraught with challenges around coordination and collaboration. That said, I've seen it often enough—and seen it be successful enough—that I believe it's a valid pattern with benefits for certain organizations.

The basic pattern is that of having a group of people speak with the single voice of a product owner.

I worked with one company that developed medical practice software. When the company first started with agile, it had seventeen different job titles on the project, which felt, at least to some developers, like an anti-agile level of complexity and specialization.

Some of this complexity was truly unavoidable. Medicine is already a complex, specialized area, and the company had clinical sub-specialties to serve. It further had the complicated relationships of the very hierarchical world of medicine, where physicians of different specialties, nurses, and practice staff all have clear places and real and different passions. On top of all that, the project included user experience designers who worked on a strategic level, business systems analysts who had expertise around the business, a product management group, and business analysts who were requirements analysts.

To handle all that specialization while managing a single product backlog, the company formed a product owner team. Here's a true village raising the child—"software that delivers value." The product owner at the head of it all was a practicing physician who drove feature prioritization. Business analysts and clinical analysts would then "plead their case" to the product owner to get detailed stories prioritized. Challenges around this effort included a lack of knowledge on how to "sell" ideas, although that got fixed pretty quickly once stories got deferred. It worked well in terms of communicating value and context to delivery teams, since stories had to be well thought out in order to rise to the top. Technically, the product owner also had to accept stories, although he usually deferred to the analysts.

This team of owners worked like a

Scrum team to evolve a ranked backlog. The owners committed to iterations for which the deliverable was a story ready for the delivery team to estimate and plan.

To provide some consistency, each of the business analysts on the product team was also assigned to a delivery team. The clinical analysts and business systems analysts would "follow" their stories to the delivery team that was going to implement them, once that was determined in sprint planning.

Thus, this group managed specialization while also providing the consistency that the delivery team needed.

A variation on the product owner team solves the problem not of specialization but of silos. At the Agile 2008 conference, British Telecom (BT) presented its solution to the product owner puzzle. A value stream mapping by Mary and Tom Poppendieck had identified incredible waste in BT's highly siloed organization.

In the wake of that assessment, BT sought to coordinate delivery among the silos and create end-to-end ownership by creating a product owner team of senior managers. The product owner team members had to be dedicated to the project, and they measured success or failure only as a team—quite a departure from their segmented history.

All decisions regarding backlog, prioritization, and acceptance were made collaboratively and by consensus. A key to success, BT reports, was a strong facilitator who could guide these senior managers to decisions.

## What Village Can Guide Your Software Development?

As you try to determine how to fill the product owner role for your organization, worry less about the form or about what the literature says, and focus instead on how to fulfill the principles underlying the role:

- Business and development should work together daily (from the Agile Manifesto).
- The delivery team needs to understand the business and user goals that underlie requirements.
- Agile software development is

about focusing on delivering value—and on avoiding building that which does not provide value. Developers should not be the ones defining value, but rather the business should define how software will create value.

- Product owners need technical input—including from the team—to rank a backlog properly to deliver value, reduce risk, and generate feedback.
- Agile software development is about collaboration. Ensure you are collaborating with stakeholders to understand their needs and concerns.
- Consensus takes more time and effort but leads to stickier decisions.

Adopting agile practices may look like a software development decision, but in the end, the biggest change is often for the product management or business analyst community. We need to set up this group for success with the same level of support and guidance as we do delivery teams. Too many organizations see one owner per team and balk at the impossibility of it. Instead, consider which one of these patterns reflects your organization and can provide the business direction that ensures your development organization delivers value. **{end}**

### REFERENCES

- [1] Schwaber, Ken and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.
- [2] Leffingwell, Dean. *Scaling Agility: Best Practices for Large Enterprises*. Addison-Wesley Professional, 2007.



# Agile Development



## Agile. We are! Are you?

Agile is here to stay. But how do you make the transition, and adopt it alongside your existing methodologies? Look no further. Our tools make adopting Agile practices easier, faster, and lower cost regardless of size and complexity of your projects. How can we be so sure? We understand, better than anyone else, how to build application development tools. We've been doing it for over 28 years. What's more, our developers co-invented Scrum. Who better to work with in your move to Agile?

[www.serena.com/agile](http://www.serena.com/agile)



# IDEs & Build Scripts

Working together  
in harmony

by Steve Berczuk

*Interactive development environments* (IDEs) are powerful productivity tools for software developers. While Java developers often work in Eclipse or IntelliJ IDEA (two popular IDEs), many projects also use automated integration build tools, such as Maven. The use of two different tools can cause inconsistencies if not well managed. This article discusses the risks that can occur when IDEs and build scripts diverge and provides guidelines for keeping the two views of the development project consistent while maintaining the relative benefits of each.

Let's begin with an example scenario:

Jim is very productive with Eclipse, his favorite IDE. To implement a new feature, he makes a change to a Java source file, runs the unit tests through the IDE, and sees a green bar indicating that all the tests passed. Since everything seems to have compiled and the tests ran, Jim uses the IDE to commit the changes to the source code repository and then takes a break for lunch.

When he returns from lunch, Jim notices an email from the integration build system saying that there was a compile error. Looking at the build log on the integration machine, Jim realizes that he forgot to update his IDE configuration to reflect the team's decision to update a third-party library. The method calls in the code that he committed to the repository were deprecated (declared obsolete) in the prior version and are no longer valid in the current version. Simple enough to fix. Too bad that anyone who updated his workspace while Jim was at lunch had to debug a broken build.

Across the hall, Mary is engaged in a heated discussion with Phil, the release engineer. Phil wants to use Maven's resource filtering to configure a properties file at build time. Mary is arguing that this is a bad idea, as the resource filtering puts a meaningless token in the source configuration file, which makes her IDE-based debugging settings not work.

In this scenario, there is a mismatch between the configuration of the developer's IDE and the configuration of the integration build. In Jim's case, the IDE settings were not updated to show the current state of dependencies as reflected in the Maven build scripts. In Mary and



Phil's case, an idiom (resource filtering) that works quite effectively in the integration build environment doesn't match the way that a developer works within an IDE. These are two examples of how the differing ways that developers configure IDEs and configure build scripts can cause disfunction and conflict in the team. Fortunately, there are approaches to reconcile these differences.

