Summer 2015 www.TechWell.com

BETTER SOFTWARE™

A TECHWELL™ PUBLICATION

TEST AUTOMATION
Apply patterns to mitigate quality issues

DOING DEVOPS
Achieve success by ignoring the hype

INCORPORATING USER EXPERIENCE INTO EARLY AGILE CYCLES

BORN to TEST SOFTWARE

STAR WEST

SEPTEMBER 27–OCTOBER 2, 2015

ANAHEIM, CA | DISNEYLAND HOTEL | #STARWEST

CONFERENCE ROAD MAP

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

SUNDAY Multi-day Training Classes begin

MONDAY–TUESDAY In-depth half- and full-day Tutorials

WEDNESDAY–THURSDAY Keynotes, Concurrent Sessions, the Expo, Networking Events, and more

FRIDAY Testing & Quality Leadership Summit and Workshop on Regulated Software Testing (WREST)

REGISTER BY
JULY 31, 2015
AND SAVE UP TO $400

GROUPS OF 3+ SAVE EVEN MORE

STARWEST.TECHWELL.COM

FULL THROTTLE KEYNOTES

STAR WEST

Things That Really Matter in Testing—Today and Tomorrow

Bj Rollison, Testing Mentor

Testing the Internet of Everything

Paul Gerrard, Gerrard Consulting Ltd.

Lightning Strikes the Keynotes

Lee Copeland, Software Quality Engineering

I Don't Want to Talk about Bugs: Let's Change the Conversation

Janet Gregory, DragonFire, Inc.

The Survival Guide for Testers and Test Managers

Bart Knaack, Professional Testing

JUST A FEW OF OUR IN-DEPTH HALF- AND FULL-DAY TUTORIALS

Critical Thinking for Software Testers
Michael Bolton, DevelopSense

Selenium Test Automation: From the Ground Up
Dave Haeffner, The Selenium Guidebook

Testing under Pressure: A Case for Session-Based Test Management
Jon Bach, eBay, Inc.

Six Essential Skills for Modern Testers
Bart Knaack, Professional Testing

Plan, Architect, and Implement Test Automation within the Lifecycle
Mike Sowers, Software Quality Engineering

Test Design for Better Test Automation
Hans Buwalda, LogiGear

SEPT. 30–OCT. 1

THE EXPO

Visit Top Industry Providers Offering the Latest in Testing Solutions

TOOLS
SERVICES
TECHNIQUES
DEMOS

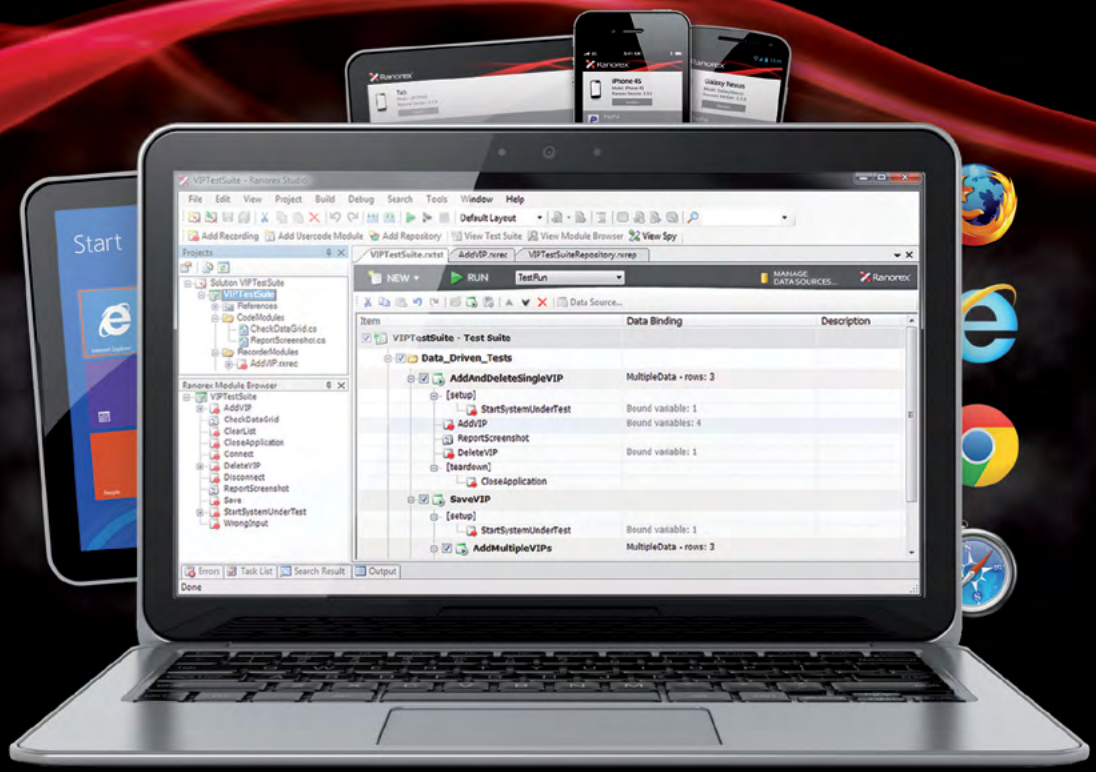
WHO SHOULD ATTEND

Software and test managers, QA managers and analysts, test practitioners and engineers, IT directors, CTOs, development managers, developers, and all managers and professionals who are interested in people, processes and technologies to test and evaluate software intensive systems

TO REGISTER CALL 888.268.8770 | STARWEST.TECHWELL.COM

Ranorex®

Automated Testing of Desktop. Web. Mobile.



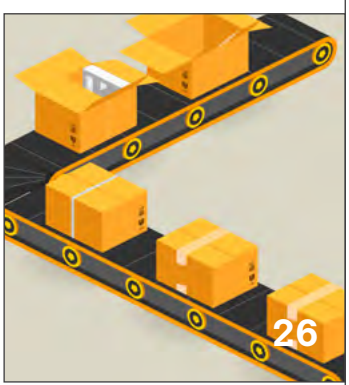
- Robust Automation
- Broad Acceptance
- Seamless Integration
- Quick ROI

Why Use Ranorex
www.ranorex.com/why

Award-winning test automation tools provide seamless testing of a wide range of desktop, web and mobile applications.
Android | iOS | HTML5 | IE | FF | Chrome | Safari | WPF | Flash/Flex | Silverlight | Qt | SAP | .NET | MFC | Delphi | 3rd Party Controls | Java

BETTER SOFTWARE™
A TECHWELL PUBLICATION
Volume 17, Issue 3 • SUMMER 2015

CONTENTS



features

columns

in every issue

MARK YOUR CALENDAR

SQE TRAINING

software tester certification

http://www.sqetraining.com/certification

Foundation-Level Certification

June 21–23, 2015
Vancouver, BC, Canada

August 25–27, 2015
San Jose, CA

August 31–September 2, 2015
Boston, MA

September 1–3, 2015
Columbus, OH

September 15–17, 2015
Atlanta, GA
Toronto, ON, Canada

Advanced-Level Certification

September 14–18, 2015
Toronto, ON, Canada

training weeks

Testing Training Week
http://www.sqetraining.com/trainingweek

September 21–25, 2015
Washington, DC

October 19–23, 2015
Dallas, TX

November 2–6, 2015
San Francisco, CA

conferences

STAR*CANADA*
http://starcanada.techwell.com
June 21–25, 2015
Vancouver, BC, Canada
Westin Bayshore

STAR*WEST*
http://starwest.techwell.com
September 27–October 2, 2015
Anaheim, CA
Disneyland Hotel

Agile Development Conference East
http://adceast.techwell.com
November 8–13, 2015
Orlando, FL
Hilton Orlando Lake Buena Vista

Better Software Conference East
http://bsceast.techwell.com
November 8–13, 2015
Orlando, FL
Hilton Orlando Lake Buena Vista

DevOps Conference East
http://devopseast.techwell.com
November 8–13, 2015
Orlando, FL
Hilton Orlando Lake Buena Vista

BETTER SOFTWARE
A TECHWELL PUBLICATION

Publisher
Software Quality Engineering Inc.

President/CEO
Wayne Middleton

Director of Publishing
Heather Shanholtzer

Editorial

Better Software Editor
Ken Whitaker

Online Editors
Josiah Renaudin
Beth Romanik

Production Coordinator
Donna Handforth

Design

Creative Director
Catherine J. Clinger

Advertising

Sales Consultants
Daryll Paiva
Kim Trott

Production Coordinator
Alex Dinney

Marketing

Marketing Manager
Cristy Bird

Marketing Assistant
Tessa Costa

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.
340 Corporate Way, Suite 300
Orange Park, FL 32073

You Could Use Some Better Approaches

You're probably wondering where *Better Software* magazine finds its authors. The best answer is—a whole lot of different ways. Some people email us article abstracts, some have written for us before, some are already presenting as speakers at TechWell conferences, and others I hunt down through social media sites, books they wrote, or even videos I've observed. (My boss calls it stalking; I justify it as simply makin' friends.)

TechWell provides valuable information to educate software professionals via the outlets that best work for you: conferences, workshops, eLearning, blogs, and *Better Software* magazine.

