

**“We need to make sure that
our company's most important
business gets done every
two weeks.”**

Erik Huddleston
CTO
Inovis



“Our business is dependent on tight turnaround times. **Rally makes it easy to juggle scope, hours and resources, so we can share up-to-the-minute project information with all of our stakeholders.** Everyone can see priorities, status and roadblocks to save time and money.”

See how development organizations like Erik's are delivering software the Agile way: www.rallydev.com/videos/

May/June 2010

\$9.95 www.StickyMinds.com

BETTER SOFTWARE

MISTAKES HAPPEN

Turn lemons
into lemonade

WEEKEND WARRIORS

Testing for the fun of it

The Print Companion to 

Once Upon a Retrospective

Agile Lessons from Children's Stories



**TAKE OUR
AGILE DEVELOPMENT
SURVEY!**

**“We need to make sure that
our company's most important
business gets done every
two weeks.”**

Erik Huddleston
CTO
Inovis



“Our business is dependent on tight turnaround times. **Rally makes it easy to juggle scope, hours and resources, so we can share up-to-the-minute project information with all of our stakeholders.** Everyone can see priorities, status and roadblocks to save time and money.”

See how development organizations like Erik's are delivering software the Agile way: www.rallydev.com/videos/

www.seapine.com/betterswift
Satisfy your quality obsession.

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**

 **Seapine Software™**

QA Wizard® Pro
Automated Testing

Seapine CM®
Change Management

Surround SCM®
Configuration Management

TestTrack® Studio
Test Planning & Tracking

TestTrack® TCM
Test Case Management

TestTrack® Pro
Issue Management

GIVE SPREADSHEETS THE BOOT!



DevTest Studio

The #1 choice for test management and defect tracking.

DevTrack

Use DevTrack to track defects/issues

- Track each issue through a definable workflow
- SCM integration-track fixes against their source code deliverables
- Deploy a resolution across multiple releases, versions and products
- Reporting and metrics to illustrate the entire defect lifecycle

DevTest

Use DevTest to manage your testing

- Create a central repository for your test cases, Knowledge items and automation scripts
- Schedule releases and test cycles using a wizard-driven interface
- Execute test assignments and submit defects from the same interface
- Track results with real-time dashboards and reports

TestLink

Use TestLink to automate your testing

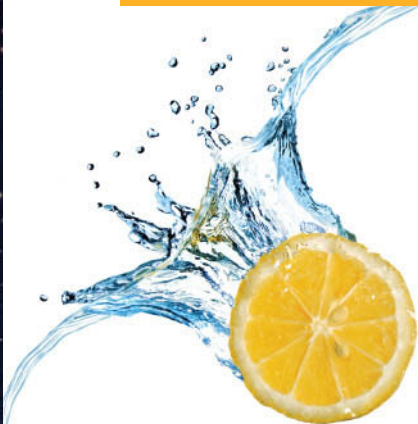
- Add automated tests to the DevTest test library
- Schedule automated tests along with manual tests
- Launch automated tests from the DevTest interface
- Track automation results with real-time dashboards and reports

Try DevTrack and DevTest live. Download free evaluation software. Watch a recorded overview demo.

www.techexcel.com



CONTENTS



Take the
Better Software
magazine digital
survey.

This month's topic
Agile Development.
pg. 17

in every issue

Mark Your Calendar **4**

Contributors **7**

Editor's Note **8**

Virtual Resource Shelf **15**

Digital Survey Results **17**

Product Announcements **33**

FAQ **34**

Ad Index **36**

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
or call 800.450.7854.

features

18 COVER STORY **ONCE UPON A RETROSPECTIVE**

Children can teach us some extremely profound things—often when we least expect it. Jennitta Andrea shares sage advice about project retrospectives that she learned while perusing the well-known children's stories on her daughter's bookshelf. These insights will help improve the way you plan, facilitate, and participate in project retrospectives.

by Jennitta Andrea

24 **LEVERAGING A LEARNING CULTURE**

Mistakes happen. It's how you respond to them that matters. Teams might react to a bug with panic and blame, leading to a quickly hacked fix and possibly more issues. Taking time to investigate and learn leverages problems into process and practice improvement and a higher quality product.

by Lisa Crispin

28 **THE WEEKEND TESTERS**

This is the true story of freedom from the monotonous work of checking. It is the story of what happened when a few passionate testers took responsibility for their own education and fun in testing. Explore how they test, learn, and contribute to the open source.

by Pradeep Soundararajan and Parimala Shankaraiah

columns

11 **TECHNICALLY SPEAKING** **UNINTENDED CONSEQUENCES** • *by Lee Copeland*

Every action elicits a response. But, sometimes that response is not what we expected. These anecdotes from industry experts are good examples of how our best intentions don't always turn out.

12 **INSIDE ANALYSIS** **A COMMON LANGUAGE** • *by Laura Brandenburg*

Do your stakeholders use different terms to talk about the same concept or use the same term but in a different way? A domain model can clarify key concepts and the relationships between them, helping your team resolve ambiguous terms and leading to more productive discussions about project requirements.

35 **THE LAST WORD** **BELIEVE THE TERRITORY** • *by Markus Gaertner*

Test plans are seldom followed as written, project plans hardly ever fit the actual progress, and process models are rarely followed to the letter. Markus Gaertner examines why most of our documents become obsolete and gives advice about whether or not to continue to write and maintain them.

MARK YOUR CALENDAR



software tester certification

www.sqetraining.com/certification

training weeks

www.sqetraining.com/public

Testing

September 13–17, 2010

Washington, DC

October 18–22, 2010

San Francisco, CA

November 8–12, 2010

Tampa, FL

June 6–8, 2010

Las Vegas, NV

June 15–17, 2010

Albuquerque, NM

Portland, OR

August 24–26, 2010

San Jose, CA

Boston, MA

August 31–September 2, 2010

Charlotte, NC

conferences



Better Software Conference

www.sqe.com/bsc

June 6–11, 2010

Caesars Palace

Las Vegas, NV

Agile Development Practices|West

www.sqe.com/adpwest

June 6–11, 2010

Caesars Palace

Las Vegas, NV

STARWEST 2010 Software Testing Analysis & Review

www.sqe.com/starwest

September 26–October 1, 2010

Hilton San Diego Bayfront

San Diego, CA

Agile Development Practices|East

www.sqe.com/adpeast

November 14–19, 2010

The Rosen Centre

Orlando, FL

STAREAST 2011 Software Testing Analysis & Review

www.sqe.com/stareast

May 1–6, 2011

Rosen Shingle Creek

Orlando, FL

BETTER SOFTWARE

Publisher

Wayne Middleton

President

Drew Thoenig

Vice President of Publishing

Holly N. Bourquin

Editor in Chief

Heather Shanholtzer

Editorial

Managing Technical Editor

Lee Copeland

Online Editor

Francesca Matteu

Online Editor

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

Subscriber Services:

info@bettersoftware.com

Phone: 904.278.0524, 888.268.8770

Fax: 904.278.4380

Address:

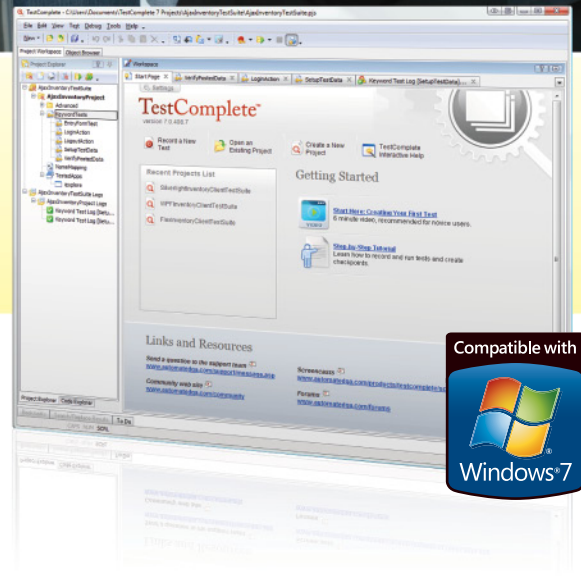
Better Software magazine
Software Quality Engineering, Inc.
330 Corporate Way, Suite 300
Orange Park, FL 32073



Can you find 7 mistakes in this photo?

Did you find them all? Answers at www.testcomplete.com/hawkeye

Fun challenge, right? Great software testing feels like that.
Now do that 5 times a day, every day, for the next year...
With the Exact. Same. Photo. Does your testing feel like that?
Time for TestComplete.



TestComplete™

Test faster. Test better. Test more.

Free yourself to do the quality work that you love: explore, investigate and discover.

TEST WITH THE BEST

TestComplete is the most exciting software test automation tool available.

TEST ANYTHING

- Web, Ajax, Flash, Flex
- Windows, .NET, WPF, Silverlight
- Java, Qt, Delphi, MS Office
- Functional, Load, Regression, Manual

TEST IT YOUR WAY

- Automate without programming via Keyword Testing
- Free extension downloads
- Professional IDE for test engineers
- Friendly and responsive support

AutomatedQA
Call Us: +1 (978) 236-7900



*Did you find
all 7 mistakes?*

Visit www.testcomplete.com/hawkeye
to see how you did and try TestComplete

NOW I CAN PUT "NO-REPRO" TO BED.

AUTOMATICALLY ATTACH CONTEXT TO BUGS.

Microsoft is making it easier to find and fix bugs. You now can add two new toolsets to help improve collaboration between testers and developers, reproduce bugs, automate repetitive tasks, and reduce the costs of setting up and maintaining multi-machine test environments.

Microsoft® Visual Studio® Test Professional 2010 with simplified test planning, manual test execution, and rich, actionable bugs not only makes your job easier, but you can start to break the silos between test and development.

Microsoft® Visual Studio® Lab Management 2010 helps teams rapidly provision virtual lab environments at a known state for test execution and build automation, effectively reducing wasted time and resources in your development and test life cycle.

WHAT WILL YOU DO WITH VISUAL STUDIO 2010?

Visit www.almcatalyst.com/test

Download Test Professional and Lab Management.

Not sure how Visual Studio 2010 can help your team?

Connect with an ALM expert

Sign up for an ALM workshop



A senior agile practitioner, analyst, and tester, [JENNITTA ANDREA](#) consults as a retrospective facilitator, agile coach, and instructor. Her multifaceted experience on more than a dozen different agile projects since 2000 is reflected in her frequent publications and conference presentations (see www.theandregroup.ca). After reading "Once Upon a Retrospective" you'll know why Jennitta's professional bookshelf contains many of her daughter's outgrown children's books.



[LAURA BRANDENBURG](#) is a business analyst consultant, author, and mentor. She has eight years of experience across technology leadership, analysis, project management, and QA. She hosts "Bridging the Gap," a blog about business analyst practices. Laura is the author of *How to Start a Business Analyst Career* and the forthcoming eBook *The Promotable Business Analyst*. Visit her blog at www.bridging-the-gap.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. He 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.



[LISA CRISPIN](#) is the co-author, with Janet Gregory, of *Agile Testing: A Practical Guide for Testers and Agile Teams* and a contributor to *Beautiful Testing*. A tester on agile teams for the past ten years, Lisa enjoys sharing her experiences at conferences and user group meetings around the world. For more about Lisa's work, visit www.lisacrispin.com.