## The IDE

IDEs can increase productivity for individual developers by enabling them to maintain *flow*, a hyper-productive mental state in which a person is fully immersed in what he is doing. Development practices support flow by working to focus the team on tasks that add value and to eliminate distractions, such as non-productive work.

IDEs help maintain flow by allowing developers to focus on the intent of their development tasks rather than the mechanics. This gives the developer more rapid feedback on the results of tasks such as refactoring, which enables him to perform refactoring tasks such as "move method" by indicating the code to be moved rather than having to perform the mechanical copy, paste, and search for compile errors elsewhere in a project. IDEs allow you to code without thinking too much about coding style conventions. A properly configured IDE can either format the code as you type or let you format it afterward. You need not worry about tabs, spaces, alignment, or other peripheral tasks, allowing you to maintain focus and avoid changing contexts to perform your primary tasks. IDEs also interface with other supporting tools, such as software configuration management and issue-tracking systems, via plug-in mechanisms to simplify their use.

IDEs improve feedback, identifying errors in the code as you type and avoiding the interruption in flow that a full compile-build-test cycle can cause. While a full compile-build-test cycle will give you an accurate assessment of the state of your application, it will interrupt

your thinking and disrupt flow if it takes longer than a few seconds. To allow for a more rapid incremental build, compile, and run scenario, the IDE environment is not always exactly the same as the full build process that runs in your integration environment, with an IDE using an incremental compiler, or different paths to resources, for example. There are risks lurking in these differences. If your IDE uses a build configuration that is different from the one used in the integration build, you need to keep them in sync to avoid the "works in my IDE but not in the integration build" scenario.

## The Integration Build

An automated integration build verifies that your code is working after every set of changes to the source code tree, allowing you to identify issues quickly so that they can be fixed closer to the introduction of the problem.

You see the real benefits of an integration build by using continuous integration, which enables you to increase the rate of feedback so that you can detect problems quickly after a change is made. A continuous integration environment polls the version repository every few minutes—waiting for a short period of time after the changes to let the system settle—and performs a build if files have been updated. This avoids problems where, for example, someone forgets a file during a commit, remembers it afterward, and then adds that file. This short wait reduces the chances for false-negative build failures.

A build on an integration server has several advantages over a build on a developer machine:

- The build takes place in a controlled environment, so you know that the build works in the "defined configuration." Developer

machine configurations have the potential to vary minute by minute.

- The build integrates changes directly from the source code repository, validating that the build reflects all changes up to the time of the build. For example, if a developer forgets to commit a change to a file, the code in his workspace will build, but the integration build will fail. Or, if a developer makes a change that conflicts with another developer, the integration build will quickly detect the problem.
- The integration build can be a source for identification (build numbers, build labels) that can be applied and referred to consistently.

The integration build provides an additional protection against developer mistakes and misunderstandings. When a team practices continuous integration, "working code" means "works on the integration build" in that the code compiles in the integration environment and passes all unit and integration tests specified to run as part of the build.

Even with this definition of "done," compiling and testing in an IDE isn't enough to guarantee that the build works; the IDE configuration must match the build configuration.

## Overlapping Functionality

Because of differing requirements, IDEs, such as Eclipse, and build systems, such as Maven, have different configuration mechanisms. An IDE typically maintains project files in its internal format. Maven defines projects using an XML-based project-object-model (POM) file. The information is similar but stored

in different places. While some IDEs can derive their configuration from the Maven project files, inconsistencies can still remain. To ensure that the code is consistent with the build tool, always use the command line to build. However, it isn't practical to use the command line to build the entire project after an edit.

Because development teams will want to work inside an IDE, it is essential for project-related configurations to stay in sync between IDEs and build scripts. The configurations that need to be synchronized between IDEs and build tools include:

- Inter-component dependency information—for example, that the “UI” project depends on the “model” and both depend on the “core”
- External dependencies (name, version, etc.)—for example, knowing that the core project depends on “Apache Commons” version 2.4
- Build-time prerequisites—for example, if the “UI” project depends on a “persistent-model” that is based on code generated from meta-data

Keeping these dependencies synchronized not only helps avoid checking in broken code but it also allows the IDE to provide accurate feedback when checking syntax and executing tests.

## Synchronizing

Modern IDEs, such as Eclipse and IDEA, can synchronize with Maven project files to ensure that dependency information is correct. Unfortunately, even after synchronizing with a project file, there are other settings, such as coding style preferences and debug and run configurations, that must be maintained manually because they may be customized to the user's environment. A challenge for teams is to decide whether to maintain IDE configurations in the same way that build project settings are maintained—under version control or not.

If your team uses a single IDE, maintaining project files under version control sounds appealing. It's easy to get a new developer workspace up and running by simply checking the IDE configuration out of your source code management system along with the source code. If

you version your IDE configuration files, you need to decide how to maintain IDE configuration. Two choices are:

- Ad hoc—The developer changing a build script also changes the IDE. This approach helps ensure that there is responsibility for updating the IDE configurations. However, since there is no automated way to verify the IDE configuration's correctness, a forgetful person has no backup.
- Assigned responsibility—Have a toolsmith be responsible for updating the IDE configurations. While this approach helps to ensure that the configurations are kept up to date, the toolsmith is essentially manually translating the information into build script in the IDE configuration. This involves duplicate work that can be of low value.

In many cases, adding a requirement that one IDE configuration work for everyone means that there are restrictions on how developers can customize their IDEs. While it's useful and appropriate for teams to develop standards for style and how code should function, developers can be most effective when they are allowed to decide how to work. Since IDEs are individual productivity tools, you might find yourself reducing productivity in an attempt to increase it by maintaining a common configuration.

On some teams, IDE configurations are checked in for convenience and developers check them out, customize them, and then make an effort never to check them in again. The IDE configurations are only in the source code management system for reference. While this seems to address the question of how to get a developer up and running quickly, asking team members to use as a starting point a resource that is not maintained can cause more problems than it solves. Any file in your source code management system that isn't used by everyone as the definitive source for information should be evaluated for marginal value.