Rather than report the latest happenings with a specific operating system, programming language, or device, *Better Software*'s mission is to present information that will better your professional life by giving you the tools so that you and your team can deliver quality software technology that delights customers. This issue hits home with best practices that should improve your skills in a variety of subjects including agility, testing, and DevOps.

Ever wonder what makes some software products easy to use and others confusing? As more software technology runs on a variety of devices—from wearables to tablets to desktops—the importance of a great user experience (UX) has become a necessary part of any design. In his article, Chris Nodder, an expert in UX, should convince you of the importance of incorporating UX early in your agile project. The article from Dorothy Graham and Seretta Gamba presents an innovative approach to applying test automation patterns to identify and mitigate defects. Speaking of testing, Jennifer Gosden speaks the truth about the risk when low-severity problems aren't properly mitigated.

In keeping with our recent trend of showing how other job functions relate to software development, Joy Beatty and James Hulgan present a compelling perspective of the importance of business analysis to the success of your project. If you're still not really sure how DevOps relates to most every software development project, you're going to enjoy Chris Riley's article. And as if we don't already have enough technical obstacles to contend with, Johanna Rothman presents some good advice on how to best set expectations to what your manager really wants.

We truly value your feedback. Let us and our authors know what you think of the articles by leaving your comments. I sincerely hope you enjoy reading this issue as much I enjoyed working with these wonderful authors.

Ken Whitaker

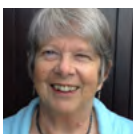
kwhitaker@sqe.com
Twitter: @Software_Maniac

Contributors

Joy Beatty is a VP at Seilevel, a business analysis consulting firm whose mission is to redefine the way software requirements are created. With fifteen years of experience, Joy evolves new business analysis methods and helps customers improve their requirements process. She worked with IIBA to develop the third version of their BABOK. She also coauthored *Visual Models for Software Requirements* and *Software Requirements, 3rd Edition*. Joy can be reached at www.seilevel.com and joy.beatty@seilevel.com.

Seretta Gamba has more than thirty years of experience in software development. As test manager at ISS Software GmbH, Seretta is in charge of improving the test automation process and developed a kind of keyword-driven testing and a framework to support it. She was invited to contribute to Dorothy Graham's book *Experiences of Test Automation*. You can contact Seretta at srttgmb@yahoo.com.

Jennifer Gosden is a software testing manager with Nationwide Insurance. She is responsible for application development testing activities in Nationwide's E&S division. Jennifer has more than twelve years of experience in software quality assurance and software testing practices. Reach Jennifer at jennifer.gosden@gmail.com and follow her on Twitter @JennGosden.

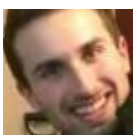


Dorothy Graham has been in software testing for forty years and has coauthored four books: *Software Inspection*, *Software Test Automation*, *Foundations of Software Testing*, and *Experiences of Test Automation*. Dot has been on the boards of conferences and publications in software testing, including program chair for EuroStar. She was a founding member of the ISEB Software Testing Board, helped develop the first ISTQB Foundation Syllabus, and was awarded the European Excellence Award in Software Testing in 1999. Dot can be reached at info@dorothygraham.co.uk.

James Hulgan is a business architect at Seilevel, where he is responsible for defining product management strategy and requirements architecture for projects with Fortune 500 companies. He also trains and mentors business analysts and product managers to define requirements and manage scope on their projects. James recently helped a top bank launch a commercial credit card program from the ground up, and is currently thinking deeply about customer master data management. James can be reached at james.hulgan@seilevel.com.

Claire Lohr has been an active professional in the computer field for thirty years, with the past twenty years emphasizing software process improvement and testing. She currently provides training (design, authoring, and instruction) and consulting services for a wide variety of both government and commercial clients. Claire was the chair of the Working Group for the revision of the IEEE Std 829-2008 Software and System Test Documentation. She can be reached at clohr@computer.org.

Chris Nodder is a user researcher and interaction designer who runs his own consulting company (www.nodder.com) helping agile teams learn how to build the systems their users truly need. He wrote *Evil by Design*, a book of persuasive design patterns, and has produced many Lynda.com video training classes. Learn more about the techniques in this article at his site, www.questionablemethods.com, and follow him on Twitter @uxgrump.



A long-time freelancer in the tech industry, Josiah Renaudin is now a web content producer and writer for TechWell, StickyMinds, and *Better Software* magazine. Previously, he wrote for popular video game journalism websites like GameSpot, IGN, and Paste Magazine, where he published reviews, interviews, and long-form features. Josiah has been immersed in games since he was young, but more than anything, he enjoys covering the tech industry at large. Contact Josiah at jrenaudin@sqe.com.

Chris Riley has spent twelve years helping organizations transition from traditional development practices to a modern set of culture, processes, and tooling. In addition to being a former Gigaom Research analyst, he is a regular industry author, speaker, and subject matter expert in areas of DevOps strategy, culture, and enterprise content management. He is interested in machine learning and the intersection of big data with information management. Follow Chris on twitter @hoardinginfo.

Johanna Rothman, known as the "Pragmatic Manager," provides frank advice for your tough problems. She helps leaders see problems, seize opportunities, and remove impediments. Johanna has just completed her latest book entitled *Predicting the Unpredictable: Pragmatic Approaches to Estimating Cost or Schedule* and has started *Agile and Lean Program Management: Scaling Collaboration Across the Organization*. Johanna writes columns for StickyMinds and ProjectManagement.com, writes two blogs on her jrothman.com website, and blogs on createadaptablelife.com. Contact Johanna at jr@jrothman.com.

Do You Give Your Manager What She Wants or What She Needs?

Management needs to know a software product's quality level and release date. Answering those questions doesn't have to be so difficult.

by Johanna Rothman | jr@jrothman.com

I bet most of you encounter a common agile project management problem: how to provide status information while your project undergoes constant change. Your managers want to know when you will finish the project, and that's a reasonable request. However, due to the inevitable adjustments of the team's backlog, product backlog, and product roadmap, providing that information is a real challenge.

It's also a challenge to even estimate. Should you provide a gross estimate of the maximum time? Should you consider estimating a minimum release date? Or do you derive a schedule with something in the middle of the range? It's not an easy decision.

As a business, management still needs to know when they can expect your team to release the product to generate revenue. That's really the question they are asking. In waterfall, revenue isn't usually recognized until the end of the project. In agile, there is a possibility of delivering product incrementally in order to recognize revenue while the project is still in flight, before the final release.

In fact, that might be the answer to your manager's question.

If you don't know, ask. That's where the transparent culture of agile smashes into the more traditional approach of management and finance. It's a culture clash and a potential huge disconnect when setting expectations. You might not be talking the same language as management.

Many managers are still stuck in "What will we get by date x?" and "Can I get the team to commit to that deliverable by that date?" The farther out those dates and the larger the deliverables, the less those questions make sense. The product owner and eagerly awaiting customers will want to adjust things. With lots of customer participation, change is going to happen. That flexibility is possible and normal on an agile project.

It doesn't make sense to tell your manager, "Don't ask those questions. They don't provide any value, and any answer I give you won't be correct anyway." Instead, consider asking these questions:

- Do you need to know the first release date so that we can recognize revenue?
- Do you need to know the first date we can release something externally that our customer will find valuable?
- Do you need to know when we will fix the problem support has with the reported user problem?
- Do you need to know when we will have this feature or features to attract specific customers?
- Do you need to know when we can capitalize the software development expenses?

You see the pattern here. These questions go to the heart of every agile project: to deliver working product frequently so you can inspect and adapt.

> "You can work with the product owner to rank those defects with the features and perform a gross estimate with confidence or multiple dates."

Your manager might need the answers to these questions. She might need other answers. How do you give your manager what she needs, as opposed to what she wants? And that's a different question. Managers need to understand their options with agile, and you can help them.

Regardless of the answers to these questions, your team still needs to provide working product at a regular cadence. You can work in iterations or flow—it doesn't matter as long as your project's cycle time between releases is short. You need to release something of value frequently. (Note: You can still have frequent internal releases and release externally when your customers are ready. When you have frequent internal releases, you have a business decision about when to release.)

This is easy with cloud-based or Software-as-a-Service (SaaS) solutions. It's more difficult if you deliver client-server, boxed software, or a hardware device with your product. In that case, you need to define your minimum viable product carefully. You have to decide if frequent product releases are possible or even desirable.

You might need to change the way you think about your product and your product development. Should you consider transitioning to a different kind of product?

Don't ignore your manager's questions or say, "We can't estimate." Your manager's questions can be the start of a conversation about how your organization approaches packaging and releasing your product and what it means to be agile.

Once you know what your manager needs, you can answer in these ways:

For the first release date: Once you know what a minimum viable product is, you can provide an estimate for a release date. You might need a gross estimate with percentage confidence or a three-date range. You can explain when you will update your estimate, based on your progress and the backlog. Your manager should be satisfied with that. This is the same answer for when you can start capitalizing your effort.

For the first external release: You and the product owner can together define the backlog for an external release. Now you do a gross estimate and percentage confidence or three-date range.

When fixing certain defects or targeting a release for specific customers: An estimate for release depends on where in the backlog those defects are. You can work with the product owner to rank those defects with the features and perform a gross estimate with confidence or multiple dates.

Maybe you need to consider how your customers configure your product. Maybe you need to transition from client-server to SaaS. Maybe you have a way to update your product in the field once you release or ship the hardware.