[MARKUS GAERTNER](#) is a software tester for Orga Systems in Paderborn, Germany. Personally committed to agile methods, he believes in continuous improvement in software testing through skills. Markus cofounded the European Weekend Testing chapter, blogs at <http://blog.shino.de>, and is a black belt in the Miagi-Do school of software testing.



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



[PARIMALA SHANKARAIAH](#) is one of the cofounders of Weekend Testing and works at Consona, Bangalore as a senior tester on the weekdays. She handles public relations at Weekend Testing and cofacilitates testing sessions for the Bangalore chapter. Parimala writes a blog (curioustester.blogspot.com/) where she discusses her testing experiences. She can be contacted at parimala.shankaraiah@gmail.com.



[PRADEEP SOUNDARARAJAN](#) is a tester, coach, author, and speaker on software testing. He defines his experience as 7,000,400 mistakes and counting. Pradeep takes pride in being a colleague of James Bach and Michael Bolton, who trained him and drained bad testing ideas from his mind. He writes a blog (testertested.blogspot.com) and loves receiving email from testers: pradeep.srajan@gmail.com.



BE PREPARED

Louis Pasteur once said, "Fortune favors the prepared mind," and I couldn't agree more. No matter what the task—testing, programming, writing requirements—the most successful people are those who have educated themselves in advance of a project.

But, learning styles are as varied as the students themselves. This became very apparent to me as I was editing the articles for this issue. Fortune favored us with three feature articles that demonstrate just how differently we learn and how inspiration and enlightenment can be found in the most unlikely places.

In our cover story, "Once Upon a Retrospective," Jennitta Andrea explains how story time with her daughter led her to a memorable (and fun!) way to engage teams in project retrospectives.

Lisa Crispin turns lemons into lemonade in her article, "Leveraging a Learning Culture." When mistakes happen, find out how taking time to investigate and learn from them leads to process improvement and a better product.

And, finally, talk about prepared minds! Parimala Shankaraiah and Pradeep Soundararajan are so dedicated to improving their testing skills that they formed a group that practices *outside* of work. "The Weekend Testers" is an introduction to this inspired and inspiring bunch of likeminded individuals, who take on open source testing projects on their own time so they can grow their knowledge and expertise.

Each of the authors in this issue also has revealed the software blogs they find the most educational. Check out the Virtual Resource Shelf to see which blogs the experts are reading.

As always, I hope you enjoy this issue of *Better Software* magazine and that you find the articles to be informative and useful in your quest for quality. Email me to let me know how you've put this issue to work for you.

Happy Reading!

Heather Shanholtzer

HShanholtzer@sqe.com

Automate Your UI Testing with Ranorex





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
- ✓ WPF / Silverlight / Win32 / MFC
- ✓ Flash / Flex / Web 2.0 / AJAX /  / 



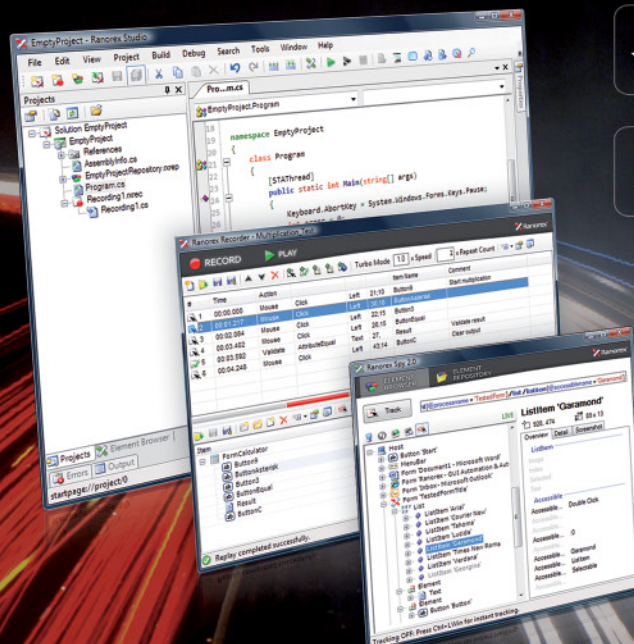
Maintainable UI Object Repositories

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



Watch Demo Video
www.ranorex.com/demo

Agile Software Development Training



Combine In-Depth Training with the Agile Development Practices Conference series and Save \$500!



June 6–11, 2010
Las Vegas, NV



Nov. 14–19, 2010
Orlando, FL

To get the most out of your multi-day training classes, combine your classes with the Agile Development Practices conferences. Delivered by top experts in the industry, training classes allow you to go in-depth in the subject of your choice. After your training concludes, attend the conference on Wednesday and Thursday and select from keynotes, concurrent sessions, the networking EXPO, and more! Plus, you save an additional \$500 when you combine any training with your conference registration.

Learn more at: www.SQETraining/Agile

Scrum Master Certification

Applying Lean-Agile Software Development Practices with Scrum

June 6–8, 2010
Las Vegas, NV

Nov. 14–16, 2010
Orlando, FL

Practical Test-driven Development

A Revolutionary Approach to Software Design and Programming

June 6–8, 2010
Las Vegas, NV

Nov. 14–16, 2010
Orlando, FL

Agile Testing Practices

People, Processes, and Tools

June 6–7, 2010
Las Vegas, NV

Nov. 14–15, 2010
Orlando, FL

PROVIDING EXPERT TRAINING TO SOFTWARE PROFESSIONALS

TOP 3 MYTHS OF AGILE TESTING

STAYING COMPETITIVE in today's economy means companies must deliver the right products to market faster and with higher quality than ever before. Adopting agile development methods is a significant way organizations are doing this. Making the switch to agile practices challenges our traditional notions of software engineering best practices specifically in the area of testing.

TO BE SUCCESSFUL, QA engineers need to contribute their unique testing perspective while developing their skills in exploratory testing and test automation. Testers on an agile team need to hold on to the benefits of traditional QA within an iterative development process.



MYTH

Everyone on an agile team tests therefore dedicated testing resourced aren't needed.



FACT

Teams interested in delivering a **quality product** realizes that the same product knowledge, preparation, discipline and skills used in traditional QA should be leveraged when transitioning to agile.

WHAT'S HOT

✓ AGILE TESTING

Agile testers are integrated, full-fledged members of the development team and participate in planning, estimating and all team activities

Agile testers work hand-in-hand with development and product management

Business requirements are written a piece at a time to accommodate changing business needs

Developers often take the lead on code-level tests while agile testers focus on acceptance test automation and building regression test plans

Throughout the release additional test scenarios are uncovered through exploratory testing

Agile testing is continuous and feedback is provided during all stages of development

Agile testing follows a fluid, continuous process with defects fixed as they are found

Agile testers are expected to radiate information and provide complete visibility into all test-related activities for the benefit of the larger group

Quality software is always ready to be delivered

what's not



TRADITIONAL QA

Testing is performed by a separate organization or group with one or more teams of QA engineers reporting to a QA manager

QA teams may work with other groups but with limited interactions and well-established boundaries

QA and development receive detailed business requirements and schedules up front

While developers write code, testers write test plans and test cases to support predefined business requirements

When development is presumed complete, the application is delivered to QA for testing

QA executes a complete cycle of tests and reports defects back to development

When testing is complete, development fixes defects and delivers another rev to QA

Traditional QA teams often keep test design, implementation and progress specifics within the QA "walls"

The process of test and fix repeats until time runs out



MYTH

Testers are second class citizens on an agile team.



FACT

Testers are valuable members of an agile team and often add a **unique perspective** in terms of identifying potential roadblocks and dependencies early in the process.



EXPLORATORY TESTING

Business requirements on an agile project may not be as concrete as requirements on a traditional project; agile methods accept that change is a healthy and real part of software development. This means that test case generation may not be as cut-and-dried as it was in the past. Exploratory testing is an essential skill to uncover additional considerations for the product owner to evaluate.



AUTOMATION

Agile emphasizes automating as much as possible, but many teams struggle with when, how much and what tools to use. While continuous integration (CI) is an accepted developer practice, the agile tester takes the lead on incorporating automated acceptance tests and creating regression test plans as part of CI.



COMMUNICATION

Traditional QA engineers tend to rely on documents. Agile testers don't get a big requirements document as a basis for test cases and don't get three months to write test cases before they see a working piece of code. They jump into the communication stream, which may be verbal, written or virtual, and assimilate the information they need.



MYTH

Important testing activities that aren't product-related won't get done.



FACT

Additional testing activities, such as performance and regression testing, are addressed through **test-oriented stories**, which are estimated, planned and tracked just like product-related stories.

3



CHALLENGES WITH TRADITIONAL QA:

- **SIGNIFICANT DELAYS** between when software is written and development receives feedback.
- **DEFECTS** found late in the process can have major implications when changed.
- **CHANGING BUSINESS REQUIREMENTS** effect test cases that have already been developed.
- **SILOED COMMUNICATIONS** create risk that different groups may have different expectations of the final product.
- **QUALITY SUFFERS** and many QA activities get left out when testing is the last activity before a fixed release date.



BENEFITS OF AGILE TESTING:

- **ON-GOING FEEDBACK** to developers allows testers to ask the right questions at the right time.
- **EARLY IDENTIFICATION** of dependencies, technical or testing challenges, and roadblocks.
- **EMBRACES CHANGE** as a healthy and real part of software development.
- **TEAM COLLABORATION** helps everyone work together toward a common goal.
- **QUALITY COMES FIRST** because final acceptance criteria are established *prior* to the work beginning.

IN CONCLUSION

Software companies today need to respond to customer and market demands rapidly to stay competitive. By practicing agile, development teams can deliver software more quickly and with higher quality. However, to be successful, QA engineers need to learn new skills so they can become key members of agile teams, as testing is still a critical activity.

Want to read more? Download VersionOne's The Agile Tester White Paper at:

www.VersionOne.com/TheAgileTester



Unintended Consequences

When our best-laid plans go astray—and they will—the results can be surprising.

by **Lee Copeland** | lcopeland@sque.com

We take action A, expecting result R (which may or may not occur), but in addition, result U occurs—an unintended consequence. Result U can be positive, but more often than not, it is negative. Sometimes, the unintended consequence can be more significant than the result we were trying to achieve. Unintended consequences are not just sometimes occurrences. In fact, the Law of Unintended Consequences states that any purposeful action will produce some unintended consequence.

A classic example is Thomas Austin's release of twenty-four European rabbits into Australia for hunting in 1859. This single act led to the explosive growth of the rabbit population, now estimated at 600 million. The most serious mammalian pest on the continent, rabbits are responsible for the extinction or major decline of many native species. Annually, descendants of these twenty-four rabbits cause millions of dollars of damage to crops.

I asked a number of industry experts to share their favorite unintended consequences stories with me. Here are a few:

Jonathan Bach wrote, "You decide that session-based test management (SBTM) is a great idea to manage and measure exploratory testing (ET). SBTM makes ET accountable and trackable. It becomes so successful that your stakeholders require you to write formal test cases based on all of the test ideas you created in your ET sessions. Since no one else is available, the task falls to your team. They go from great explorers to bored clerks, filling in templates for the next four weeks, longing for the days when they used their brains creatively."

Jean Tabaka shared, "Pushing features out too quickly causes a pile up of defects, which then results in reduced ability to push features out. In our attempt to serve too many masters, we create bloated, under-tested software. The unintended consequences are that you produce less and less that is high value, spend more time managing defects than creating new software, and make everyone unhappy."