Since an IDE is a key part of a developer's toolset, it is not essential that IDE configurations be turnkey. As long as the IDE can import project dependency in-

formation from a build configuration, a developer will quickly learn to configure any other settings he needs to be productive.

## Balancing Tools

Another common issue in teams that heavily use IDEs is that techniques that work well in a build environment may not work transparently in an IDE. For example, if your project uses code generation, you need to generate the code (using the build tool); otherwise, your IDE will give errors when you reference the generated code. While the default build mechanisms for IDEs work well, IDEs also allow you to customize how projects are compiled. You can add a step to the build to run a Maven phase before building your code. For example, Eclipse lets you define project builders and IDEA lets you define Maven goals to execute before running the debugger.

Remember that an IDE is a tool you use to create an application, but the final application will be constructed using a build tool. Don't sacrifice functionality and value to the automated build because it doesn't work immediately in your IDE.

## What To Do

Since the definition of “done” is “works in the integration build,” use the build configuration as your primary source for project definitions and don't maintain derived sources in version control.

Use an IDE that allows you to define a project in terms of the build script mechanism. IDEA and Eclipse allow you to import a Maven project file. Alternatively, there are tools that create IDE configurations from build configurations. The Maven-Eclipse plugin and Maven-IDEA plugin allow you to generate Eclipse and IDEA project files from your Maven POM files.

While some configurations, such as those related to debugging, may not be derivable from the project definition used in the build, it still may be easier to provide guidelines for developers to customize their environments once they have imported the build configuration. Even developing internal tools to set up an IDE configuration from a project definition is more cost effective than manu-

ally maintaining two definitions. Since IDEs are extremely customizable, make the requirements of the automated build environment primary.

As a final step, establish a policy in which each developer is required to run a command line build before committing changes to the repository. This step helps ensure that any discrepancies between the IDE and build environments are detected immediately rather than after the commit.

While IDEs are useful tools—the tools that developers spend the majority of their development days working in—developers should be aware of the capabilities of their build tools and use their IDEs to support integration build, rather than optimizing their process for what works best in the IDE.

## Postscript

In anticipation of making some code changes, Jim updates his source tree and opens up his project in his IDE, which is configured to synchronize from the Maven POM file. He makes his changes and runs tests, but before committing his changes, Jim runs a final build using Maven from the command line. Since his IDE was configured from settings in the build scripts, it works perfectly. Jim checks in his code, waits for the message from the integration machine that all is well, and then goes to lunch.

After talking to the release engineer, Mary learns how to have her IDE run the Maven process-resources goal before a build. Having learned the power of resource filtering, she gets to work on making her code more modular and configurable.

By viewing the build tool as the primary source of definition for a project and using the IDE as a developer productivity tool to keep the build running, teams can work effectively and reduce duplication of effort. **{end}**

## Sticky Notes

For more on the following topics go to [www.StickyMinds.com/bettersoftware](http://www.StickyMinds.com/bettersoftware).

- Continuous integration
- Tools



## Wind River Test Management

### *A Collaborative Automation Solution*

Software quality assurance has never been more important—or more difficult.

Wind River Test Management brings teams together with a collaborative automation solution, specifically designed for embedded device test and diagnostics. Its unique run-time analytics help you focus your time and resources on what really needs testing, so you can deliver higher-quality products on time and on budget, and with greater confidence.

Trust the device software leader.  
Wind River.

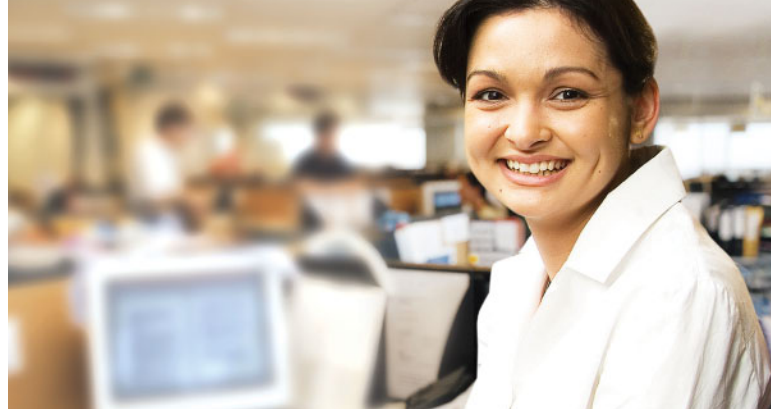
Learn more:  
[www.windriver.com/products/test\\_management](http://www.windriver.com/products/test_management)

Be sure to visit the Wind River booth at the STARWEST Expo, October 5–9, 2009, at the Disneyland Hotel, Anaheim, Calif.  
[www.sqe.com/starwest](http://www.sqe.com/starwest)

# WIND RIVER

© 2009 Wind River Systems, Inc. The Wind River logo is a trademark, and Wind River is a registered trademark of Wind River Systems, Inc. Other marks are the property of their respective owners.





## TOP 10

## Functional Testing Tips

Functional testing verifies the application behaves correctly from the user/business perspective and according to the requirements used to develop the application. TCT Computing Group is a leading functional and performance test consulting and training organization. With almost 10 years as a Mercury/HP partner and reseller, as well as a Borland® and AutomatedQA partner, TCT Computing has extensive experience implementing best practices in testing. Here are a few issues organizations often bring up and ways to address them.

### FREE WEBINAR! TOP 10 FUNCTIONAL TESTING TIPS

Thursday, November 12, 2009 at 1:00PM ET

Register at [www.TCTcomputing.com/webinars/functional](http://www.TCTcomputing.com/webinars/functional)

#### 1 We have never done any formal testing on our applications. Where do we start?

Start with a testing strategy. The test strategy defines how testing is to be performed. The strategy contains items such as: (a) objectives, (b) various types of testing to be performed, (c) testing schedule, (d) resource planning, (d) what should be tested and how, (e) automated tools to be used, (f) quality metrics needed, (g) test procedures that need to be created with each project.

#### 2 How do we start functional testing?

Start with analyzing, prioritizing and planning tests based on the functional testing requirements. The test goals are the measures used to determine if the application has been developed or upgraded to do what the end-user wants as specified in the requirements.