With agile, you can take advantage of your ability to update the product quickly. The question is, should you? The more you are able to create features with no technical debt and no defects, the easier this is. That requires substantial discipline in the team.

Your manager's questions deserve consideration. The "When will it be done?" and "How much will it cost?" and "What features will I see when?" questions are not trivial. The answer that no manager wants to hear is, "It depends."

You need to consider providing your manager with the answers she needs, not necessarily the answers she wants. That's tricky. It will take tact and diplomacy. Use your agility to supply answers your manager needs. **{end}**

Assurance is the certainty of a risk-free experience. With confidence.

In today's vast and unpredictable technology landscape, does your QA and Testing function give you the confidence of fail-safe systems and zero-risk performance? There exists a way: Tata Consultancy Services (TCS). With TCS' independent enterprise testing arm, Assurance Services Unit (ASU), you can provide the certainty of risk-free systems to your customers, with market-proven, world-class experience, expertise and guidance.

Visit **tcs.com/assurance** and you're certain to learn more.
Or write to us at: **global.assurance@tcs.com**

TATA

IT Services
Business Solutions
Consulting

Scan the code to know about TCS Assurance Services

TATA CONSULTANCY SERVICES
Experience certainty.

MOVE YOUR TESTING FORWARD

Maximize the impact of your training by building a custom week at one of our Testing Training Weeks. Choose from 16 specialized courses led by the industry's most respected professionals. The more training you take, the greater your savings.

COMBINE AND SAVE — TRAINING WEEK

2015 FALL SCHEDULE

TESTING TRAINING WEEKS

September 21–25, 2015
Washington, DC

October 19–23, 2015
Dallas, TX

November 2–6, 2015
San Francisco, CA

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
Software Tester Certification—Foundation Level			Mastering Test Design	
Security Testing for Test Professionals		Integrating Test with a DevOps Approach		DevOps Test Integration Workshop
Fundamentals of Agile Certification—ICAgile		Agile Tester Certification		Agile Test Automation—ICAgile
		Mobile Application Testing		Mobile Test Automation Workshop
Essential Test Planning and Management		Measurement & Metrics for Test Managers	Leadership for Test Managers	Test Improvement for Agile
Risk-Driven Software Testing		Performance Load and Stress Testing		

SQE TRAINING
A TECHWELL COMPANY

Project Management Institute R.E.P.

Green background Indicates courses pre-approved for Project Management Institute PDUs.

WITH TRAINING FROM SQE TRAINING

Instructor-led training in a city near you

Live, instructor-led classes via your computer

Self-paced learning, online

Instructor-led training at your location

WHO'S BEHIND THE TRAINING?

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

Why Choose SQE Training?

- Expert instructors with 15–30 years of real-world experience in the software industry
- The most relevant selection of specialized software training courses available anywhere
- Highly interactive exercises designed to keep you engaged and help you implement what you've learned immediately
- Small classroom workshop environment
- Over 20,000 students trained worldwide who provide constant valuable feedback on our courses

For information on our 60+ Public and 40+ Live Virtual Course Dates
visit www.sqetraining.com

SQE TRAINING
A TECHWELL COMPANY

Jeff Patton

Years in Industry: **20**
Email: **jeff@jpattonassociates.com**

Interviewed by: **Josiah Renaudin**
Email: **jrenaudin@sqe.com**

"You're engaged in the lean process and if you have a commercial product or a consumer product, you don't put it out in front of your whole audience, especially if it's half-baked. There's fast, and there's stupid."

"The idea behind lean as it's applied to products is, instead of increasing the speed that we build things, instead of increasing that throughput of working software, we're increasing the throughput of knowledge."

"'Nail it before you scale it.' We want to nail it or get it right with a small subset of our audience before we roll it out to everybody."

"Wherein traditional user experience design we'd spend time really understanding the problems we're solving, we'd spend time designing, and then we'd spend time validating our solutions—in lean user experience, we do a lot of that all at once."

"One of the core product-thinking concepts that I want people to understand is that you choose your customers. I know people, for instance, that still think iPhones are stupid or still don't like them."

"If we don't really think about user experience and get it right, our products fail in the market."

"One of the things that's fundamentally changed about user experience is that user experience work can effectively involve whole teams: developers, QA people, product managers, business people, business analysts. Everybody can be involved in doing this work with a good user experience person as a guide."

> "I think we're finally acknowledging that our ability to guess right, even with deep understanding of the users we're solving problems for, they're still our guesses. Our ability to guess right isn't so good."

STICKYMINDS.COM™

For the full interview, visit
https://well.tc/IWAE17-3

NEWSLETTERS FOR EVERY NEED!

Want the latest and greatest content delivered to your inbox every week? We have a newsletter for you!

- **AgileConnection To Go** covers all things agile.
- **CMCrossroads To Go** is a weekly look at featured configuration management content.
- **DevOps To Go** delivers new and relevent DevOps content from CMCrossroads.
- **StickyMinds To Go** sends you a weekly listing of all the new testing articles added to StickyMinds.
- And, last but not least, **TechWell To Go** features updates on the curated software development stories that appear each weekday at TechWell.com.

Visit StickyMinds.com, AgileConnection.com, CMCrossroads.com, or TechWell.com to sign up for our weekly newsletters.

Curious About DevOps? Start Here!

We can trace the beginnings of the DevOps movement back to a Belgian named Patrick Dubois. In 2007, Patrick lamented that the two worlds of development and operations seemed miles away from each other, and there were conflicts everywhere. He observed that development and operations teams tended to fall into different parts of a company's organizational structure (usually with different managers and competing corporate politics) and often worked at different geographic locations. Since then the DevOps movement has gained global momentum and has become a lightning rod for people who have something to say about how IT is—or should be—running.

The movement found traction online through social media and discussion boards. DevOps is clearly touching a nerve within the industry, likely because DevOps is from practitioners, by practitioners; it's not a product, specification, or job title. DevOps is an experience-based movement about cooperation and collaboration.

In response to the call for more information and resources about DevOps, TechWell is introducing the inaugural DevOps Conference West from June 7–12 at Caesars Palace in Las Vegas. DevOps Conference West will accompany the fifth annual collocated Agile Development & Better Software West conferences, the premier event for software professionals. This year's program is even more robust, bringing all aspects of the software development lifecycle to the forefront.

DevOps Conference West features industry practitioners passionate about the DevOps movement and focuses on topics like:

Why DevOps Changes Everything—Keynote presenter Jeffery Payne talks about what steps need to be taken to successfully achieve a DevOps process while avoiding the pitfalls. He'll also leave the audience with some take-home ideas about how to leverage DevOps to advance their careers.

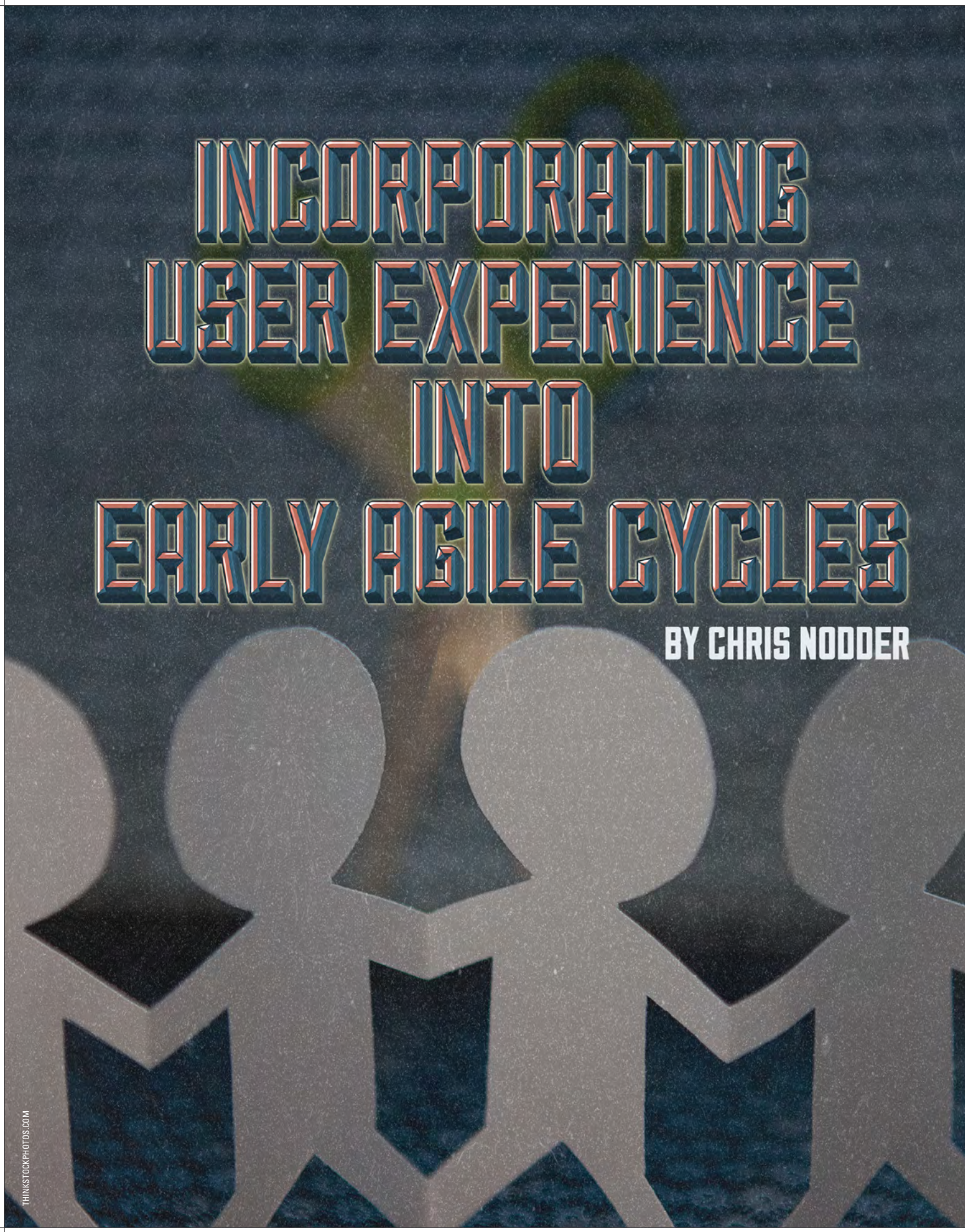
Continuous Delivery: Rapid and Reliable Releases with DevOps—Bob Aiello explains how to implement DevOps using industry standards and frameworks in both agile and non-agile environments, focusing on automated deployment frameworks that quickly deliver value to the business.