Linda Hayes wrote, "I worked on a project that had a bonus program based on reducing the number of defects coming out of development. The previous release had been thrown over to testing before it was ready, and it elongated the test cycle and delayed the release far past the promised de-

livery date. The bonus scheme created side negotiations about whether to report a defect and whether it was, in fact, a defect (as opposed to a missed requirement, etc.) and resulted in a 'black market' of defects that went unreported. The whole thing backfired because it did not encourage fewer defects, just less reporting, and it turned developers and testers into adversaries."

John Fodeh wrote, "I remember an incident where a company introduced a bonus program rewarding the testers who found the highest number of defects (the severity of the de-

fects was also taken into account using a sophisticated weighting algorithm). The program was introduced with the best intentions—promoting testing and finding as many prerelease defects as possible. The unintended consequence was that testers stopped putting much effort into defect-prevention activities, such as requirement validation, design reviews, etc. The silliness of this program was manifested when one tester stumbled across a defect regarding an unsorted list. He reported multiple defects—first list item is wrong, second list item is wrong, etc."

Distinguished sociologist Robert K. Merton [1] described four possible causes of unanticipated consequences: ignorance (we can't anticipate everything), error (incorrect analysis), immediate interests (that override long-term interests), and basic values (that proscribe other, more beneficial actions).

While I appreciate Merton's list of causes, I think the solution is much more basic—we become so enamored with the benefits of our ideas that we don't consider how they could go wrong. The dictionary defines "perverse" as "willfully determined or disposed to go counter to what is expected or desired." Before implementing any actions, actively seek to discover the perverse consequences of those actions. You can do it. And, it will be fun! (But, beware—there's an unintended consequence to your investigation—your ideas will be rejected, and you'll be branded a negative thinker and not a team player.) **{end}**

Sticky Notes

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

■ References

"Sometimes, the unintended consequence can be more significant than the result we were trying to achieve."

A Common Language

Domain models can help foster conceptual understanding on teams, thus reducing conflict and driving productive conversations among stakeholders.

by **Laura Brandenburg** | laura@clearspringanalysis.com

Models are critical in driving understanding about what is to be achieved among a project's business stakeholders and technology implementers. Models help us communicate, and, if you're like me, you like to keep different models in your back pocket to help you facilitate different kinds of discussions or solve difficult problems.

I tend to pull out a domain model when I find that participants are getting caught up in the terms they are using or with *conceptual understanding*—a view of the business domain that identifies the key business concepts and the relationships between them within the context of an organization. Examples of business concepts are “customer” and “lead.” We know we have a shared conceptual understanding if we agree about the boundary between a lead and a customer, the relationship between these two concepts, and the attributes that are tied to each concept. A shared conceptual understanding is the foundation of fruitful discussions and clear communication about requirements.

Does Your Team Lack a Shared Conceptual Understanding?

There are some common indications that your team needs help creating shared conceptual understanding. For example, we'll often hear our business and technical teams use different terms to talk about the same ideas or use the same terms with different implied meanings. These conflicts frequently are rooted in differences between an operational, business-oriented representation of a term and a database, technology-oriented representation of a term.

Another common indicator is when you are working with multiple business stakeholders and they seem to be talking past each other. Sometimes this results in a nearly violent disagreement about some aspect of the requirements. The root of this problem is often that each person's conceptual understanding is derived from how he interacts with instances of that concept in his day-to-day work. For example, a “customer” is a deceptively simple term within the context of a specific business. A salesperson might think of a customer as any potential customer. A customer service representative might consider a customer an organization with an active product and stop providing support to past customers who have accounts overdue

by ninety days. An accounts receivable representative might be responsible for ensuring a past-due customer pays his bill. Individuals in each role interact with the “customer” in different ways and, therefore, develop contradictory conceptual understanding over time.

Using a Domain Model

A domain model logically represents the business concepts to be fulfilled by the system. A domain model should not be confused with diagrams that represent the actual or intended

database design—often referred to as entity relationship diagrams or database models. A domain model diagram uses terms from the business domain and enables participants to discuss concepts and relationships regardless of how they are represented in the formal database model for the organization's systems.

Like many of these other models, a domain model can be represented in a UML class diagram. Class diagrams can have many elements, but class, associations, attributes, and multiplicity will get you started in creating a basic domain model. In the example of a domain model diagram in figure 1, we are looking at a deliberately simple example for illustrative purposes. Here, each lead has an optional relationship with a customer. Each customer can have one or more product subscriptions and invoices. Without an actual business context, this diagram is ambiguous. Within a specific business context each element of the model will drive analytical questions about each business concept.

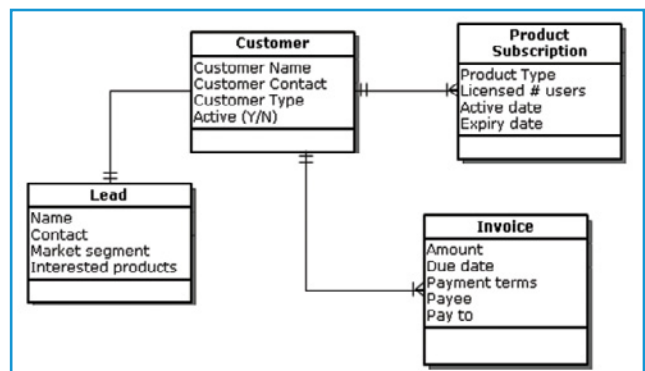


Figure 1

How Domain Models Create Conceptual Alignment

Domain models help to surface underlying conceptual disagreements and provide a tool for collaboratively developing a shared understanding. When technical terms are clouding the business domain, a relentless focus on business concepts helps participants clarify their thinking. While focusing on only one aspect of the term, you can drive productive conversation around exactly what this concept means, independent of any technical considerations.

Where business stakeholders use the same term to represent different concepts, digging into the concept's attributes will surface disagreements. For example, once we begin assigning "customer" with attributes for an expiration date for their products, we would discover the concept of "past customer," because someone from accounts receivables would say "Wait, I need to get that customer to pay his bill!" Now you can launch a deeper conversation into what customer means to each department and can begin to represent different meanings through separate concepts.

Overcoming Some Common Challenges

Although domain models are extremely useful, there are challenges in using them effectively. Business stakeholders sometimes struggle with the formality of the notation. A brief tutorial can help. I typically start with two relatively clear concepts—for example, customer and lead—and use these, with their attributes and associations, to describe what each piece of the notation means.

Another challenge is that people fail to engage in conversation for fear of saying something that appears to be misinformed. I often stimulate conversation by using a white board to redraw the model as we discuss it. This technique provides the benefit of working from a collection of known items from a pre-drafted model while encouraging changes through the easy-to-change nature of that model as drawn on a white board. I find participants are less nervous about suggesting changes when the model is redrawn. In the best scenario, one or more participants engage in redrawing the model on the white board.

Another approach I use is to start the walkthrough where I have the clearest alignment and move out toward the concepts where there is more disagreement. Starting the conversation where there is clarity provides participants with time to become comfortable with the model. Where there is less agreement, I come prepared with questions about concepts to surface conflict and stimulate conversation. I might ask about a specific attribute or how a relationship is important to a specific department. If conversation is lagging, I might summarize conflicting meanings I heard in previous conversations and ask participants to clarify within the structure of the model on which we're working.

Another challenge is that, as the facilitator, you may lack familiarity with UML. Domain models provide a good entry point into UML. The class diagram is one of the simpler notations, and four elements of the notation create a useful domain model. Any introductory text on UML should get you started. My favorite book on the subject is Martin Fowler's *UML Distilled*, and, so far, it's provided all the information I need to apply basic UML to my projects.

Because the domain model is independent of the database structure, it can be a challenge to get your developers to set aside database tables, fields, and joins. As the facilitator, it's important that you keep an ear open for technical terms and redirect them to business-focused concepts. Once technologists become accustomed to this approach to communication, they often revel in the freedom of discussing concepts without being wedded to database designs. You also are helping them take a significant step forward in understanding what the business wants.

While most teams face initial challenges employing domain models, the benefits of a shared conceptual understanding far outweigh the costs of disjointed discussions. I hope I've encouraged you to consider using a domain model to drive alignment around business concepts. **{end}**

THE Secret TO Designing Products Customers LOVE



Read the findings
from Aberdeen at
jamasoftware.com/go



build great products™

Becoming Agile in Challenging Times

Shifts in Organizational Thinking

Agile provides a great acceleration method to market, but orchestrating the change can seem challenging.

New approaches in project methodologies, aligning multiple resources, and introducing new technologies—all under time constraints—can be daunting. Join HP and Avnet to see how bringing together the different IT teams in an organization can help address concerns around less time to planning and less time to execution, and provide the right visibility into the overall status of your projects.

Putting our
imprint on
technology
since 1921

www.avnet.com



Virtual Resource Shelf

Author recommended books, blogs, gadgets, Web sites, and other tools for building better software

Q: What is your favorite software-focused blog?

Mike Griffith's *Leading Answers*

(leadinganswers.typepad.com/leading_answers/)

"Focusing on leadership and agile project management ideas, observations, and links, Mike takes the time to select valuable and thought-provoking topics. He clearly describes the topic and then sprinkles in his own



personal experiences to bring the ideas to life."

Jennitta Andrea

Curious Tester (curioustester.blogspot.com)

written by Parimala Shankaraiah.

"Parimala studies a wide range of testing approaches and practices and distills this knowledge for us. She reminds us why diversity is so important to promote innovation and creativity in software development and testing."

Lisa Crispin

Practical Analyst (practicalanalyst.com)

by Jonathan Babcock

"It's my favorite because the author shares his real-world business analyst experiences and blends practicality with passion to be a great business analyst."

Laura Brandenburg

test Obsessed

Elisabeth Hendrickson's *Test*

Obsessed (testobsessed.com)

"I like the way Elisabeth uses examples to solve many real-time problems."

Parimala Shankaraiah

Matthew Heusser's *Testing at the Edge of Chaos*

(blogs.stpcollaborative.com/matt/)

"Matt blogs on the social aspects of software delivery. Though his blog title might indicate differently, his thoughts go well beyond software testing. His blog is a regular inspiration."

Markus Gaertner

The Consulting Software Tester (www.satisfice.com/blog/) by James Bach

"James Bach is one of the most innovative testers today. His elucidation and popularization of exploratory testing is a great contribution to the field."

Lee Copeland

Want to get our authors' take on a particular resource?

Email us at editors@bettersoftware.com Subject: Virtual Resource Shelf.

Get Noticed. Get Certified.