#### 3 Our developers do unit testing. Why is functional testing necessary?

Unit testing is done by the developers to validate small sections of code. This is very different than testing paths through the code that simulate a real end-user. Functional testing ensures that possible sequences through critical business functions are able to complete successfully.

#### 4 What should be functionally tested?

In an ideal world, everything should be tested. However, it is not possible to test any complex software system completely in a reasonable amount of time. Therefore, focus on testing critical functions, data variants, most-used functionality, and functions that have significantly changed or include new features to the application.

#### 5 What type of data should be included when testing?

Test with positive and negative data. Don't always test the "happy path." When testing with negative data, ensure the

application produces the expected error messages. Include test data that you wouldn't expect a user to enter. For example, what happens when you enter "00/00/0000" or "08/32/2009" in a date field?

#### 6 The manager wants me to change a test case. What should I do?

Be flexible! Applications may change due to requirements or environment changes. As a result, the tests may need some modifications to reflect the changes. Be sure to include adequate time for test maintenance and retesting.

#### 7 I found a defect! What next?

Be diplomatic when identifying and reporting defects. Remember that a defect is a problem that was discovered after the software was released to the test or production environment and does not meet the end-user specification. Developers demonstrate pride in their work. They work hard to deliver an application that is error free, functions according to the customers' requirements and is delivered on time.

#### 8 So you found a defect and reported it. Now what?

Track defects by using a defect tracking process that is easy to implement and follow by the project team. Using this process, follow up on reported defects and make sure the developer can reproduce and fix it.

#### 9 When should we get involved in functional testing?

Engage the functional testing team early in the development life cycle. Testers can begin gathering the testing requirements and begin the test design well before the application is ready. This allows the testers to begin testing when the application has migrated to the test environment. You can save a significant amount of time and money by starting early.

#### 10 Is the test environment ready for testing?

A test environment that is not configured correctly can be frustrating. The test environment should be separate from the development environment and be as close to the production infrastructure as possible. Work with the infrastructure team to ensure the test environment is really ready for testing.

#### Having issues with your functional testing?

Let the experts of TCT help you with your next functional test. TCT has a refined discipline and methodology with a proven track record.

### Changepoint Agile Accelerator

GARDEN GROVE, CA—Compuware Corporation announced at the Vision Events Project Portfolio Management Summit that it is helping its customers more effectively manage agile development efforts by offering the Agile Accelerator, a pre-configured version of Changepoint, its IT portfolio management solution.

The Changepoint Agile Accelerator provides:

- Best practices for managing backlog and for adding requirements to sprints and iterations
- Scrum project template
- Sprint management configuration and reports
- Artifact management
- Key reports including task boards, sprint task detail, and burn up and burn down reports

Visit [www.compuware.com](http://www.compuware.com) for more information.

### Gomez Web and Mobile Applications

LEXINGTON, MA—Gomez, Inc. has announced a major platform-wide upgrade, simultaneously releasing integrated enhancements to its Web load testing, Web performance management, and Web cross-browser testing solutions for optimizing Web and mobile applications. For businesses delivering rich Web and mobile applications, the upgraded Gomez software-as-a-service platform provides the most integrated and cost-efficient solution for optimizing the performance and quality of Web 2.0, rich Internet applications (RIA), and streaming and mobile applications. This upgrade includes new technology to test RIA transactions, an expanded family of multi-browser testing agents that precisely measure Web performance for different browsers, and an innovative new diagnostics dashboard that reduces the time to discover the root cause of a performance issue.

Visit [www.gomez.com](http://www.gomez.com) for additional information.

### Perforce SCM System

ALAMEDA, CA—Perforce Software has announced the availability of release 2009.1 of its software configuration management system. The latest version features support for previewing Web pages and multimedia files as well as the ability to customize the cross-platform interface to individual requirements.

With this release, Web developers and artists can immediately preview changes to Web, video, and audio content from within Perforce's cross-platform graphical interface, the Perforce Visual Client. A preview tab in the Perforce Visual Client automatically displays the version of the content selected by the user.

Visit [www.perforce.com](http://www.perforce.com) for more information.



Find Words that Work at [StickyMinds.com](http://StickyMinds.com) Blogs

# Implementing HP Quality Center Software

## A Customer Perspective

### “Best Practices for Quality Center”

November 5, 2009

Putting our  
imprint on  
technology  
since 1921

Time is of the essence, resources are few, deadlines loom, and leveraging what works becomes critical. Take a short break and join HP Software and Avnet on a customer journey. Learn about the “best practices” around HP Quality Center Software and how implementing new testing tools or process starts with thorough thought and planning. We’ve learned that sometimes the best tools alone aren’t enough; well-defined processes aren’t enough; and insightful plans aren’t enough. So, whether you’re an HP customer or a perspective customer ~ join us for this informative webinar and ensure your requirements, questions, and concerns are understood and addressed from a lifecycle approach to quality.

**[hpsoftware@avnet.com](mailto:hpsoftware@avnet.com)**



# Agile Development Practices Conference

## November 9-13, 2009

# Orlando • Florida

## Rosen Shingle Creek

**[www.sqe.com/adp](http://www.sqe.com/adp)**



## Conference Sponsor



**Combine Discounts & Save Up to \$600!**  
**Groups Save Even More**

- **4 Keynote Speakers**
- **26 Pre-conference Tutorials**
- **30 Concurrent classes**
- **Networking Events**
- **Agile Testing Workshop**
- **Lean-Agile Scrum Master Training**
- **APLN Agile Leadership Summit**
- **and More!**

**WIN A FREE  
CONFERENCE REGISTRATION!**

Go to [www.sqe.com/adp](http://www.sqe.com/adp) for a chance to win 1 of 5 free registrations to Agile Development Practices 2009!



# BUILD YOUR CONFERENCE!

Conference schedule includes pre-conference tutorials, keynote presentations, concurrent sessions, summit sessions, Agile Testing Workshop, bonus sessions, and more!



## MONDAY – TUESDAY

Choose from 26 half- and full-day tutorials that allow you to learn in-depth about specific topics. Or attend the new Agile Testing Workshop.