A DevOps Journey: Leading the Transformation at IBM—Dibbe Edwards describes the journey she went through leading the DevOps transformation at IBM. She will share her experiences, the best practices she discovered, what techniques she used, and how she recommends a software development team get started on the DevOps journey.

We hope your DevOps journey brings you straight to DevOps Conference West. See you in June!

INCORPORATING USER EXPERIENCE INTO EARLY AGILE CYCLES

BY CHRIS NODDER



THINKSTOCKPHOTOS.COM

Big design up front performed at the early stages of a project's lifecycle is the bane of agile teams, but no design up front is a recipe for disaster. Done right, a little bit of upfront user experience (UX) work can provide a plan for the whole release, improve the team's understanding of the project goals, provide useful tracking metrics, and significantly reduce rework.

User-centered design (UCD) is a technique for finding users' pain points—the things they most need your software to help them with—and then validating the solutions you build as early and often as possible. Frequent end-user interactions give you confidence that the designs you flesh out from your early UX work stick to design guidelines you created and end up truly delighting your customers.

Along the way, this focus on resolving user pain points during the build and validating phases of a project's lifecycle gives the team a common vocabulary to describe what's wrong with the current process, to prioritize the backlog, and to define "done."

Who Are Your Users?

UCD starts by identifying your users, who often have very different roles from your customer. Your customer pays you money to fix their problem, and the user is the person who works with the solution you provide.

Often, the first time true end-users are involved in a project is during user acceptance testing. UCD turns this on its head by ensuring that the whole development team observes different end-users working with their current processes, preferably in their typical work environment. The only equipment you need for this activity is a notepad, pen, and a roll of duct tape. The duct tape is for the development team's mouths. The primary instruction to the team members is "Shut up and watch."

Mapping the Experience

Even a couple of hours of observing six to eight users will yield enough high-priority pain points to keep the development team busy for months. [1] Figure 1 shows what can happen when you get the team to plot out their observations using sticky notes on a wall-sized experience map. [2]

This orders users' tasks across the top of the page and describes the process and issues for each subtask vertically. Team members get to dot-vote the issues into a rank-order list of pain points they feel most need attention. This ordered list becomes the focus for the next release.

Because each team member observes different users, the experience-mapping exercise is an opportunity to share observations and find common threads. The resulting pain points can easily be turned into user goals, and the team can agree on metrics to determine what successful completion of a goal might look like.

As an example, say that several developers observe users struggling to capture data from a phone conversation into a customer relationship management system. Because this is a user's priority task, the goal might be to make it easy to capture contact details and current query information. Metrics for success could include the maximum time this task should take, the acceptable error or rework rate for users, and identifying a way to measure the user's confidence that the data has been stored successfully. Now, whatever solution the team builds for this task, they have some metrics to use for usability testing in much the same way as they might for unit testing.

Build the test first, then build the product.

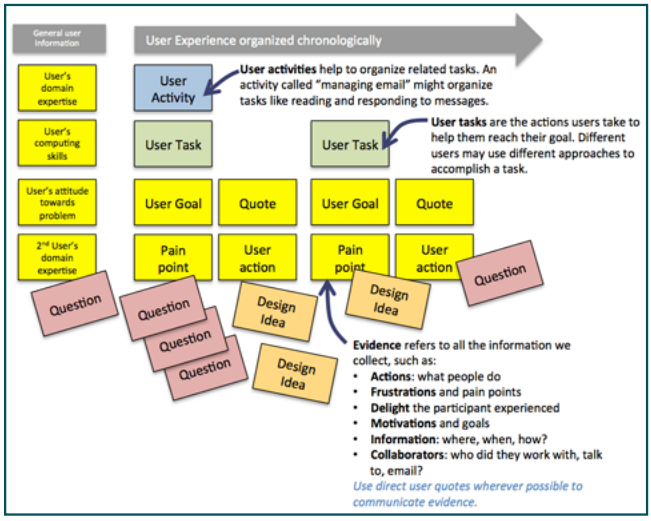


Figure 1: Building an experience map

Imaginary People for Real Focus

With user goals and metrics in place, it's time to start designing solutions. However, it's good to have a focus for these solutions. Everyone on the team has a different perspective of who the user is. It's useful to condense these diverse impressions into a couple of common, archetypal user descriptions. [3] Each persona might only be a thumbnail sketch on a 5x7 index card, but it's important that they are believable individuals with qualities similar to the people the team observed during their site visits, as shown in figure 2.

To make it more realistic, it's good to give each persona a name and some basic biographical information. Then list the context in which they'll use the product, their goals, and the implications of these goals. You'll also want information for

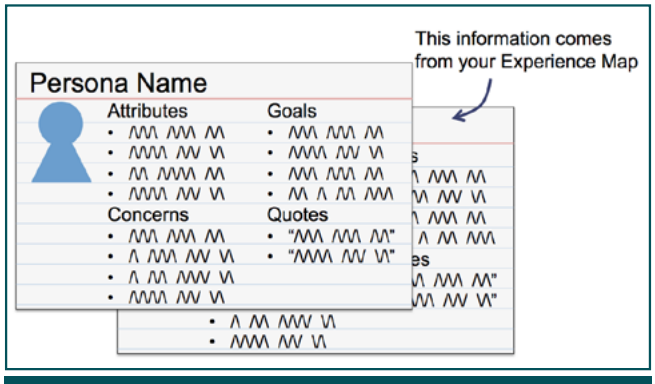


Figure 2: Creating personas

how the persona will want to interact with the product.

Now you can start writing scenarios for how each of your personas would interact with the system in an ideal world. [4]

These scenario descriptions should stay away from describing interface components and instead talk about interactions and outcomes. It only takes a couple of hours of the whole team working together to come up with a good, believable set of scenarios that covers the main tasks the team will be developing solutions for.

Getting the Entire Team Involved in Design

Once the team has some scenarios in place, it's time to get creative with some ideation time. You could just dive straight into writing stories around the scenarios and call it good, only who knows whether your stories would capture the most compelling design? Spending a bit of time at this point to broaden your perspective using an ideation technique such as design charrettes will allow you to explore multiple options and agree as a team on the most productive way forward. [5]

The design charrette process involves every team member working individually for ten to fifteen minutes to sketch a high-level interface design for one of the scenarios. After the time is up, everyone gets together to share and critique the designs. After hearing everyone's perspectives, each individual creates a second round of designs. This beats brainstorming because no one voice can dominate. Instead, everyone draws out his own solution but still benefits from the group's shared creativity. After the second round of design work, the team reconvenes to discuss the updated interface ideas.

Cheapest Prototypes Ever

At this point, you should have a good idea of what the user stories are going to be and what the interface might look like. You could just start coding, but code rework is expensive and you don't know for sure that your ideas are correct yet. Why not spend a day or two to validate your ideas with users? Paper prototypes—literally sticky notes stuck on 11x17 sheets of paper—are a great tool to get your first round of user validation. [6]

Reading through their scenarios is the best way to motivate teams to create paper prototypes. Each time the scenario mentions an interaction, it's time to create an interface component. Using the design charrette output for inspiration, draw each interface element on a sticky note with a marker, cut the sticky note to size (you soon learn to always start drawing in the area that has glue behind it), and stick the element on the sheet. You'll probably develop a color convention, such as buttons are always green, labels are always blue, and text entry fields are always yellow. Continue reading through each scenario until you have created interface elements for every part of the interaction.

Consider always using sticky notes. Because every element is independent, it's easy to quickly reorder the interface components. Stickies make it easy when changes impact earlier parts of the interface that have been already defined. Using a wireframing tool, teams can become reluctant to change their drawings even though they know the design needs updating. Not so with stickies.

Even if the resulting interface looks ugly, you're testing concepts, not the final implementation. The team shouldn't be interested in designing every last detail of the interface during prototyping. Instead, we want to know whether the general flow is in line with users' expectations, whether they understand the concepts we've introduced, and whether they believe the new design will eventually meet user needs.