**NEW Fall
Schedule!**

Michelle Dillon, CTFL
Certified Software Tester

**CERTIFIED
TESTER**



Software Tester Certification Certified Tester—Foundation Level Training

Empower Your Team

FALL 2010 SCHEDULE

San Jose, CA	August 24, 2010	Raleigh, NC	October 12, 2010
Boston, MA	August 24, 2010	Omaha, NE	October 12, 2010
Charlotte, NC	August 31, 2010	San Francisco, CA	October 18, 2010
Washington, DC	September 13, 2010	Atlanta, GA	October 19, 2010
Minneapolis, MN	September 14, 2010	Pittsburgh, PA	October 19, 2010
Salt Lake City, UT	September 14, 2010	Austin, TX	October 26, 2010
Philadelphia, PA	September 21, 2010	Columbus, OH	October 26, 2010
St. Louis, MO	September 21, 2010	Bethesda, MD	November 2, 2010
San Diego, CA	September 26, 2010	Palm Bay, FL	November 2, 2010
Indianapolis, IN	September 28, 2010	Tampa, FL	November 8, 2010
Baltimore, MD	October 5, 2010	Sunnyvale, CA	November 16, 2010
Toronto, ON	October 5, 2010	Phoenix, AZ	November 30, 2010

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

www.sqetraining.com/certification

PMI Project Management
R.E.P. Institute
Earn 22.5 PDUs

SQE
TRAINING
www.sqetraining.com

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

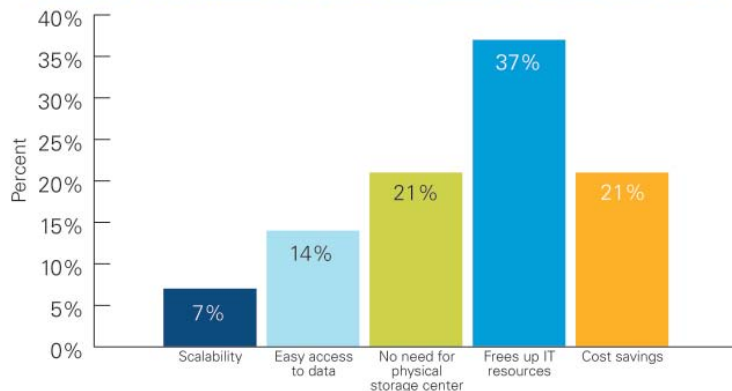
Participate in our new survey: Has Your Organization Gone Agile?

www.stickyminds.com/agileorgsurvey

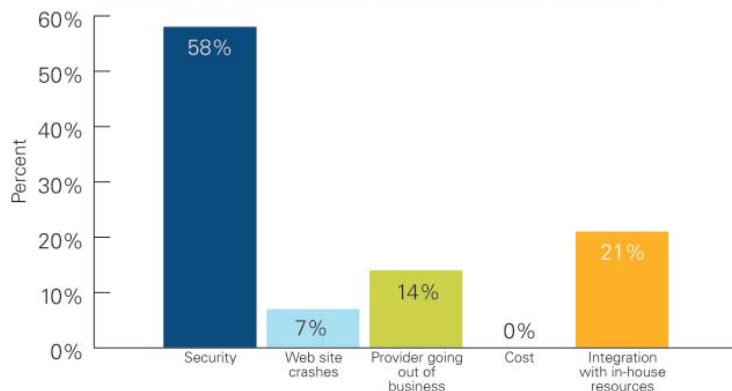
Last Issue's Survey Results:

Q: What has been your experience with cloud computing?

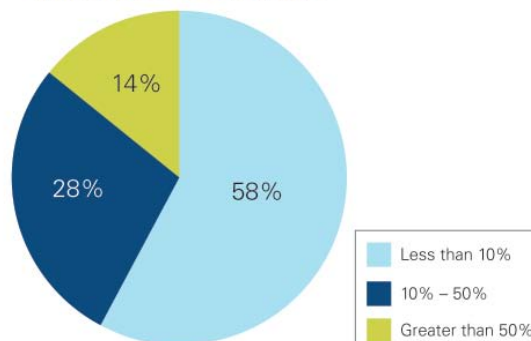
What do you consider the biggest benefit to cloud computing?



What is your main concern with cloud computing?



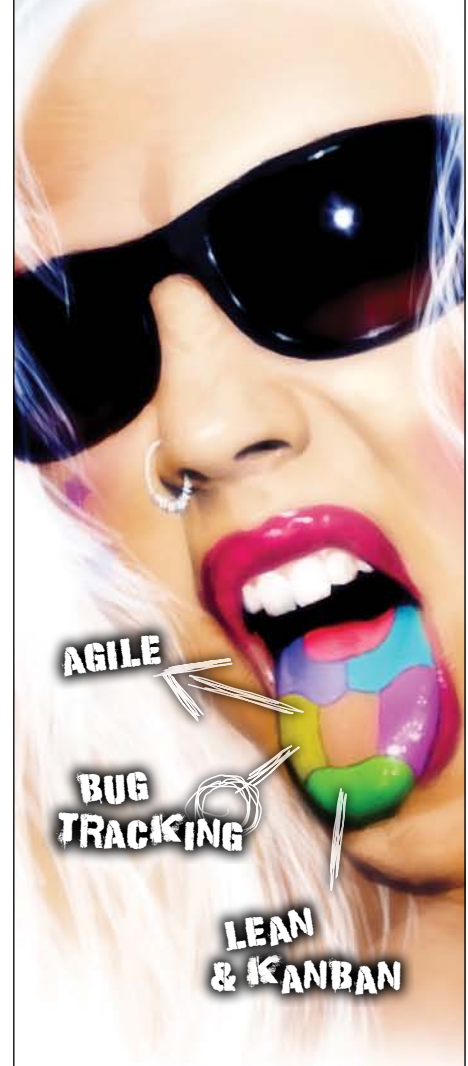
What percentage of your organization's IT resources have moved to the cloud?



We are excited...

...that Hansoft can handle both schedule driven projects and agile methods like Scrum. The possibility of combining these methods in the same project has really been useful. Hansoft makes our projects more efficient and helps us collaborate much better with partners, customers and suppliers."

General Manager,
Space Systems, Swedish Space Corporation.



HANSOFT

Download a free 2-user solution trial at www.hansoft.se

Once Upon a Retrospective

Agile Lessons from Children's Stories

by Jennitta Andrea

Retrospectives in a Nutshell

At key milestones within the life of a project, teams reflect on their strengths and weaknesses for the purpose of continual improvement. The facilitator guides participants through a variety of individual and group activities to develop a big-picture view of the project and provide a framework for detailed discussion, leading to concrete action planning. The difference between a retrospective that is considered dull and worthless and a retrospective that produces team bonding and valuable insights is in the planning and preparation.

A common retrospective structure [1] is summarized in figure 1, showing for each stage the goal, percentage of overall meeting time spent, grouping of people, and how the chairs are arranged. An effective retrospective systematically progresses through the five stages—safety, discover, analyze, plan, and close—to ensure the team’s learning is maximized and actionable closure is reached.


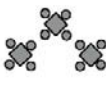

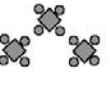

	Safety	Discover	Analyze	Plan	Close
Goal	Maximize participation	Everyone can see the “Big Picture”	<ul style="list-style-type: none"> Continue Fix Stop 	Where do we want to be? How do we get there from here?	How to document How to execute
% time	5-10%	30–50%	20–30%	15-20%	10%
Groups	Everyone (anonymous)	Affinity groups	Everyone	Cross-functional groups	Everyone
Room Setup					

Figure 1: Retrospective skeleton

	Daily	Iteration	Milestone (Release)	End of Project
Focus	Remove roadblocks	Team and process issues	Organizational learning	
Duration	30 seconds + follow-up time	1+ hour	1+ day	
	Duration depends on the length of period under review, complexity of project, size of team, and level of conflict or controversy			
Timing	During stand up	Before iteration plan (break in between)	After milestone or project completion; good to coincide with the celebration activity	
Participants	Core team	Core team	Contributors from across the organization	
Agenda	-	Activities based on goal. Incremental (reference previous)	Activities selected based on goal	
Facilitator	-	Team member, external	Experienced third party	

Table 1: Compare and contrast retrospective formats

While each retrospective follows the same basic framework described above, the time it takes and the types of activities performed should be adjusted based on context (i.e., whether this iteration covers the previous day, iteration, milestone, or the entire project), as shown in table 1.

Context (e.g., health of project or team; purpose of retrospective—continual improvement, crisis management, needs assessment, etc.—experience performing retrospectives., etc.) should drive the selection of activities [2] to perform during the safety, discover, analysis, plan, and close stages of the retrospective. If your team practices retrospectives frequently, consider varying the activities over time to maintain interest and effectiveness.

Safety

What activities are we doing?	<ul style="list-style-type: none"> • Develop team meeting norms • Safety poll
How long should this take?	5–10% of meeting
Who is working together?	Individually, anonymously
How is the room arranged?	Arrange chairs in a circle so that everyone can face each other
We're done when we have ____.	<ul style="list-style-type: none"> • Team commitment to participate • Team norms for the meeting

“Safety” refers to making the retrospective an environment where all team members fully participate—sharing their personal insights and opinions and actively listening and reflecting on what others say. When a team works together, each person has a unique perspective and experience. Typically, individuals come into a retrospective with their own points of view, unaware that others may have had a completely different experience. We want everyone to feel comfortable participating so that we can see the project from all the different perspectives.

The stage is set for safety through an agenda that supports different participation styles, both introverted (need to think before speaking) and extroverted (need to speak in order to think). The team agrees on the norms for the meeting, and then safety is evaluated through an anonymous poll ranging from 5 (I will say anything) to 1 (I will say nothing and just agree with the manager). The more retrospectives the team performs, the more comfortable team members get and the shorter this safety segment is. It is important to validate the list of team norms and level of safety each time, as things may have changed.



The Gigantic Turnip

Several years ago, I stumbled upon a source of wisdom about retrospectives while trying to get my then five-year-old daughter, Ava, to sleep. I wanted to get her to bed early so that I could fine-tune my strategy for the retrospective I was facilitating the next morning. Having been warned that the team considered this retrospective to be an unnecessary, touchy-feely exercise, I was anxious. “Just one more book, Mommy,” Ava pleaded as she handed me *The Gigantic Turnip* and snuggled beside me.

Long ago, an old man and an old woman lived together in a crooked old cottage with a large, overgrown garden ... On a fine September morning, the old man ... said,

“It’s time for us to pull up that turnip.” ... The old man pulled and heaved and tugged and yanked, but the turnip would not move. The old man went to find the old woman. The old woman wrapped her arms round the old man’s waist ... Both of them pulled and heaved and tugged, but the turnip would not move. [3]

As I read the book aloud on autopilot, my thoughts wandered back to the retrospective.

... The old man, the old woman, the big brown cow, the two pot-bellied pigs, the three black cats, the four speckled hens, the five white geese, the six yellow canaries, and the hungry little mouse pulled and heaved and tugged and yanked. Pop! The gigantic turnip came flying out of the ground ...

By the time I got to the last page, I had a great idea for getting the team to fully engage in the retrospective. “Let’s read it again!” For the first time, this was *my* enthusiastic suggestion.

The next morning, I started the retrospective by re-enacting *The Gigantic Turnip* with the team playing out the various parts. The story starts with a farmer deciding it’s time to harvest a gigantic turnip. It then follows a typical cumulative structure, in which the farmer calls his wife, the cow, pigs, cats, hens, geese, and canaries one by one to join the chain and pull. Just before giving up, the wife catches a mouse scurrying past and asks her to join the chain. The mouse hesitates, thinking she is too small and insignificant to help with such a big problem. But when the mouse joins the chain and they all pull together, the turnip finally pops out.

A LESSON RELATED TO THE SAFETY PHASE OF A RETROSPECTIVE

During the debrief session, team members easily reached the conclusion that everyone’s contribution is crucial, no matter how small or insignificant it seems. They developed team norms to enable everyone to fully participate in the retrospective.