### Popular Tutorials Include:

ADAPTING to Agile: A Guide to Transitioning  
Advanced Agile Project Management  
Leading Successful Projects in Challenging Environments  
Agile Product Planning: Building Strong Backlogs  
Writing Effective Use Cases in the Agile Age

Principles and Practices of Lean Agile Development  
The Beginner's Mind: Keeping Your Agile Adoption Fresh  
The Bridge to Agility for Traditional Project Managers  
Getting Agile with SCRUM  
User-Centered Software Development

**NEW**  
— for —  
**2009**

### Two-day Agile Testing Workshop on Monday and Tuesday

Explore the testing skills, approaches, and techniques that will help make your projects more successful at the new two-day Agile Testing Workshop Monday and Tuesday. Plus take the 6 agile testing classes on Wednesday and Thursday in our Testing & Quality track, for a full week of learning!

## WEDNESDAY – THURSDAY

4 Keynote Presentations  
30 Concurrent Sessions  
Networking EXPO  
Special Events  
Bonus Sessions  
...and More!



Covering topics like:

User Stories for Agile Requirements  
Transitioning to Agile  
Key Factors in Agile Testing Success  
Scaling Scrum and XP  
Managing Iterations

Patterns of Mature Agile Teams  
Agile Peer Code Review  
Agile Architecture  
Realistic Test-Driven Development  
Kanban—Integrating Lean and Agile



# KEYNOTES BY INTERNATIONAL EXPERTS



## Beyond Scope, Schedule, and Cost: Rethinking Performance Measures for Agile Development

*Jim Highsmith, Cutter Consortium*



## Agile Brushstrokes: The Art of Choosing an Agile Transition Style

*Joshua Kerievsky, Industrial Logic, Inc.*



## Agile: Resetting and Restarting

*Alistair Cockburn, Humans and Technology, Inc.*



## Navigating Conflict on Agile Teams: Why "Resolving" Conflict Won't Work

*Lyssa Adkins, Cricketwing*



***"I am excited to take so many great ideas back to work with me. The facility was first-class. All of the speakers I had, tutorial and concurrent [sessions], were experienced in their fields and had valuable insight to share."***

— Agile Development Practices Delegate

## FRIDAY

### Agile Leadership Summit

Add a fifth day to your conference event and attend the co-located Agile Leadership Summit, in cooperation with the APLN, Thursday evening and Friday. The Agile Leadership Summit gives you the opportunity to learn from industry leaders who have embraced agile principles and put them to work successfully in their organizations.

#### Leadership for Changing Environments

*Sandi Parkes, Ph.D., VP and Dean of Continuing Education, University of Utah*

#### Collaborative Leadership Management

*Chris Matts, Program Manager, Calyon UK*

#### Maturing Your Organization's Agile Adoption

*Robbie Mac Iver, Principle Consultant, The Mac Iver Group and Bryan Campbell, Senior Program Manager, BMC Software Inc.*

#### Leading Agile in a Large Enterprise: Overcoming the "Horror" of Self-Managed Teams

*Robert Begg, Lean/Agile Development Coach, Software Engineering Transformation and Improvement, IBM*



# Ways to Save on Your Conference Registration



## Special Early Bird Offer!

Receive up to \$300 off the regular conference registration fees if payment is received on or before October 9, 2009.

## Buy One, Get One Half Off

Register two people at the same time and save half off the second registration. The 50% savings will be taken off the lower of the two registration amounts. To take advantage of this offer, please call the Client Support Group at 888.268.8770 or 904.278.0524 or email them at [sqeinfo@sqe.com](mailto:sqeinfo@sqe.com) and reference promo code **BOGO**.

## PowerPass Discount

PowerPass holders receive an additional \$100 off their registration fee.

## Alumni Discount

Agile Development Practices alumni receive up to an additional \$300 discount off their registration fee.

## Scrum Master Training + Conference

If you attend the co-located Lean-Agile Scrum Master Training Course and the Conference, you save an additional \$500 off the training class plus conference registration fees.

## Groups of 3 or more Save 25%

Register a group of three or more at the same time and save 25% off each registration. To take advantage of this offer, please call the Client Support Group at 888.268.8770 or 904.278.0524 or email them at [sqeinfo@sqe.com](mailto:sqeinfo@sqe.com) and reference promo code **GRP3**.

Check out all the Ways To Save at:  
[www.sqe.com/agiledevpractices/WaysToSave.aspx](http://www.sqe.com/agiledevpractices/WaysToSave.aspx)

*Please Note: We will always provide the highest possible discount and allow you to use the two largest discounts that apply to your registration.*



## THE EXPO November 11-12

### Visit Top Industry Providers Offering the Latest in Agile Development Solutions

Looking for answers? Explore the Agile Development Practices EXPO, designed to bring you the latest solutions in technologies, software, and tools covering all aspects of agile software development.

- Gather information to compare the latest solutions available in the industry
- Attend technical presentations and demos
- Meet one-on-one with some of today's most progressive and innovative organizations

For Sponsor/Exhibitor news and updates, visit [www.sqe.com/adp](http://www.sqe.com/adp).

#### Conference Sponsor:



#### Industry Sponsors:



#### Media Sponsors:



[www.sqe.com/adp](http://www.sqe.com/adp)