Usability Testing to Validate Your Design

To answer those questions, you'll need to run a usability study with your paper prototype as the interface. [7] Recruiting just five representative users—preferably not the same ones you visited at the beginning of the process—will give you sufficient feedback at this point. Figure 3 shows how paper prototyping can evolve simply by listening to your users.

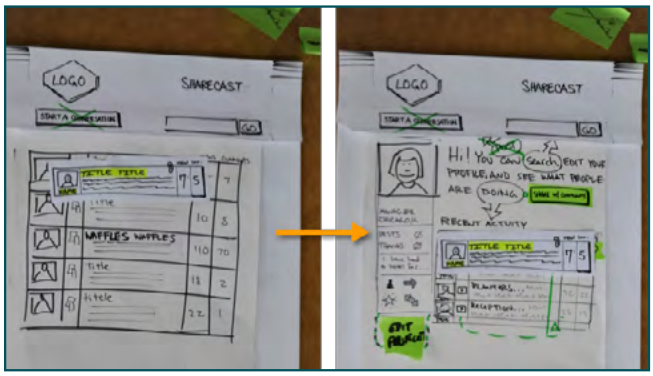


Figure 3: Creating personas

Usability study fidelity will increase later along with interface fidelity, but for now you're mostly interested in getting early verification that your design concepts are suitable.

Usability testing your paper prototype is your first opportunity to capture metrics. The tasks you ask users to perform with the paper prototype should be based on your scenarios, which in turn were based on the list of user goals you created. Each of those goals has associated metrics, so it should be pretty readily apparent which metrics apply to each user task. The paper prototype usability test should give us a good general indication of whether this new interface design and interaction style is likely to resolve a user's pain points. Users love working with paper prototypes because it's clear to them that the interface isn't done and that their comments and suggestions really count for something.

Just like in regular sprints, it's unlikely your first prototype design iteration will be perfect. There could well be areas where usability study participants lost the plot, the interface wasn't clear to them, or they wanted features or interactions that just weren't available. That's OK. Early usability studies like this are formative, designed to help the team work out where the problems are and to identify improvements.

It might be that there are clear solutions, in which case you can probably just make some changes and move forward to

the next step. On the other hand, it may be necessary to build more paper prototypes with a new design concept and do another round of usability testing before you continue. The thing is, paper prototyping is fast and cheap. Getting the general concepts right at this point will save untold weeks of rework later in the process.

Now Create Your Backlog

Once you are confident that your prototype concepts form a good basis for an interface design, it's time to create your product backlog. Jeff Patton's story mapping technique provides a great framework that will augment the initial experience map. [8] The experience map describes user processes as they exist today, and the story map will describe the future state interface and user behaviors. Story maps add an extra dimension to your backlog because they allow you to prioritize stories within interface areas and across functional areas at the same time. This makes it easy to see which stories the team needs to tackle first in order to get to a minimum viable product—or in our case, a minimum user-testable product.

To create a story map, I typically lay out each page of the paper prototype in the order users would experience it, then create stories to describe the capabilities that are present on each page of the paper prototype. In this way, there is a direct link from our initial experience map, through the user pain points and goals, to the scenarios, the paper prototype, and finally the story map. These artifacts give the team a common vocabulary and understanding. If you write your stories in the form "As a [user type] I want to [task] so that [goal]," it becomes very easy to insert a persona name, interface capability, and user goal to create each story.

Prioritizing a story map is easy because the paper prototype interface is available for reference. That way, it's clear which stories are essential for basic functionality and which are aimed more at adding delightful elements to the interaction.

I regularly facilitate weeklong sessions with teams, moving from user observation on Monday morning to a usability tested prototype on Thursday afternoon and backlog creation on Friday. If you want to try this for yourself, I suggest you use a two-week cycle so that you can familiarize yourself with the techniques as you progress.

The True Benefits of User-Centered Design

Building the basic functionality first means you get to usability test with actual alpha-ready code earlier in the process, and only then should the team carefully add additional features once you're sure you're on the right track.

Regular usability sessions are the key to reducing rework. Each session gives you more data to help you understand whether you're on track toward meeting users' needs.

If users don't understand your UI, it's time to revisit your basic design assumptions before you pile even more features on top of the core concepts. Because you'll be working with alpha code, usability sessions also tend to be a great way to report bugs.

To summarize, the benefits of a UCD approach are:

- ***Early user feedback:*** Identify the pain points
- ***Time and cost savings:*** Build the right thing the first time
- ***Faster delivery:*** Less rework, clearer communications
- ***Faster to market:*** Realize business benefit sooner
- ***Better focus:*** Clear goal-setting for team and management with the team striving to get to done
- ***Better measurement:*** Metrics for success are baked in

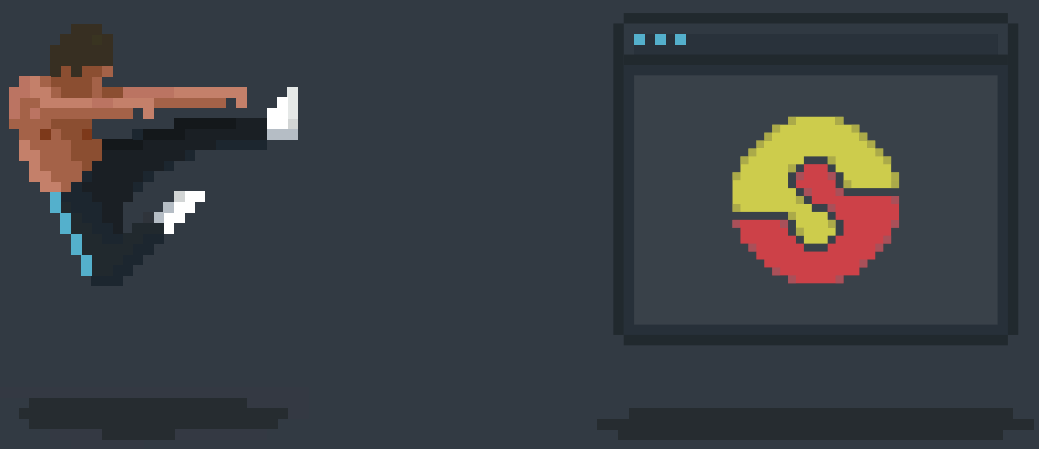
Bringing all necessary expertise together in one room rather than holding interminable meetings produces major time savings and cost benefits. Design decisions are based on quantifiable member data rather than personal opinion. And the team ends up with a common understanding to create the best user experience possible. **{end}**

chris@nodder.com

Sticky Notes

Click here to read more at StickyMinds.com.

- References



MARTIAL ARTS
HAS BRUCE LEE.

AUTOMATED TESTING
HAS SAUCE LABS.

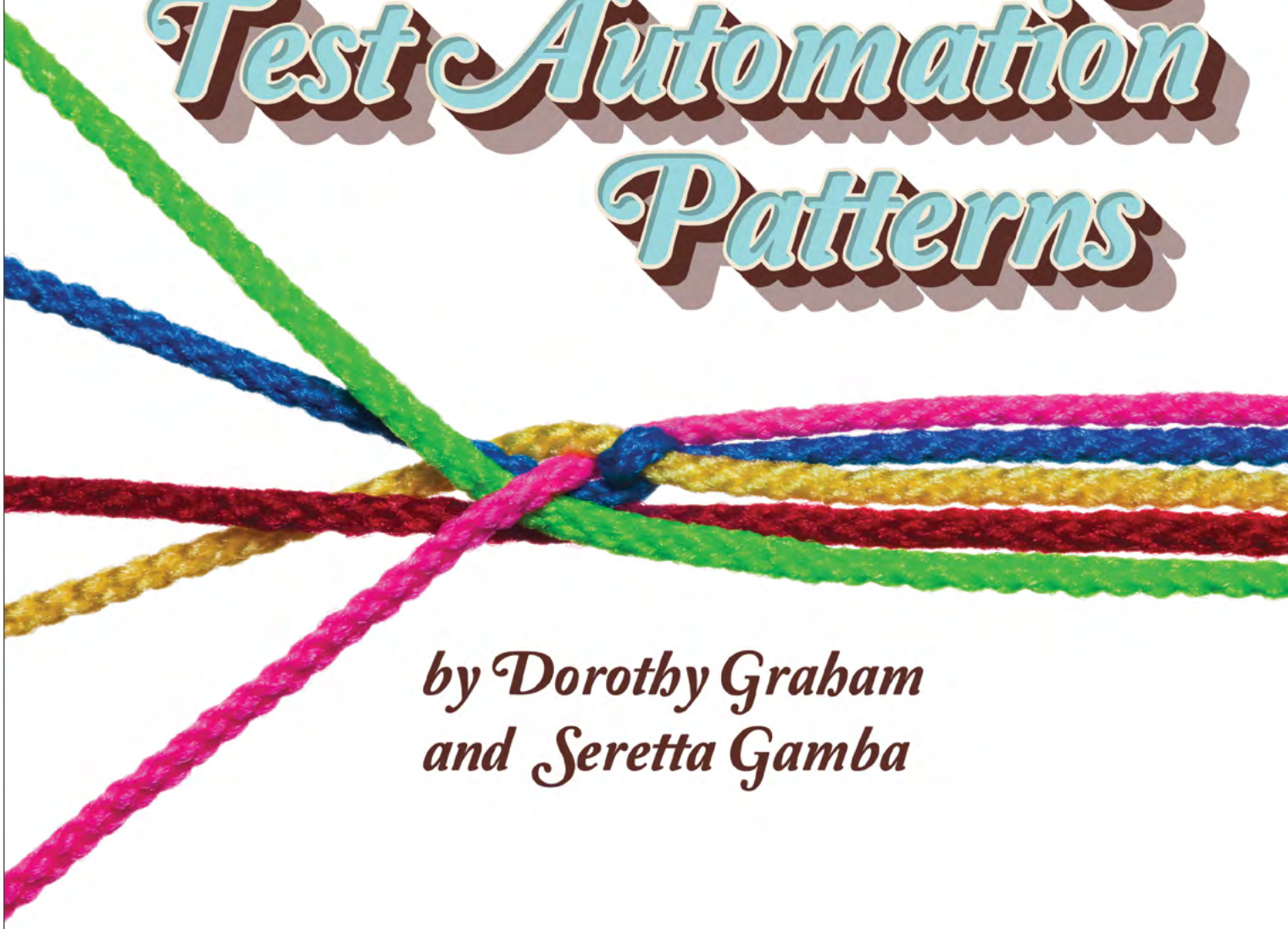
Maybe you can't do a one-fingered push-up, but you can master speed and scale with Sauce Labs. Optimized for the continuous integration and delivery workflows of today and tomorrow, our reliable, secure cloud enables you to run your builds in parallel, so you can get to market faster without sacrificing coverage.