Pleased with how well this simple simulation achieved the desired results, I paid more attention during story time and found several more nuggets on Ava’s bookshelf.

Discover

What activities are we doing?	<ul style="list-style-type: none"> • Build timeline • Participate in focused simulation • Create radar graph
How long should this take?	30–50% of meeting
Who is working together?	Affinity groups
How is the room arranged?	A table and chairs for each group
We're done when we have ____.	A visible, big picture of facts, trends, issues, etc., for the time period being reviewed

At this stage, the team collaboratively creates a big picture of significant facts, trends, issues, etc. that occurred during the time period being reviewed. Many different activities can be undertaken to capture this shared understanding. It is effective to separate the team into affinity groups (people who share a perspective, perhaps because of their role or because of what they were working on) to brainstorm.



Commonly, a wall-sized timeline is built to provide a slow-motion replay of the project. The structure of the timeline can vary to help give focus and clarity to the data gathered (e.g., horizontal rows group topics together, colored cards highlight severity, etc.). Overlaying an anonymous “emotional seismograph” provides visibility into important human dimensions of the project.

Regardless of the activity used, the key is to emerge with a visible representation of the project that includes the perspective of all team members. The artifacts produced are the raw materials for the analysis phase that comes next.

Goldilocks and the Three Bears When I sense a team is starting to point fingers of blame as problems are identified during the discovery phase, I reference *Goldilocks and the Three Bears*. The bears come home from a walk to discover that a stranger has entered their house, eaten their food, broken some furniture, and slept in their beds. The predominant question upon discovering each problem is “Who did it?” disguised in accusatory statements like “Someone has eaten *my* porridge.” As the bears follow the trail of clues, they finally discover the culprit, Goldilocks, who immediately runs away. Unfortunately, by focusing on “who” instead of “why,” the bears remain vulnerable to the root problems—their home security is weak, their stove heats unevenly, and their furniture is fragile.

A LESSON RELATED TO THE DISCOVER PHASE OF A RETROSPECTIVE

Assigning blame is not the purpose of a retrospective, and trying to pin blame will lead you away from the collective learning that is needed. During the discovery phase, identify issues and problems in a factual manner so that you can later focus your energy on asking “why” in order to develop enduring solutions that target the problems’ root causes.

Analyze

What activities are we doing?	Facilitated discussion
How long should this take?	20–30% of meeting
Who is working together?	Whole team
How is the room arranged?	Chairs moved into a U shape so that everyone can see the “big picture” and each other
We’re done when we have ____.	A prioritized list of items team members feel they should continue, start, change, and stop

The big picture is mined for the key nuggets of learning. First, each individual reviews the big picture, looking for trends, key points, and surprises. Then, the entire group comes together to discuss insights and observations. Skilled facilitation is required to ensure the discussion progresses through all topics and that everyone has a chance to speak. The key points captured during each mini-discussion topic are things that team members feel they should continue, start, change, and stop.

The team then prioritizes the items in the lists created. Typically, the focus is on the start and change lists; however, don’t forget to include the continue list, because often the greatest gains can be achieved by taking up a notch the things already done. Tabulating individual votes is a quick and easy way to prioritize. Additional approaches to prioritization in-

clude elements of discussion, negotiation, and elimination to achieve a more balanced prioritization result.

Itsy Bitsy Spider I bring out the *Itsy Bitsy Spider* when a team has sidestepped disaster through sheer luck and then continues to blindly ignore the ongoing risk.

*The itsy bitsy spider went up the water spout.
Down came the rain, and washed the spider out.
Out came the sun, and dried up all the rain
And the itsy bitsy spider went up the spout again. [4]*

The spider is completely unaware of the danger associated with climbing a water spout during the rainy season. When it starts to rain, the spider makes a reasonable decision: Don’t panic, go with the flow, and wait out the storm until the sun comes out again. Because the spider put this experience on his “what went well” list, he later makes a very poor choice—rather than reflecting on the close call and addressing the ongoing risk, he resumes climbing the same spout. All his hard work will be wiped out yet again by the next rainfall, with no guarantee he’ll emerge unscathed.

A LESSON RELATED TO THE ANALYZE PHASE OF A RETROSPECTIVE

Be mindful about what you put in each of the lists collected during the retrospective (what to continue, start, change, and stop). Think critically about what you believe went well. Were you just lucky? Risk (re)assessment is a key part of a retrospective.

IKnow an Old Lady Who Swallowed a Fly From *I Know an Old Lady Who Swallowed a Fly*, we learn how small problems can quickly escalate into big problems. As soon as she swallowed the fly, the old woman knew she had a problem.

I know an old lady who swallowed a fly. I don’t know why she swallowed a fly. Perhaps she’ll die. I know an old lady who swallowed a spider that wriggled and jiggled and tickled inside her. She swallowed the spider to catch the fly. I don’t know why she swallowed the fly. Perhaps she’ll die. [5]

She hastily decided that swallowing a spider would solve the problem because, as we all know, spiders eat flies. In her haste, she doubled her problems instead of eliminating them. Without hesitation, she stuck with her default strategy.

... I know an old lady who swallowed a cow, don’t ask how she swallowed a cow. She swallowed the cow to catch the goat, she swallowed the goat to catch the dog, she swallowed the dog to catch the cat, she swallowed the cat to catch the bird, she swallowed the bird



to catch the spider that wriggled and jiggled and tickled inside her; she swallowed the spider to catch the fly, I don't know why she swallowed the fly. Perhaps she'll die. I know an old lady who swallowed a horse. She died of course!

By clinging to an ineffective solution, the old woman's simple case of indigestion became tragically fatal.

ANOTHER LESSON RELATED TO THE ANALYZE PHASE OF A RETROSPECTIVE

Choose your battles carefully. It is important to assign priority to the problems to be fixed. By applying an ineffective solution to a problem (small or large), the problem can be made much worse. If the situation is worsening, try something different.

Plan

What activities are we doing?	<ul style="list-style-type: none"> Root cause analysis Action planning
How long should this take?	15–20% of meeting
Who is working together?	Cross-functional groups
How is the room arranged?	A table and chairs for each group
We're done when we have	Concrete action plans (tasks, names, dates)

Initial action plans are developed for the top priority items identified in the previous stage. Cross-functional teams (representation from all project areas and roles) are formed to provide a broad perspective to the discussions. Begin the action plan by understanding the current state and the root cause of the problem. Envision the desired future state and identify the things that may hinder getting to this state. The final piece is a concrete action plan that has specific tasks with associated names and target dates.



The Three Little Pigs

The story of *The Three Little Pigs* is a great case study of a team successfully recognizing its problems and adapting its solutions accordingly.

A mother pig has three baby pigs. One day she says, "You've grown too big for my little house. It's time you had houses of your own ... Build your houses and never open the door to the Big Bad Wolf. He'll eat you." [6]

The first pig builds a house of straw, the second pig builds a house of sticks, and the third pig builds a house of bricks. As the wolf's threat escalates, the pigs move from the straw house to the stick house to, finally, the brick house. They start with a simple solution and move to more complex and costly solutions only as the risk increases. When I ask teams which type of house the pigs should build next time around, the answer is typically "brick" because only the brick house kept the pigs safe from the wolf. Unfortunately, this is not necessarily the best answer—what if there are no wolves next time around?

A LESSON RELATED TO THE PLAN PHASE OF A RETROSPECTIVE

I like to reflect on *The Three Little Pigs* at the start of the very first project retrospective and at the conclusion of the very last project retrospective to remind the team that con-

text is important in identifying a good solution for the current problem.

Close

What activities are we doing?	<ul style="list-style-type: none"> Review action plans Appreciations Hopes and wishes
How long should this take?	10% of meeting
Who is working together?	Whole team
How is the room arranged?	Arrange chairs in a circle so that everyone can face each other
We're done when we have	Made sure everyone knows the next steps

It's important to maintain the energy and synergy. A clear, firm closure to the retrospective will review the action plans and give people a chance to say final words of appreciation or hopes and wishes for the next time period. A quick retrospective on the retrospective itself helps to keep this team practice continually improving, as well.

Happily Ever After

Fairy tales are not just for entertainment; they can be used to teach children valuable life lessons, often through harsh themes and exposure to the darker side of life. Fortunately, the situations usually are resolved and the stories come to a satisfying close with *They all lived happily ever after*.

A LESSON RELATED TO THE CLOSE PHASE OF A RETROSPECTIVE

Retrospectives aim to expose difficult and uncomfortable project issues for the purpose of navigating the project story toward the "happily ever after" conclusion. In real life, this takes time and effort and repeated retrospectives. It is critical to the success of the next retrospective that the current one is carefully closed, so that everyone knows what concrete actions will be taken to make the next incremental improvement. {end}

Jennitta@theandregroup.com

Sticky
Notes

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

■ References

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



Join the StickyMinds.com Book Review Program!

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

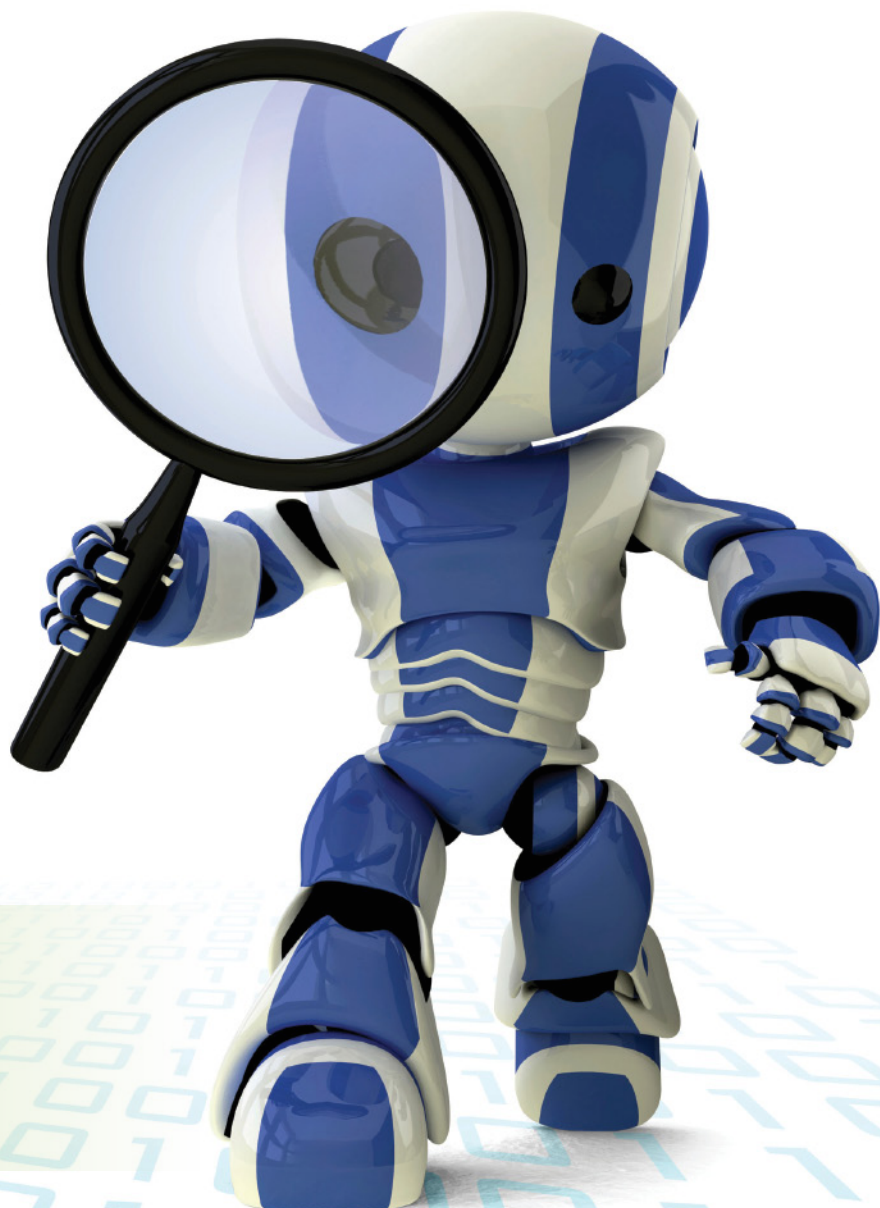
For an application or more information, contact
Cheryl M. Burke: cburke@sqe.com.