COMBINE DISCOUNTS & SAVE UP TO \$600



# Things You Might Not Know About

## Software Performance Testing

by Dale Perry

- 1 **PERFORMANCE TESTING IS FUNDAMENTALLY DIFFERENT FROM FUNCTIONAL TESTING.** In functional testing, we are interested in the user perspective: Does the software provide the required or requested functionality? In performance testing, we are interested in system characteristics. Many of these system elements will affect how the software is perceived but are indirectly related to functionality—e.g., a slow transaction will annoy the requester but the function will still work.
- 2 **THE ROLE OF THE PERFORMANCE TESTER IS SIMILAR TO THAT OF A CONSULTANT.** A well-planned performance test requires the involvement of all key personnel—DBA, system administrator, network administrator, developer, etc. The tester understands the test issues and requires the support of others to comprehend the technical issues. It is a rare individual who fully understands all aspects of a system—application, network, database and file system, operating system, etc.
- 3 **THERE IS A CRITICAL DIFFERENCE BETWEEN PERFORMANCE GOALS AND BUSINESS GOALS.** Many people confuse the two, and this misunderstanding negatively impacts the viability of the performance test. The fundamental difference between the two is simple. If you cannot measure a system-based resource to assess the performance, it is not a performance goal—i.e., a goal is to improve user productivity. However, there is no system resource you can measure to prove this goal has been met. The system may be performing fine, but the user is just slow in using the application.
- 4 **PERFORMANCE TESTING REQUIRES EXTENSIVE TIME AND EFFORT.** To be successful, a performance test has to begin at the start of the project with planning, analysis, and design. This is especially true for rapid development projects (agile, etc.). The remaining six areas listed below take considerable time to analyze and plan the testing. Trying to attempt a first-time performance test one to two weeks (or days) before shipping is insane.
- 5 **PERFORMANCE TESTING REQUIRES AN ENVIRONMENT THAT RESEMBLES AS CLOSELY AS POSSIBLE THE ACTUAL TARGET ENVIRONMENT.** Many aspects of the application and system will behave differently on varying-size environments. Many aspects of applications and systems are non-linear, so methods such as extrapolation are highly dangerous and typically invalid from the start. Extrapolation is a linear predictive model; systems tend to be nonlinear.
- 6 **LOAD TOOLS ARE THE LEAST IMPORTANT ISSUE IN PERFORMANCE TESTING.** If you do not understand the key issues and aspects of the test, especially those listed below, a tool will give you numbers that have no real meaning. Tools provide answers; if you don't know the question, what good is the answer? Additionally, there are three types of tools that must interoperate successfully: load generators, monitors, and response time tools (end to end). There are numerous issues that must be addressed for tools to be useful.
- 7 **A SUCCESSFUL PERFORMANCE TEST REQUIRES A PROPER LOAD DEFINITION—AN OPERATION PROFILE (OP).** All too often I hear people refer to performance as a simple calculation:  $\text{Load} = \text{Volume} / \text{Time}$ . This is where many tests fail. Volume of what? And more importantly, over what time period? Key elements of an OP are the time period, the number and types of activities in each time sample, and the frequency and distribution of those activities. The interaction of events is often the cause of performance problems. Failure to have an accurate model of event (user) behavior renders the test invalid.
- 8 **THERE ARE MANY TYPES OF TESTS THAT WE CAN RUN.** There are tests to measure specific resources (CPU, memory, bandwidth, etc.), assess response time, and press the system to its limits (stress). The types of tests in which we are interested will be driven by our performance goals, possible environmental issues, etc. Just running a lot of events and measuring system elements is not performance testing.
- 9 **MANY TESTS FAIL BEFORE THEY BEGIN BECAUSE THERE ARE NO MEASURES DEFINED TO INDICATE WHAT CONSTITUTES POOR PERFORMANCE.** What are poor response times, excess load, and limits on memory and CPU? These measures need to be understood at the beginning of the test so the tests can indicate where and when problems are occurring.
- 10 **TOOLS PROVIDE ANSWERS.** The issues noted above are essential to ensure that your tools provide useful information. Most load generation tools generate dozens of measures. These numbers make great graphs but are of little use if you don't know what you are looking for. If you want to get maximum value from your tools, you need to address the previous nine items on this list. Tools are great, but a fool with a tool is still a fool.

# Next Generation Automated GUI Testing



## Object-based Capture & Replay Editor

- ✓ Maintainable recordings via the actions table editor
- ✓ Integration of Ranorex repositories



## Automated Testing of Web & Windows Applications

- ✓ Winforms / C# / VB.NET
- ✓ Flash / Flex / Win32 / MFC
- ✓ WPF / Silverlight / Web 2.0 / AJAX



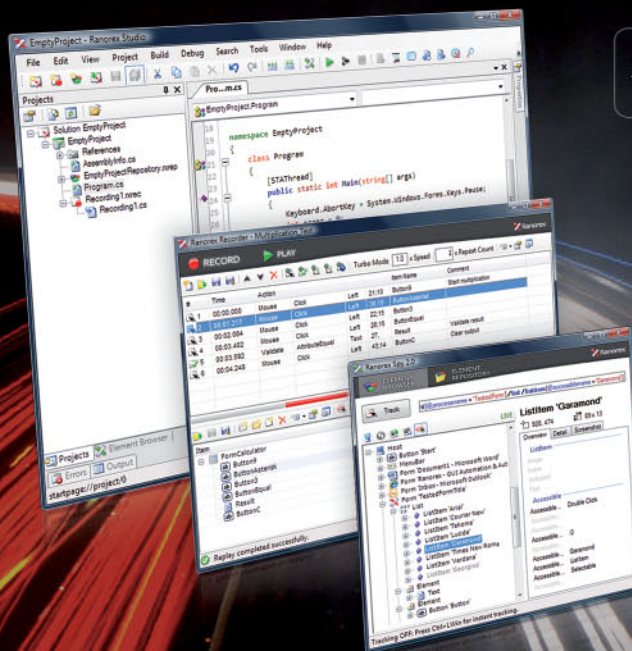
## Maintainable GUI Object Repositories

- ✓ Easy to maintain all types of GUI objects
- ✓ Separate test automation from GUI identification



## Write tests in C#, VB.NET and IronPython

- ✓ Automatic and flexible code generation in C#, VB.NET and IronPython
- ✓ All-on-one test environment with code editor, code completion and debugging



Get your 30-day Trial  
[www.ranorex.com](http://www.ranorex.com)

Visit us at  
Booth #28



# The State of the Practice

by Ross Collard

We all have unrecognized biases and unquestioned assumptions that shape our beliefs. With regard to software, we need a realistic, objective sense of where testing and quality are today—before we determine where we want to be and how to get there.

I present here contrasting overviews of the current situation, labeled the best and worst of worlds, based on different sets of assumptions. These may help you examine your own opinions and their underlying beliefs. If you are interested in participating in an ongoing debate of this topic, see the links at the end of the article.

## The Best of Worlds?

### CLAIM #1: SOFTWARE IS A FORMIDABLE ACHIEVEMENT.

Software is mastering intellectual and creative pursuits once considered the exclusive domain of humans. These activities include leisure (chess, movie animation), environmental science (simulating global warming, finding new oilfields), politics (predicting elections, war games), medicine (brain scans, psychotherapy, mapping the genome), and infrastructure (smart cell phones, driverless automobiles). Software influences our very concept of civilization.