Try it for free at saucelabs.com and see why these companies trust Sauce Labs.

Dropbox salesforce YAHOO! PayPal mozilla VISA

Sauce LABS

Understanding Test Automation Patterns



by Dorothy Graham and Seretta Gamba

THINKSTOCKPHOTOS.COM

Let's start with a short story. Nick Knowall has just joined the Software XYZ company. He is young and energetic and has experience with test automation tools. Software XYZ has been performing test automation for a while with an in-house framework that implements all the tricks of the trade (test case independence, tool independence, abstraction levels, and so on).

They use a tool (AAA-Tool) that Nick hasn't worked with before. Nick learns how to use AAA-Tool, and it doesn't take him long to start automating tests. AAA-Tool is state of the art, and keyword-driven automation is so much easier than with the tools he has used before.

However, Nick decides that he doesn't need to use the existing framework. Why take the time to learn to use it, considering the framework developer will retire soon and nobody will be left to maintain it?

Nick is already becoming very productive and appears to be doing a good job. He automates a lot of new tests. Testers and managers alike are really excited with the speed the work is getting completed. At this point, management agrees that the in-house framework is no longer up to date. From now on, new test cases will be automated directly using AAA-Tool and the framework will only be used for automating the older test cases.

Developing new keyword tests is easy and fun, although parameterizing them to be flexible in execution depending on the data is much more boring.

Because it's so easy to write new tests, why bother? Nick also knows that he should write some documentation, but now he doesn't have time. Testers are waiting for the new automated tests, so he will document the tests when he has some spare time later.

A year later, Nick has become an expert in AAA-Tool and leaves the company for a new, more lucrative job.

If you have worked for some time with test automation, you will have either experienced a similar scenario yourself or know somebody who did.

In all these scenarios, there are many common problems (or issues) that need common solutions (or patterns).

Test Automation Issues

Test automation issues are the problems one encounters when performing test automation (such as high maintenance costs), as well as the tasks that have to be completed (like selecting a tool). Test automation issues can have very different roots.

Some are technical in nature, such as inefficient execution or inconsistent data. Other issues are related to company culture, such as inadequate communication or late test case design.

And, of course, many issues arise when management doesn't really support test automation, often because of unrealistic expectations.

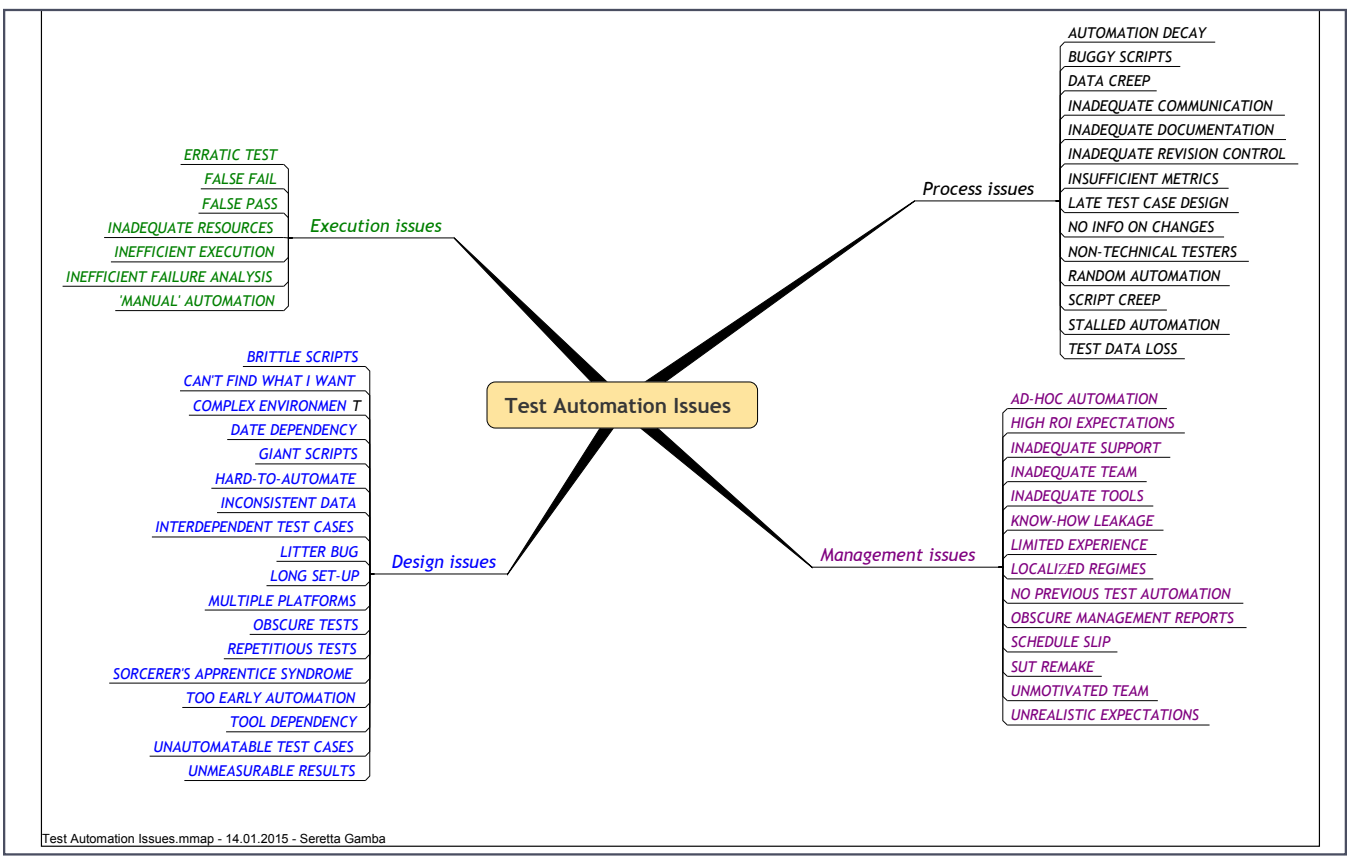


Figure 1: Test automation issues in the wiki

We have classified the test automation issues into four categories:

- ***Process issues:*** The less than optimal way we work with automated tests and tools
- ***Management issues:*** Issues with management, staffing, and objectives
- ***Design issues:*** Lack of a good testware architecture and other technical factors, including maintainability
- ***Execution issues:*** Issues related to the running of tests in their automated form

Test Automation Patterns

Test automation patterns are successful solutions to issues that practically every automator (the person responsible for test automation) sooner or later will encounter.

Because these solutions have been proven repeatedly and are readily available, automators don't have to invent them from scratch—they can just apply an appropriate pattern.

Test automation patterns are different from design patterns used in software development. Design patterns are prescriptive: You have a design problem, and you can solve it with a specific set of code.

Test automation patterns give suggestions for how a particular issue has been solved successfully by other people.

A pattern doesn't necessarily include the perfect solution for every situation, but the suggestions can lead to better and more appropriate actions than you might have thought of otherwise.

Automation patterns can use other automation patterns. Sometimes a pattern can only be applied after another pattern has been carried out.

As an example, think about the pattern for a car; you can't use this pattern if the patterns for paved roads and gas stations haven't been implemented.