**STAR
WEST**

SOFTWARE TESTING

ANALYSIS & REVIEW



September 26–
October 1, 2010

San Diego, California
Hilton San Diego Bayfront

THE GREATEST SOFTWARE TESTING CONFERENCE ON EARTH

Choose from a full week of
learning, networking, and more

SUNDAY

Software Tester Certification—Foundation Level
Training (3 days), Using *Visual Studio® 2010 Ultimate*
to Improve Software Quality (3 days)

MONDAY – TUESDAY

30 In-depth tutorials in half- and full-day formats
Multi-day Training Classes continue

WEDNESDAY – THURSDAY

5 Keynotes, 41 Concurrent Sessions, the EXPO, Networking Events,
Receptions, Bonus Sessions, and more

FRIDAY

Testing & Quality Leadership Summit



www.sqe.com/starwest

**REGISTER BY JULY 30, 2010
AND SAVE UP TO \$400.
GROUPS OF 2 SAVE EVEN MORE!**

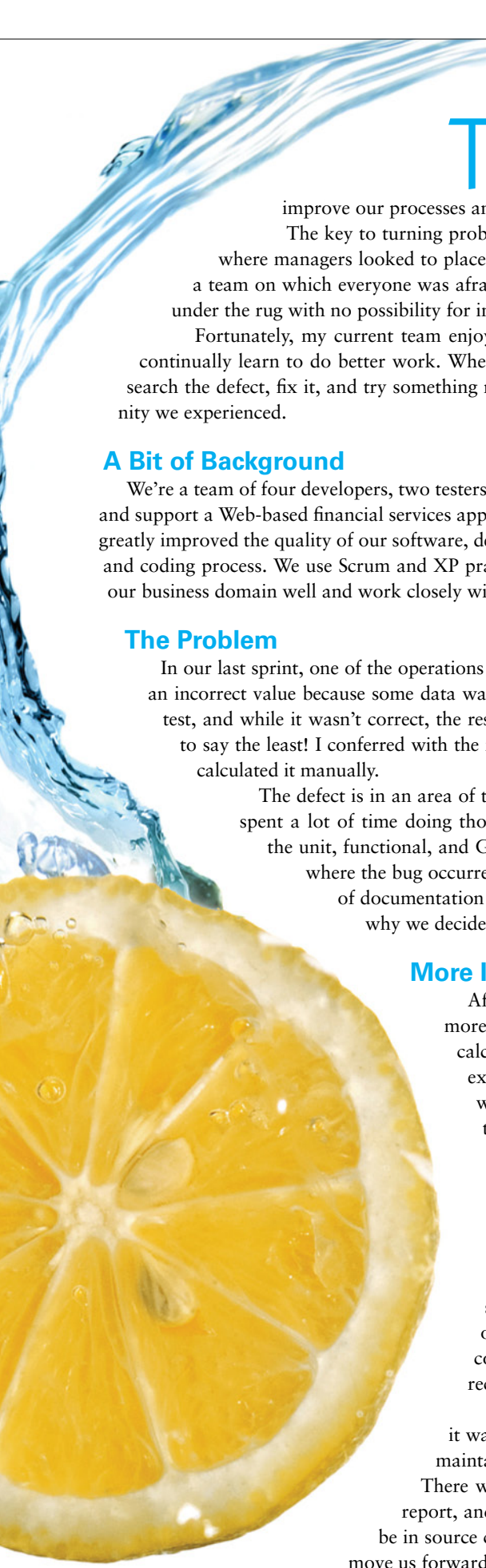


A high-speed photograph of a water splash, with numerous droplets and a main plume of water rising from the bottom right. A slice of a yellow lemon is partially visible in the bottom right corner, partially submerged in the water.

MAKING LEMONADE

Leveraging a Learning Culture

by Lisa Crispin



There's one thing I know for sure: We should never stop learning. Even if our team gets close to zero defects in production, ugly bugs can still escape. Mistakes can happen. However, if we can learn from our mistakes, we can improve our processes and practices and avoid making similar mistakes in the future.

The key to turning problems into improvements is a learning culture. In the past, I've worked on teams where managers looked to place blame for mistakes and punished team members for problems. That resulted in a team on which everyone was afraid to raise issues because the team might get into trouble. Problems were swept under the rug with no possibility for improvement.

Fortunately, my current team enjoys an enlightened management that gives us time to experiment, research, and continually learn to do better work. When a defect is found in production, we don't waste time pointing fingers. We research the defect, fix it, and try something new to improve future development. What follows is a recent learning opportunity we experienced.

A Bit of Background

We're a team of four developers, two testers, a DBA, two system administrators, a manager, and a ScrumMaster. We develop and support a Web-based financial services application. Six years ago, we had a buggy legacy application. Over the years, we've greatly improved the quality of our software, delivering what our customers want with very few defects slipping past our testing and coding process. We use Scrum and XP practices, along with some techniques borrowed from lean and kanban. We know our business domain well and work closely with our stakeholders.

The Problem

In our last sprint, one of the operations managers discovered a high-severity bug. One of our customer reports displayed an incorrect value because some data was being left out of the calculation. I decided to investigate. I ran the same job in test, and while it wasn't correct, the result was different from in production, given the same inputs. This was puzzling, to say the least! I conferred with the manager, who provided me with the value that should be on the report. She had calculated it manually.

The defect is in an area of the application that we rewrote about five years ago. It is highly complex, so we spent a lot of time doing thorough testing. We have many automated regression tests for this subsystem at the unit, functional, and GUI levels. However, I was surprised to discover that the particular functionality where the bug occurred had no automated regression tests above the unit level. Although there is a lot of documentation about this subsystem on our team wiki, no one had made any notes there about why we decided not to automate tests for this part of the code. I was puzzled.

More Investigation

After discussing the issue, the programmer tasked with production support did more research. He discovered that the logic to decide what data to include in the calculation was not in the Java code but in a view in our Oracle database. That explained the lack of automated regression tests—it would be impossible to test without using the database and, using the database, it would be costly to write a test.

Worse, there were “old” and “new” versions of the view. A synonym in production with the same name as the old one pointed to the new one. However, the schema we were using in test had only the old version. This explained why the results were different in test.

First lesson learned: Make sure your test environment truly is a duplicate of production. The SQL queries used by our application are maintained by our source code control system, but views, stored procedures, and DDL components of our Oracle schemas are not. This bug showed just how much we need to incorporate database objects into source code control, and we are in the process of rectifying this.

None of us could recall when this new version of the view was created or what it was meant to fix. There was no record of the change in the spreadsheet of changes maintained by the DBA, and the DBA who wrote the view left the company in 2006. There were no explanatory comments in the SQL code, we couldn't find a related bug report, and we found nothing on the team wikis about it. This is why everything needs to be in source code control. But, beating ourselves up for not having done that wasn't going to move us forward.

The Bug Becomes a Story

Clearly this was going to take significant time to research, fix, and test. We turned this bug into a story for the next sprint. The programmer spent two days studying the SQL in the new view—a complicated set of queries with an outer join—to understand what it does. He eventually identified two issues. One was with the report: A required value was omitted from the total. The second was with the SQL query itself: It was joining with a table that had no relevance to the calculation and caused the query to miss some of the data.

The programmer quickly fixed the report problem. I verified that fix while I continued to try to understand the problem with the view. The programmer went over the SQL query with me and showed me under what circumstances it would fail. It took me several hours, but at last I was able to reliably reproduce the problem.

The programmer fixed the view, added a lot of comments to it, and had the DBA put it in the test schemas, and I set about verifying the fix. Since I had found cases in the test schemas to reproduce the problem, it wasn't hard to test the solution.

More Testing

The problem occurred only with an edge case. Clearly, we had not tested this view and surrounding code adequately when it was released or when the view had been changed (whenever that was!). I took time to do exploratory testing, researching different cases in the database and trying every one I could think of. Why didn't I do this when we originally developed the functionality? I'm guessing it was time pressure. We had a hard deadline, and our risk analysis at the time must have been incorrect. Either we thought it was improbable that edge cases would occur, or we thought the edge cases would have a low impact.

Obviously, we needed some automated regression tests for this functionality. The only way to do it was laboriously to set up data in a test schema with a FitNesse test and use a FitNesse fixture to select rows from the view. The first test took me an entire day to set up, though subsequent test cases went faster.

What Did We Learn?

We were lucky in this situation—the errors only occurred in edge cases, and very few customers would have been affected. Nevertheless, we found a fragile area in our application. What can we do about it now? How can we prevent that from happening in the future? The following are some lessons we took from this experience.

We should put into version control all our database views, stored procedures, and anything else that can change from one release to another. Tools are available for database version control, and it's possible to use test-driven development for the database. My teammates make the argument that this would require a big effort, and it's rare that we have a problem like this. We'll keep studying and discussing this idea.

Make sure all team members understand the database design, how different schemas and users are set up, and why. I

went six years without understanding the difference between the “test-dba” and the “test-app” users in our main test schema. I didn't even know enough to ask the question, so someone who did know should have thought to explain it and verify that our test server used the correct one.

Don't skip automating regression tests just because it's hard and expensive—find a way to solve the automation problems. It was expensive to write these new automated tests, and the tests will be expensive to maintain. However, not having the tests cost the company more. When this code fails to work properly in production, it upsets our customers, can result in money out of pocket for them, and certainly takes up time for our own customer support staff. We erred in our original risk analysis.

Question design decisions such as putting business logic in SQL instead of in the application code. Ask questions. Be tactful and don't assume a Quality Police attitude—but ask. Raise your concerns about testability and stability. Ask your team to think through how each approach would be tested to see what seems most robust in the long term. Experiment with different design ideas until the team finds one that works for everyone. Personally, I would never want the team to put business logic in a database view—especially if that view isn't controlled—but there may be a good reason to do so. Perhaps, if we had done version control of the database from the beginning, having logic in the view would have been fine. We just didn't take time to think it all the way through or evaluate how it was working after we released the feature.

It's essential to experiment. When you encounter functionality that seems hard to test, spike a solution to your test automation problem. Review your own risk analysis with your customers and with other developers. When the whole team gets engaged in solving testing problems, the problems are suddenly much easier to solve.