**This revolution could not happen without dependable quality.** Software problems per hour of use—especially critical problems—are already low and declining. Quiet breakthroughs in places like Microsoft's research labs, for example, enable expert-advisor systems to detect specification and coding bugs as soon as developers introduce them.

**Organizations have major incentives to excel in software testing.** When performed and presented competently, test return-on-investment calculations often show compelling justification. Quality leads to user satisfaction and trouble-free live operation, enhances profits and corporate images, and enables firms to capitalize adroitly on opportunities.

### CLAIM #2: SOFTWARE TESTING HAS EVOLVED TO A POWERFUL BUG COUNTEROFFENSE.

Software operates in an amazingly rich diversity of contexts. Without context, my claims are meaningless generalizations. We can give meaning and substance to them by asking about each claim—In which situations is it likely to be true? When and where is it unlikely to be true?

Testers contribute significantly to their employers and society. Their contributions are recognized and rewarded. Testing provides intellectual challenges, honing critical thinking and growing investigative skills. With the widespread deployment of test automation and the increasing effectiveness of that automation, test coverage is becoming more extensive and testing more efficient. There are contexts in which these statements are true, though based on my observations I doubt they are all mainstream.

## The Worst of Worlds?

### CLAIM #3: TODAY'S SOFTWARE PRACTICES IMPEDE QUALITY.

Failing to deliver on time is seen as the greatest problem facing the software profession. This is closely followed by—and often associated with—last-minute, unpleasant surprises, reduced features, incomplete testing, cost overruns, and, of course, poor quality. The game goes to the quick, with few second chances for latecomers.

**“Let's build it and see if it falls down.”** Building software today is analogous to cathedral building. When cathedrals failed in Reims and San Juan Capistrano, a common rationale was: “God is angry.” Agnostic NASA scientists have been heard invoking the same rationale for modern cathedral equivalents like the Mars mission and the Hubble scope.

**While we do not understand software well, our knowledge of bugs and their underlying causes is particularly poor.** We confuse failures (external symptoms)

with faults (internal causes). The relationships among failures and faults are not one to one, but complex and many to many.

### CLAIM #4: TESTERS RECEIVE LITTLE RESPECT.

Testing is a choice career—if your alternative is flipping burgers. Polls find that significant percentages of testers feel underappreciated, with little influence.

Many believe anyone can test; it does not require any specific expertise. In the TV series “The Outer Limits,” a scientist inserts software-driven nanobots into a patient's brain. Reacting frantically when the bots run amok, the scientist loses track of his software changes—which rapidly obscure his earlier changes and introduce unintended side effects. The scientist tests live, inspired by desperation as his patient becomes gruesomely sicker. He must have skipped the ten-minute lecture on software testing in his MD/PhD/Board Certification program. The TV series is fiction; truth is stranger.

Biases are widespread in software, with consequences that can be dangerous. Statements like “software will always be buggy” are insidious because we are immersed within a culture with its own common sense and conventional wisdom. We do not see these biases, let alone question them.

### CLAIM #5: TESTERS (SOMETIMES) DESERVE LITTLE RESPECT.

Testing is still a craft, not an engineering discipline or science. Are software testing practices effective? Testers' infamous, friend-losing answer to all questions is: “It depends.” My surveys show that about 60 percent of knowledgeable test practitioners assess current practices as mediocre, with only 10 to 15 percent (the lunatic fringe?) optimistic about test effectiveness.

Quality has multiple dimensions, such as usability, security, and recoverability. Some dimensions may receive



inadequate consideration on any particular project. For example, software for straightforward tasks should be much easier to use than today's norms. Hundreds of millions of frustrated users endure unfriendly software every day.

**Questioning status quo shibboleths can be dangerous.** In the Vatican in 1508, Saint Bruno was burned at the stake for various heresies including questioning if the Earth was flat. A modern equivalent is questioning the value of testing certifications or, depending on which circles you travel in, *failing* to question their value.

### CLAIM #6: QUALITY IS POOR AND BECOMING WORSE.

By any defensible accounting, the quality of many systems is unacceptable. Bugs can be deadly, as in the Therac-25 disaster in which a software error is believed to have caused a medical device to administer lethal doses of radiation. Even when lives are not lost, financial costs can be sobering. A service outage at eBay caused the market value of its stock to plummet \$6 billion in two days.

**Expectations appear to be increasing for software quality.** As software becomes more pervasive, demands for always-on, 24/7/366 services are more voracious.

**Quality may be declining.** Software complexity, interoperability, and dead-

line pressures appear to be increasing—and, consequently, so is the number of bugs buried in systems.

### How Can Software Improve?

#### CLAIM #7: THERE IS NO CLEAR PATH TO BETTER SOFTWARE.

**Eager believers expect imminent breakthroughs will solve quality problems.** Variants of the theme include: “We don’t have to test because we have this next great thing, Fortran” (1950s); “because we follow the waterfall method” (1960s); “relational databases” (1970s); “object-oriented programming” (1980s); “extreme programming” (1990s); “automated testing” (2000s); “cloud computing” ...

**Testing and quality improvement practices are evolving at a glacial pace.** They are not keeping up with the ongoing revolution in software—e.g., how do you test a search engine? Our test methods today were mostly known twenty-five years ago.

**Most innovations in software testing and quality improvement over the past fifty years have been too simplistic to work well.** The list is long: five nines, software reliability engineering, cyclo-matic complexity, mutation analysis, Six Sigma, debugging, function points, etc.

**Significant software quality breakthroughs require paradigm shifts.** These shifts can be sudden or require a long gestation period but ultimately are disruptive. The more subtle shifts are melded into the existing status quo, becoming an integral part of a new world order. Agile testing is an excellent example. In the meantime, we must wait and practice Kaizen: continuous incremental improvement.

### Ongoing Debate

There are two ways to participate in the debate about whether this is the best or worst of worlds:

- Takethesurvey: [www.StickyMinds.com/testsurvey](http://www.StickyMinds.com/testsurvey).
- Attend the ongoing series of panel discussions at the STAR conferences.