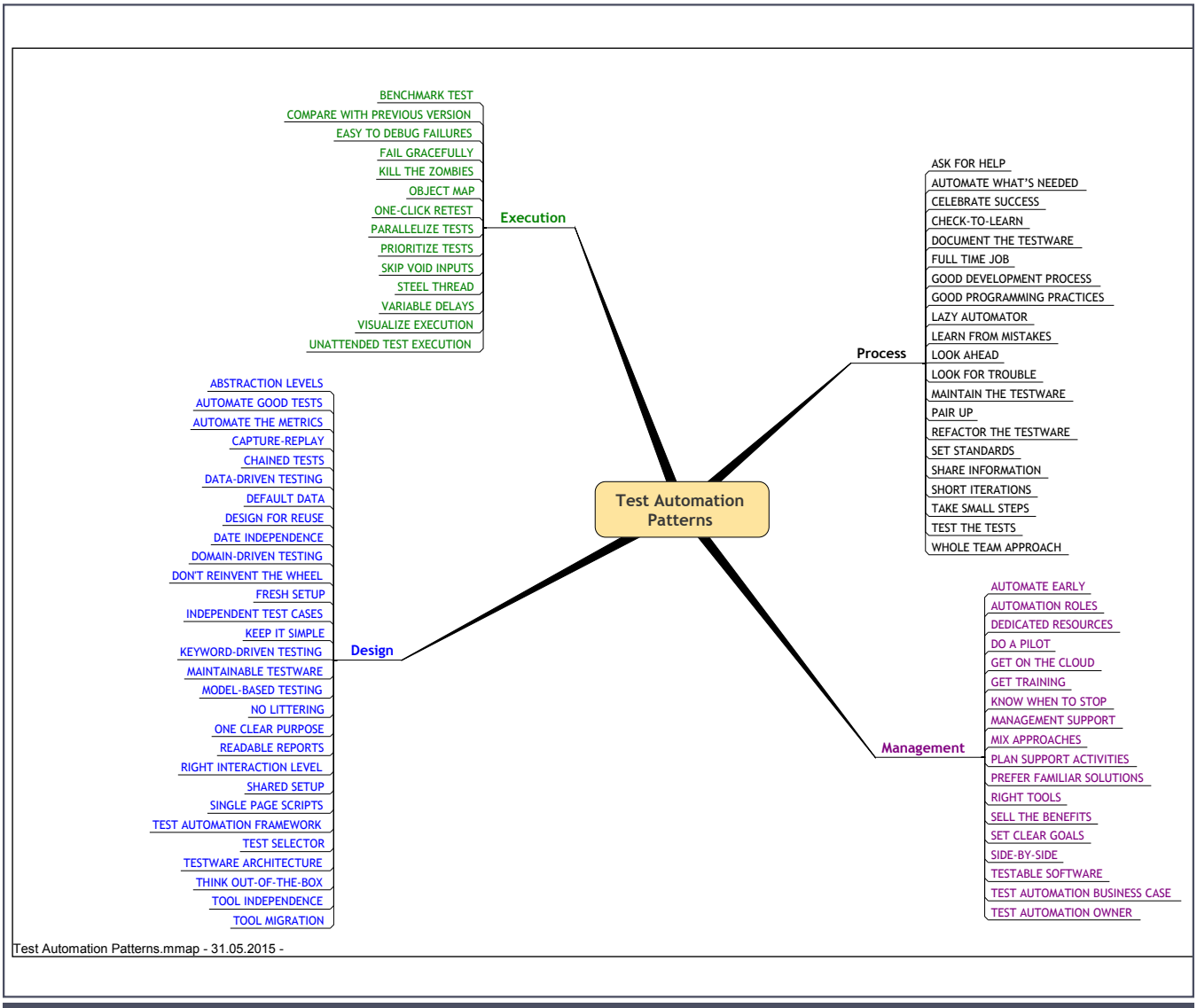


Figure 2: Test automation patterns in the wiki

Test automation issues	Test automation patterns
LIMITED EXPERIENCE (management issue): Nick is new and, even if he already has experience with other tools, he still has to get to know the new tool, the application, the framework, and the processes in use at Software XYZ.	GET TRAINING (management pattern): The pattern gives lots of suggestions that apply in different contexts, including both formal training (in test automation principles or the tool itself) and informal training, such as about the application and internal framework.

Table 1: Mitigating Nick's limited user experience

Test automation patterns also can be classified into four categories:

- ***Process patterns:*** How the test automation process should be set up or how it can be improved
- ***Management patterns:*** How to manage test automation—as an autonomous project or integrated within the development process—and issues with managers
- ***Design patterns:*** How to design the test automation testware so it will be efficient and easy to maintain
- ***Execution patterns:*** How to ensure that test execution is easy and reliable

The Test Automation Patterns Wiki

In the Test Automation Patterns wiki, we describe issues and give examples, along with suggestions as to which patterns to apply. [1] Figure 1 shows a mind map that gives an overview of the issues we have collected so far. Note that we show issues in italic capitals.

Figure 2 shows an overview of the patterns we have collected so far. For each pattern, the wiki explains the context in which it can be applied and gives recommendations and suggestions for implementing the pattern. Note that we show patterns in capital letters (not italic).

To encourage open dialog and to share lessons learned among test automation professionals, this wiki is available for viewing and community updates.

Applying the Patterns

How can you identify the issues and the patterns that may be useful for you as a tester, automator, or manager? The first step is to decide which issues are the most pressing for you now.

There are two ways to do this in the wiki:

- ***General issues:*** Useful when test automation is not going as expected but a specific problem can't be pinpointed—usually because there is more than one issue causing pain. General issues can be imagined as containers for groups of similar, more specific issues, so that by choosing a container one can home in on more specific issues and, finally, to the issue that is most pressing.
- ***Diagnostic:*** The idea for the diagnostic came from observing how a medical doctor pinpoints an illness in a patient he has never seen before. The line of questioning goes something like this: First, the doctor asks the patient general questions (age, previous diseases, family history, and so on) and, depending on the answers, asks additional specific questions. Once all of the questions have been asked along with the necessary examinations, a good doctor can identify exactly what's wrong with the patient.

In the test automation wiki, we start the diagnostic process by asking general questions. Each question has a limited number of possible answers, which lead to more detailed questions, and eventually to the issue that best describes the most pressing problem. Just as with the general issues, this method leads to the patterns that can be applied to solve the issue.

Understanding Test Automation Issues

Let's use Nick's story as an example to demonstrate how issues can arise, and the patterns that could be used to mitigate them. When Nick joined the Software XYZ company, he was told about the company's test framework and the AAA-Tool.

Test automation issues	Test automation patterns
TOOL DEPENDENCY (design issue): The new automated tests will be strongly dependent on AAA-Tool.	TOOL INDEPENDENCE (design pattern): Interestingly, this pattern suggests developing a test automation framework, which is what Software XYZ actually already did. (Nick ignored using it for no obvious reason.)
	ABSTRACTION LEVELS (design pattern): This capability was already incorporated in the existing test automation framework.

Table 2: Not taking advantage of lessons learned or existing best practices

Test automation issues	Test automation patterns
KNOW-HOW LEAKAGE (management issue): Automation know-how (in this case, support for the framework) is easily lost if only one person is in charge. The real issue is probably inadequate support.	SHARE INFORMATION (process pattern): Specific knowledge needs to be shared—in this case, about the benefits the framework brings, especially long-term.
INADEQUATE SUPPORT (management issue): Management has not really supported the framework. Otherwise, a backup developer resource would have been found long ago.	MANAGEMENT SUPPORT (management pattern): Support from management is needed not only at the beginning of an automation effort, but on an ongoing basis. To achieve it you may need to implement other patterns, such as (1) SELL THE BENEFITS and (2) TEST AUTOMATION BUSINESS CASE.

Table 3: Ensuring that the automation test framework is supported

The first issue is shown in table 1, along with the pattern that would have helped.

In this case, the best pattern for lack of experience is for Nick to receive training.

Next, rather than take the time to learn the existing framework, Nick starts using the tool immediately. Table 2 shows that not taking the time to understand existing tools and approaches can result in reinventing the wheel for no good reason.

Nick assumes that the existing framework shouldn't be used because the developer will retire, and nobody can maintain it. The issue here has nothing to do with Nick. If one developer leaves, will no one be able to support the framework? This doesn't sound like good management! By clearly identifying the issue leading up to the main issue, there are several patterns in table 3 that can be used to address the core issue.

This is an example of how one seemingly obvious issue masks a root cause issue. In this case, applying patterns for the root cause should resolve the situation.

Continuing with Nick's story, he is very productive, automating new tests with AAA-Tool faster than management expected. The original test framework will no longer be maintained and will be used only for older test cases. Table 4 shows the issue that has been caused and possible patterns to use to correct the situation.

Test automation issues	Test automation patterns
UNREALISTIC EXPECTATIONS (management issue): Management and testers alike feel that automation with the existing test framework took too much time. At this point, Nick and the managers don't realize that this blazing speed of test development up front may mean heavy maintenance tasks later on, and they don't take notice of the framework developer's views.	SHARE INFORMATION (process pattern): Inform management about what automation can and can't achieve. (Listen to the framework developer!) SET CLEAR GOALS (management pattern): The goals of automation have to be shared by automators, testers, and management. Maintenance of automated tests in the future is not being considered now. DO A PILOT (management pattern): With a pilot project it's easy to show what automation can do and what resources or costs will be involved, including the effect on maintenance of different ways of automating.
TOOL DEPENDENCY (design issue): Now it's confirmed! The new automated tests will be strongly dependent on AAA-Tool. Uh-oh.	TOOL INDEPENDENCE (design pattern): Separate the technical implementation that is specific for the tool from the functional implementation of the tests.