Lemonade

In the short term, we've fixed the bug in production, and the manager who reported it is happy. We've done some clean up; we no longer have “new” and “old” versions of this view. We understand the view now and have created documentation to retain this knowledge. We're motivated to research database version control and other ideas for avoiding similar issues. Our future design decisions will be smarter. We'll work to ensure we can automate adequate regression tests and make future changes safely and quickly.

Small improvements like this build over time and make our team better. Accept that mistakes will happen despite everyone's best intentions and maximize the benefits of problems. Create a learning culture that allows the development team to improve continuously. In the long run, everyone will enjoy the work more, and the business will reap the benefits of continuous improvement. **{end}**

lisa@lisacrispin.com

☺ We are excited...

...that Hansoft can handle both schedule driven projects and agile methods like Scrum. The possibility of combining these methods in the same project has really been useful. Hansoft makes our projects more efficient and helps us collaborate much better with partners, customers and suppliers."

General Manager, Space Systems, Swedish Space Corporation.



Download a free 2-user trial at www.hansoft.se

>> Hansoft is an integrated solution for agile and lean development as well as collaborative Gantt scheduling, real-time reporting, bug tracking / QA, workload coordination, portfolio and document management, used by the most demanding game development studios in more than 20 countries worldwide. Hansoft does not only make team members and managers more productive in their everyday work, it also increases organizational productivity by enabling more efficient production methods and practices. Reduce your project risks with Hansoft, control your success. <<




THE WEEKEND TESTERS

TEST, LEARN, CONTRIBUTE

BY PRADEEP SOUNDARARAJAN AND
PARIMALA SHANKARAI AH

ISTOCKPHOTO



I was reviewing a test case when Outlook announced, “You have mail.” Since mail is always more interesting than test case documents, my curiosity kicked in. It was from the human resources team and read, “Dear Parimala, our monthly movie screening at the office is scheduled for this afternoon. The 1936 Charlie Chaplin classic *Modern Times* will be shown.”

The movie begins with Chaplin as a factory worker. His job is to tighten every nut that comes down the assembly line conveyor belt. Chaplin cannot afford even to sigh; otherwise, he’ll miss a nut and his manager will punish him. It was fun to watch how Chaplin coped when he missed a few nuts for reasons such as a bee swirling over him, his manager interrupting him, and wanting to stretch his hands.

To increase productivity of the factory workers, management installs a machine to “help” Chaplin. The machine is an automated feeding and cleaning device that ensures that the staff take the minimum amount of time to eat lunch and get back to work. The way Chaplin responds to the automatic feeding machine is hilarious because of the bugs in the machine.

Later, as some of the scenes replayed in my mind, I wondered, “Were we laughing at Chaplin or at ourselves?” *Modern Times* was made seventy-four years ago. We thought the world had changed completely since then, but we were wrong. Consider the following:

- As testers, are we rated on the number of nuts (test cases) we tighten (write, execute, evaluate)?
- As testers, are we punished for missing a nut (defect)?
- As testers, are we asked to use tools to speed up our work—tools that actually slow down the overall process?
- As testers, do we find it ironic that some managers will hold a four-hour meeting about a defect that was missed, thereby preventing our finding more during that time?

In testing, we seem to be in a rut. It appears we are paid to be “checkers” but given the fancy title of “testers” to make things look better.

In Michael Bolton’s words, “Checks are machine decidable; tests require sapience.” [1] Testing is an exploratory process, and checking is mostly a non-exploratory and confirmatory process. Machines do a better job of checking compared to humans, but only humans can do testing.

The Idea of Testing on Weekends

The *weekend testers* are a group of people who wanted to do more testing and learn to do it better. After spending the work week as specialist checkers, we wanted to focus on testing. Having met other checkers who also wanted breaks from checking, we thought we should open this up to the world after our initial experiments provided satisfactory results. We wanted to test, but what to test? Microsoft Word? Gmail? Why not an open source product that also helps the community? Should we close doors to testing commercial software? What better time than weekends to do all of this? That’s how the idea of weekend testing was born.

Typical weekend testing includes registration, facilitation, a testing session, and a discussion session. Announcements about upcoming sessions are published on weekendtesting.com, Twitter, testing forums, and testing groups. Testers assemble on a Skype group chat at a specified time. The facilitator for that session provides the product download details and mission to test and sets a time limit of one hour. The facilitator is available for queries during that hour, just in case testers get stuck. The facilitator may have set some traps without the testers’ knowledge. Once the testers begin testing, the facilitator reminds them to stick to the mission and avoid traps and diversions. At the end of the hour, testers stop testing and participate in the discussion session for the next hour. This is a debriefing session in which testers share their experiences, challenges, bugs found, traps that they failed to clear, and many other items. This time also includes discussing the product, mission, test strategies, tools used, etc. Many times, testers learn from other testers about new test ideas, tools, and strategies.

Test, Learn, Contribute

In a work environment where we are watched very closely—and where making a mistake can affect our performance reviews or even cost us our jobs—most testers are not interested in exploring new approaches to testing. In many cases, someone has labeled a technique as a “best practice,” and we follow it—no matter how long the idea has been around, how ineffective it is, or how expensive it is for the client.

Security Analysis

Do you have Path Insensitive Insecurity?



McCabe IQ

Vulnerability Analysis May Be the Answer

Path Insensitive Insecurity, *noun*,
[path\ in-sen(t)-s(a-)tiv\ in-si -kyur-a-tē]:
a software security disease that occurs when
programmers, designers, and software security
analysts are not cognizant of paths within their
source code or design that may lead to or be part
of a known or unknown exploitable software
vulnerability.

McCabe IQ helps to eliminate path insensitive software insecurities. The key to our uniqueness and the key to finding vulnerabilities is paths.

When designing functions and protocols, you should provide as few run-time options as possible to keep the amount of code exposed to attackers to a minimum. A lot of security flaws turn up in unused options and rarely-executed code. Commonly used code paths are routinely tested by end users, while obscure functions receive little attention.

Security breaches are often a result of interactions in software that, on the surface, appear innocent. Attackers can disrupt a system or defeat its security goals by exercising sequences of interdependent decisions to produce unforeseen, and possibly disastrous, consequences and unexpected results.

Structural Security Analysis with McCabe IQ can help uncover serious security flaws in code by analyzing control flow paths and subtree structures and verifying control flow integrity.

As part of a secure, trustworthy software development process, McCabe IQ helps identify and exercise paths through the code to ensure that program behavior is correct and expected. Traditional techniques for line and branch coverage leave too many security gaps. If security is important for your systems, you need a comprehensive validation approach that includes cyclomatic complexity and basis path analysis to scrutinize risky code structures using data and control flows. **You need McCabe IQ.**

Download "**Path Insensitive Insecurity**" at security.mccabe.com/bettersoftware, or call 800-638-6316 to schedule a live demo.



McCabe
SOFTWARE

Weekend testing offers individuals freedom and allows them to try new things and to make and learn from mistakes. Weekend testing also helps teach the responsibility associated with freedom. Our experience is that the value testers can contribute when provided with freedom is much more than what they do under strict supervision. Testers learn about their own skills and set out to do better the next time. They also appreciate and try to copy others' skills, ideas, and approaches.

From experience, the weekend testers have learned to:

Fail faster and safer: All of us want to do things that make us look good to the management or the client. It is human nature to do things based on what we are evaluated on. Testers participating in weekend testing are not rated on anything; they make mistakes that they wouldn't otherwise. This is a safe place to experiment with ideas and take them back to work.

Appreciate diversity: At work, hiring often happens without valuing diversity. As an example, everyone must be proficient with Quick Test Professional otherwise, there is no place on the team. Weekend testers discover the value of having different opinions. We hope this will improve how test managers think about testing.

Uncover hidden talent: Sometimes, an idea that we thought was stupid actually gets appreciation from our peers. This surprises us, and we start discovering new strengths in ourselves. What one tester does often surprises another, and when they communicate about it during the retrospective, it helps a lot.

Become aware of traps: Quite often, we fall into traps and are unable to recover from them. During retrospection, when one tester describes her experience of falling into a trap and how she recovered, it becomes educational for the rest. Sometimes, facilitators intentionally set traps to help others grow.

Vary experience: In our opinion, the idea of a Web-application-only tester, telecom-only tester, etc. is bad. There is often knowledge from testing other products that we can carry over to testing the product we are currently testing. In weekend testing, we choose to test a variety of products from various business, functional, and technology segments. This has proven to be a valuable learning experience for everyone.

Contribute to open source: In 2004, SourceForge called for testers to work for them and no one was interested. Today, through weekend testing, we service a number of open source projects at no charge. Some of the project owners have approached us directly, too, which equates to value for others for the time we spend learning. Programmers working on three different open source projects—WireMaster, FreeMind, and TuxPaint—applauded our passion for testing and the number of bugs we have found in their products. They paid us back by promising to fix some bugs, if not all of them.

Build unity in our field: Separated by schools, certifications, regions, onshore-offshore controversies, technologies, domains, and interests, we are tired of not being united. Here at weekend testing, irrespective of whether you are ISTQB certified or CSTE certified or even self-certified, it's all the same as long as you test, learn, and contribute.

Some people ask us if weekend testing could be the beginning of the renaissance of the global testing community. We will leave it to them to decide while we continue to enjoy testing.

Want to start a chapter? An email to chapter@weekendtesting.com will get the ball rolling or you can register at weekendtesting.com to participate in upcoming weekend testing sessions. {end}

parimala.shankaraiah@gmail.com

pradeep.srajan@gmail.com

Sticky Notes

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

- References
- Who we are



Extend Rally® and Bridge the gap between Development and IT Operations

Developers shouldn't have to deal with the complexities Agile and SOA have placed on your release plan. With StreamStep they won't.

StreamStep extends Rally by bridging the gap between Development and IT Operations.

View our quick video demo and you'll see how StreamStep fits right into your existing toolset. You'll stop the grind of troubleshooting release events, accelerate deployment, reduce release errors and risk.



View a quick video demo at
www.streamstep.com



stream|step

DRIVE

**technology with strategy
and have the bottom line
come along for the ride.**

The successful company doesn't react, it predicts. It doesn't wait for opportunity, it creates it. HP Enterprise Business can drive your company forward by infusing your infrastructure with business intuition and deep human intelligence, so when your marketplace moves, you've already leapt ahead.

Outcomes that matter.

Join us at HP Software Universe 2010 and learn how your business can get to where it needs to go. Visit hpsoftwareuniverse2010.com for more information.



Load and Performance Testing Services

BOSTON, MA—uTest has announced the launch of its load testing services, offering live load, simulated load, and hybrid load testing. These services help uTest customers ensure their Web applications are optimized for peak performance under heavy, real-world loads.

Load testing services include:

- **Live Load:** A team of live testers from around the globe can test an application simultaneously, enabling customers to see how their Web app performs under truly real-world usage conditions.
- **Simulated Load:** Requiring no live testers, simulated load testing provides customers with a complete analysis of a Web app's performance under peak synthetic load.
- **Hybrid Load:** Combining live testers with simulated load tools, uTest's hybrid load testing enables customers to perform functional testing while their Web application is under heavy synthetic load.

For more information, visit www.utest.com.

C++test

MONROVIA, CA—Parasoft Corporation has unveiled the release of Parasoft C++test. In this new release, Parasoft significantly expands the scope of C++test with the introduction of integrated runtime memory analysis that is suitable for both enterprise and embedded development.