Results will be released at STARWEST 2009. You do not need to attend the conference to receive a copy, but you must complete the survey. {end}

## Index to Advertisers

Agile Development Practices	<a href="http://www.sqe.com/ADP">www.sqe.com/ADP</a>	37-40
Avnet	<a href="mailto:hpsoftware@avnet.com">hpsoftware@avnet.com</a>	36
Cognizant	<a href="http://www.cognizant.com">www.cognizant.com</a>	2
Conformiq	<a href="http://www.conformiq.com">www.conformiq.com</a>	8
Hewlett-Packard	<a href="http://www.hp.com/go/software">www.hp.com/go/software</a>	Back Cover
IBM	<a href="http://www.ibm.com/delivery">www.ibm.com/delivery</a>	16
MKS	<a href="http://www.mks.com">www.mks.com</a>	22
Rally Software	<a href="http://www.rallydev.com/bsm">www.rallydev.com/bsm</a>	Inside Front Cover
Ranorex	<a href="http://www.ranorex.com">www.ranorex.com</a>	42
Seapine	<a href="http://www.seapine.com">www.seapine.com</a>	1
Serena	<a href="http://www.serena.com">www.serena.com</a>	29
SQE Training—Agile	<a href="http://www.sqetraining.com/agile">www.sqetraining.com/agile</a>	23
SQE Training—Testing	<a href="http://www.sqe.com/go?BSF20">www.sqe.com/go?BSF20</a>	11
STARWEST 2009	<a href="http://www.sqe.com/STARWEST">www.sqe.com/STARWEST</a>	5
StickyMinds.com Blogs	<a href="http://blogs.stickyminds.com">blogs.stickyminds.com</a>	35
TCT Computing	<a href="http://www.tctcomputing.com">www.tctcomputing.com</a>	34
TechExcel	<a href="http://www.techexcel.com">www.techexcel.com</a>	Inside Back Cover
ThoughtWorks	<a href="http://studios.thoughtworks.com/mingle-agile-project-management">studios.thoughtworks.com/mingle-agile-project-management</a>	7
Wind River	<a href="http://www.windriver.com">www.windriver.com</a>	33

### Display Advertising [advertisingsales@sqe.com](mailto:advertisingsales@sqe.com)

### All Other Inquiries [info@bettersoftware.com](mailto:info@bettersoftware.com)

*Better Software* (USPS: 019-578, ISSN: 1553-1929) is published seven times per year January/February, March, April, May/June, July/August, September/October, November/December. Subscription rate is US \$40.00 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](mailto:info@bettersoftware.com) or call 800.450.7854. Back issues may be purchased for \$15 per issue (plus shipping). Volume discounts available. Entire contents © 2009 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](mailto:info@bettersoftware.com).

# Wiki-powered

## Easy As Pie



## DevSpec

**Wiki-powered requirements management**

Implementing a requirements management solution can be challenging if it's not intuitive and easy to use. Experience the DevSpec difference:

### Easy

- DevSpec's Wiki interface makes adoption easy

### Powerful

- Fully-definable graphic workflow
- Completely customizable user interface

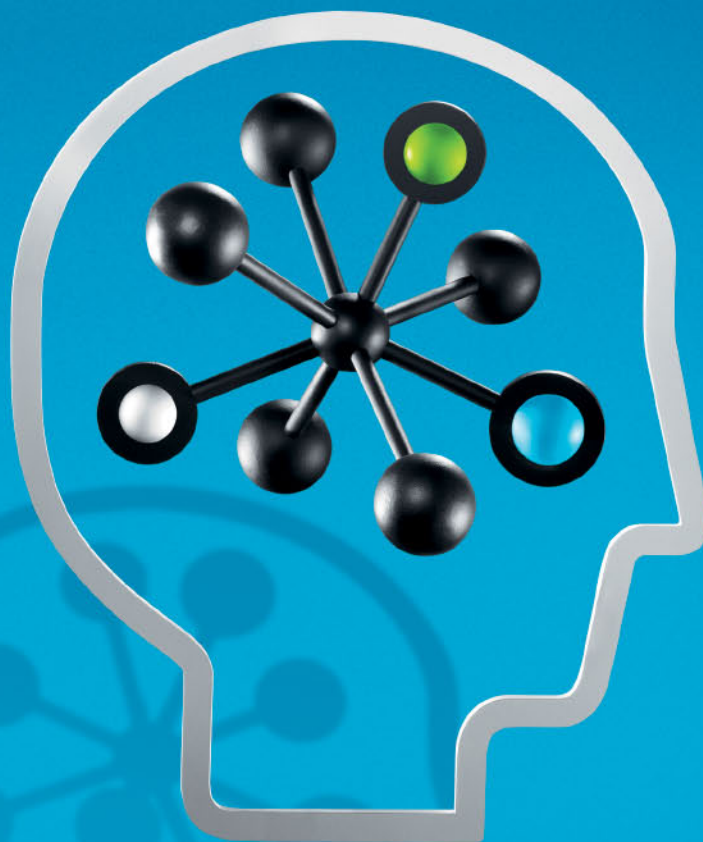
### Full Traceability

- Integrated with DevTrack for requirement-driven development
- Integrated with DevTest for requirement-driven testing
- Integrated with Community so your customers are part of your requirements team

**TechExcel** [www.techexcel.com](http://www.techexcel.com)



© 2009 TechExcel, Inc., All rights reserved.



ALTERNATIVE THINKING ABOUT APPLICATION LIFECYCLE MANAGEMENT:

## Computers Don't Run Your Apps. People Do.

Alternative thinking is looking beyond the development cycle and focusing on customer satisfaction. Because the real application lifecycle involves real people – and the customer's perception is all that matters in the end.

HP helps you see the big picture and manage the application lifecycle. From the moment it starts – from a business goal, to requirements, to development and quality management – (and here is the difference) – all the way through to operations where the application touches your customers.

HP ALM offerings help you ensure that your applications not only function properly, but perform under heavy load and are secure from hackers. (Can't you just hear your customers cheer now?)

Technology for better business outcomes. [hp.com/go/alm](http://hp.com/go/alm)