New features:

- Runtime memory analysis enables teams to automatically identify during execution serious runtime defects—such as memory leaks, null pointers, uninitialized memory, and buffer overflows. It operates at both the unit level and application level. Since the instrumentation used for this analysis is lightweight, it can be run on the target board, simulator, or host for embedded testing.
- Parasoft Concerto Task Assistant brings change-based testing and automated task management directly to the developers' familiar work environment. Parasoft Concerto interacts seamlessly with Visual Studio and Eclipse-based IDEs to distribute, manage, and monitor each team member's assigned tasks. Moreover, the correlation of requirements, tasks, test cases, and artifacts not only establishes a flawless audit trail but also provides the base correlations for change impact analysis.
- Extended environment and compiler support for ARM, Keil, IAR, Texas Instruments, QNX, AIX, and Eclipse 3.5 enables more developers to use Parasoft C++test as a seamless part of their development process.

To learn more, visit www.parasoft.com/c_cpp_testing.

TestRail 1.2

GERMANY—Gurock Software has announced TestRail 1.2, a new version of its Web-based test management tool. The new version comes with various improvements, such as the option to invite new users to TestRail and support for PHP 5.3-based environments.

Check out a complete list of new features, improvements, and bug fixes, as well as get more information on how to install this update, in the Gurock support forum.

For more information, email: contact@gurock.com.

Copyright © 2010, Varad Corporation. All rights reserved.

\$0

TECH. SUPPORT



WE HAVE THE BEST NERDS

**TO ANSWER ALL YOUR
SUPPORT QUESTIONS.
OH, YEAH. FOR FREE!**

- ✓ **BUG TRACKING**
- ✓ **REQUIREMENTS**
- ✓ **TEST CASES**
- ✓ **TASK TRACKING**
- ✓ **FREE TECH. SUPPORT**
included with every BugHost subscription

**THE RIGHT TOOL.
THE RIGHT PRICE.
THE ORIGINAL**

 **BugHost™**
www.BugHost.com

BugHost, the BugHost logo and "Seymour" the bug are trademarks of Varad Corporation.

FAQ

expert answers to
frequently asked
questions

by Robert Sabourin
rsabourin@amibug.com

How do you know you're finished?

This question is fundamental to all the software engineering I've ever done. It's the question I'm asked by my clients as I organize projects. It's the question I'm asked by my team members to focus their testing. It's the question I'm asked by my students to help apply lessons to their testing.

The question challenges notions of quality by forcing the issue—What is good enough?

If products deliver value to stakeholders, then it is important to know what value is delivered to whom. *I know I am finished when the value I deliver matches the needs of project stakeholders.*

The question challenges notions of completeness—Did we test enough?

If I did not test enough, I may have missed important issues. Did I explore each testing objective reasonably well? *I know I am finished when I have done the tests I set out to do and explored to the depth I intended while adapting to the knowledge gained as I learned about the product.*

The question challenges notions of effort—Should we invest more effort in testing?

I often think of testing like insurance. My willingness to invest in insurance depends on my risk tolerance and potential payout. The payout from testing is knowledge. I want to make sure I spread my budget reasonably across the risks. Did I rebalance my testing portfolio as I gained product knowledge? *I know I am finished when the budget is reasonably spread across risks based on knowledge of technical risk and business importance.*

The question challenges notions of time—Is it time to move on?

There may be a point in time at which the market does not need the product I am testing. *I know I am finished when any further delay in the project would dramatically diminish the value of the product no matter what works and what fails.*

The question challenges notions of focus—Did we test the right things?

Verification helps us understand if we tested the thing right. Validation helps us understand if we test the right thing. *I know I am finished testing when I am testing the right things well.*

The question challenges notions of skills—Did we apply the best testing in the best way?

Domain and testing skills, knowledge, and experience are critical factors. *I know I am finished when the best testers applied their skills to the right problem with the support of expert domain specialists to assess correctness.*

The question challenges notions of decision making—How do we decide we're ready?

In testing, it is important to know how we are going to make decisions before we start testing. *I know I am finished when I focus testing and fix bugs with stakeholder-supported decision making.*

I wrote a children's book called *I am a Bug*, which offers this simple answer: *"I know I am finished when the bugs that are left are the bugs that we can live with at least for now!"*

Believe the Territory

Change is inevitable. Don't rely on documentation alone—be alert for signs you need to change course.

by **Markus Gaertner** | mgaertne@googlemail.com

The Swedish Army has the following dictum [1]:

When the map and the territory don't agree, always believe the territory.

This dictum applies to software development in a number of ways.

Requirements

When the requirements document and the requirements don't agree, always believe the requirements.

On projects, the most fragile document is the requirements document. This occurs for several reasons. First, your customers may not completely know what they want. This is normal, since they are trying to create a mental picture of their future. While the customers might have a mental picture of what they need, transmitting it to others can be problematic. Mental models are translated into written and verbal language, transmitted over noisy communication channels where relevant information may be lost, interpreted by the receiving side of those channels, and, finally, incorporated into some software. And then you wonder why you did not meet your customer's requirements.

The second problem with requirements is that they change over time. For competitive advantage, customers must be able to react to their particular business and markets. If they can't react, they will go out of business. Therefore, if you cannot adapt to the changing requirements of your customer, you, too, may go out of business. Since it's nearly impossible to think about all the changes that may come up during projects, you should leave the code and the tests as flexible as possible.

Third, requirements change not only for the sake of your customers' competitive advantage but, over time, the perception of "high quality" software may change. A new operating system may launch with a new user interface look and feel, and your customer may want to adopt it. Human interaction models change and while some of these changes may be predictable, many others are not. Some changes may creep in slowly; others, like changes in regulations and law, may occur month to month.

If the document that holds an interpreted snapshot of the

requirements at some point in time is not able to cope with these changes as they occur, we had better believe the requirements rather than the document.

Test Ideas

When the test plan and the tests don't agree, always believe the tests.

There can be a big difference between the test plan document and the ongoing activities on a software test project. The test plan document is an output of the planning activities surrounding testing. Therefore, it lists some testing activities that were thought to be necessary at some previous point in time. This might be early in the project, when little knowledge about

the product to be built existed. Therefore, when we run into a situation where the test plan lists an activity different from what the testers should be doing right now, reflect on whether the test plan defined a different activity based on lack of feedback on the actual progress, experience gained, or whether the tester is going off on a personal tangent.

If the document was written with some particular situation in mind and that situation unfolds in a different manner, testers need the right to change their course of action. When testing is not adapting to the context in which it is executed, we may miss critical bugs and information about the software that should be shared with our stakeholders.

Projects

When the project plan and the progress don't agree, always believe the progress.

In chapter ten of *Quality Software Management: Congruent Action* [2], Jerry Weinberg describes the Addiction Cycle. When the actual progress made falls behind the planned progress, a project manager is put under pressure to catch up. Giving in to the pressure, the project manager deliberately chooses to skip the review process and the test activities so that the product can be delivered faster. The short-term effect of his decision is some pressure relief for him. Of course, in the long term, the problems and bugs that were not addressed during the skipped practices are revealed and put the project under even greater pressure—calling for more shortcuts and skipping other prac-

“... if you cannot adapt to the changing requirements of your customer, you, too, may go out of business.”

tices. The only way out of the Addiction Cycle is to believe the progress being made and take necessary adaptations to the course, renegotiating the plan as needed.

Processes

When the process description and the process don't agree, always believe the process.

According to Alistair Cockburn, the process description defines which roles your team will fulfill [3] and explains how you hand off work products to another team member. But, processes seldom fit their written descriptions. That's why a temporary snapshot of the processes in the form of a process description is not likely to reflect the actual working habits in all projects everywhere in your company. Process descriptions may help new employees learn how a team works, but as the team gets through its norming and storming phase [4], these descriptions are likely to become out of date. Team members get into a habit of working together and sometimes decide to do things differently depending on the day's circumstances. Therefore, when the process description does not reflect the actual process followed, it's not necessarily bad.

Instead of constraining highly effective software teams by process descriptions that are not followed for long, team

members must be able to adapt their process to their particular needs. This involves regular reflection over the course of the project and small adaptations to the way work actually gets done. If a process description does not give the opportunity for a team to adapt on the actual project, team members will feel they are on a death march and, in the end, may even boycott the project.

Investigate

So, what should we do with all this documentation? My advice is to document everything that has value to your situation. But, when looking something up in a document, be suspicious; be very suspicious. Like a private investigator, dig into the available documentation and let it guide you, rather than making you inattentive. Keep your eyes and ears open for any changes that may have occurred after the document was written or reviewed. Compare the document with what is happening and act accordingly. Last but not least, make sure you update the document when a serious difference jumps out at you. **{end}**

Sticky Notes

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

■ References

index to advertisers

ADP East 2010	www.sqe.com/adpeast	Inside Back Cover
AutomatedQA	www.automatedqa.com	5
Avnet	www.avnet.com	14
BugHost	www.BugHost.com	33
Hansoft	www.hansoft.com	17
Hansoft	www.hansoft.com	27
Hewlett-Packard	www.hp.com/go/agile	Back Cover
Hewlett-Packard	www.hpsoftwareuniverse2010.com	32
Jama Software	www.jamasoftware.com	13
McCabe	security.mccabe.com	30
Microsoft	www.microsoft.com	6
Rally Software	www.rallydev.com/bsm	Inside Front Cover
Ranorex	www.ranorex.com	9
Seapine	www.seapine.com	1
SQE Training—Certification	www.sqetraining.com/certification	16
SQE Training—Agile	www.sqetraining.com/agile	10
STARWEST 2010	www.sqe.com/STARWEST	23
Stream Step	www.streamstep.com	31
TechExcel	www.techexcel.com	2
VersionOne	www.versionone.com	Opposite Page 10

Display Advertising
advertisingsales@sqe.com

All Other Inquiries
info@bettersoftware.com

Better Software (USPS: 019-578, ISSN: 1553-1929) is published six 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 or call 800.450.7854. Back issues may be purchased for \$15 per issue (plus shipping). Volume discounts available. Entire contents © 2010 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.

NOVEMBER 14-19, 2010

ORLANDO, FLORIDA

AGILE
DEVELOPMENT
PRACTICES
CONFERENCE

East

Conference Sponsor:



www.sqe.com/adpeast

Choose from a full week
of learning:

SUNDAY

Multi-day Training Classes begin

MONDAY-TUESDAY

In-depth Tutorials In Half- and Full-day formats

WEDNESDAY-THURSDAY

Keynotes, Concurrent Sessions, EXPO,
Receptions, Bonus Sessions, and more

FRIDAY

Agile Leadership Summit

REGISTER BY SEPTEMBER 17, 2010 AND

SAVE UP TO \$400

GROUPS OF 2 SAVE EVEN MORE!



100% OF 2009 ATTENDEES RECOMMEND THE AGILE DEVELOPMENT PRACTICES CONFERENCE

HARNES

the power of Agile.

HP Application Lifecycle Management for Agile initiatives

Looking to improve the visibility and consistency of your Agile initiatives?

The HP Application Lifecycle Management solution goes beyond traditional development management. It's the Agile solution that delivers quality *and* value, speed *and* control, for the whole team *and* the whole lifecycle.

Outcomes that matter.

Get more information at
hp.com/go/agile