Table 4: Jumping into execution mode without understanding the long-term ramifications

Test automation issues	Test automation patterns
INADEQUATE DOCUMENTATION (process issue): Documentation that is not done immediately will probably never be done.	DOCUMENT THE TESTWARE (process pattern): Document scripts and data for later reuse. DESIGN FOR REUSE (design pattern): Reusable testware has to be documented only once.
SCRIPT CREEP (process issue): This issue may not be visible at first, but its impact will be felt when maintenance starts due to so many more scripts having to be adapted.	ABSTRACTION LEVELS (design pattern): This capability was already incorporated in the existing test automation framework.

Table 5: Mitigating the issue with documentation that never gets written

While new tests using AAA-Tool are fun to write, Nick isn't spending any time parameterizing the data and documenting how the tests work. Unfortunately, there ends up being no time for these critical tasks once maintenance work settles in. Table 5 clearly shows the issues and patterns that have evolved over a period of bad decisions.

Finally, after Nick becomes an AAA-Tool expert, he leaves Software XYZ for a better job. The obvious pattern that could mitigate this situation was described in table 3.

Nick's story is just an example, but we see how various mistakes made along the way have sowed seeds for trouble later on. We also see that there already are patterns that could have helped Nick's company to do better with their automation.

Wherever you are in your test automation, we hope the wiki will help you identify the issues that are hindering you and will suggest patterns that can solve your problems. We welcome other example stories and experiences using test automation patterns. So have a go at finding a useful pattern or two for yourself using the issues in the wiki, and share your experiences with us and other wiki users.

Be aware of issues that you may be falling into just as Nick did, and try not to repeat his mistakes.

Conclusion

The test automation issues and patterns in the wiki are still a work in progress. We welcome feedback, particularly your own experiences with test automation patterns.

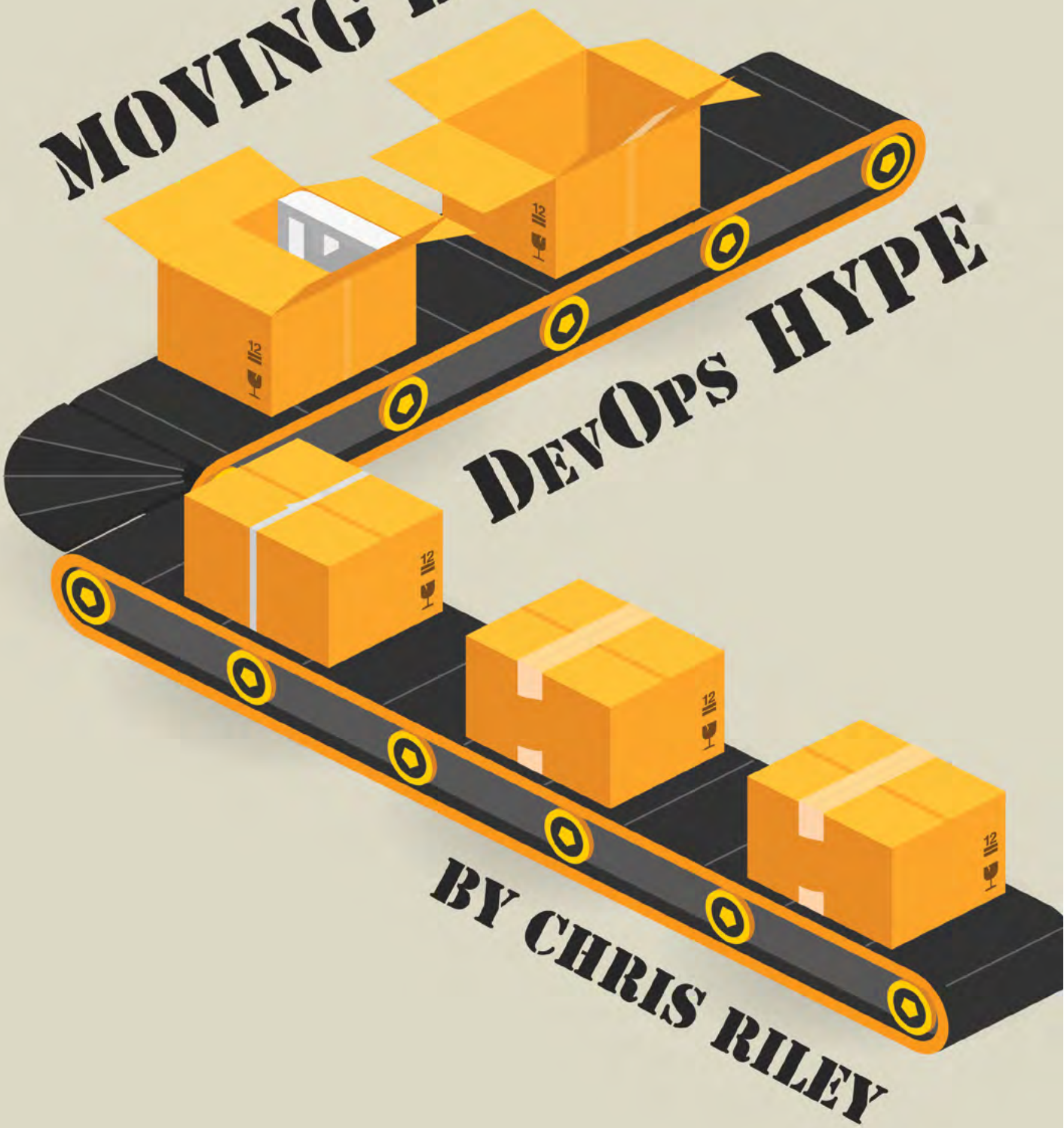
Why reinvent the wheel when there are already lots of wheels for the taking? **{end}**

info@dorothygraham.co.uk
srttgmb@yahoo.com

Click here to read more at StickyMinds.com.

- References

MOVING BEYOND DEVOPS HYPE



BY CHRIS RILEY

THINKSTOCKPHOTOS.COM

The words culture, continuous delivery, iterations, and analytics are usually associated with DevOps, but they tell you very little about what the movement actually is. DevOps is frustrating to some while being useful to others. And even though at times it seems to be the prevailing marketing hype, there is no doubt that its principles and practice can help any enterprise software development organization.

DevOps can be characterized as the assembly line of building, testing, deploying, and updating enterprise applications. At its core, DevOps is the framework that enables software to be released faster and with better quality. When you boil it down to a least common denominator, this is the release management goal that teams have maintained since the beginning of software development. DevOps simply formalizes "more frequent releases at a higher quality" into a framework that guides team members into a continuous flow from code to end-user.

How you envision DevOps in your organization depends on your starting point. Are you a developer in a large enterprise who wants to streamline deployment, a developer in a small startup who thinks IT is going away, or a person in IT in an agile development shop who is excited about adopting DevOps and wants to lead the movement? Each backstory impacts your first impression and a surprising amount of what you think DevOps is. There are two separate DevOps: the movement and the practice.

Movement: Some DevOps evangelists have been guilty of comparing all who write code to the giants who have mastered DevOps, such as Netflix and Facebook. This has mistakenly made people believe that DevOps is an unattainable pipe dream. But your organization doesn't need to be Google to benefit from DevOps. Software quality in frequently released code is in everyone's best interest.

The movement can be a source of frustration for many because some of those in the movement represent a development environment, which is not practical in existing enterprises and development organizations.

Practice: The process and procedures incorporated in DevOps are not necessarily new. Large organizations have dedicated departments focused on this for years. Apple is an example of a company that has invested in DevOps teams for each application unit for quite some time. Etsy, arguably one of the leaders in DevOps practices, is another organization that uses the DevOps framework.

Culture May Destroy Any Possibility of DevOps Success

Culture can be a highly distracting element of DevOps by being responsible for the deaf ears and turned heads from existing development teams. It is unfortunate that this can happen, and it seems it is the result of startups whose team members have job titles such as "ninja" or "master."

Even if you don't like to label culture, elements of what needs to be accomplished already exist in every team. Culture exists no matter what. What DevOps is telling us is that instead of letting culture happen accidentally, it needs to be deliberate. People and politics tend to be the largest yet hardest to quantify inhibitors to success. Only in a deliberate culture is it possible to identify bottlenecks and ultimately seek to eliminate them.

Once you have the entire team working at a unified drumbeat, you can focus on DevOps execution consisting of two tactical components: *process* and *tooling*.

Process: The processes in the DevOps framework are well established and could stand on their own. They are put in two broad categories: *automation* and *metrics*. Automation leads to processes: continuous integration, continuous delivery, continuous deployment, and automated testing. These processes take code from developer to production as quickly as possible, considering quality each step of the way. The idea is to move far away from long-distant waterfall releases or from agile's biweekly releases to a continuous deployment of code within a day or even hours after the developer commits to production. Instead of release chunks, it's a river of code.

This continuous deployment process may not be applicable for all applications simply because some environments don't have the user base or transaction numbers to justify the work. However, continuous delivery is available to most, and continuous integration is a goal that should work for the entire organization.

With automation, humans alone cannot keep up with the documentation and performance of such a system. The only way to do this is to build in metrics. "Measure everything" should be the mantra of all DevOps teams, and great analytics tools that track everything from development to production infrastructure have come a long way to free teams from thinking about collecting information to more real-time analysis. Such tools deliver analytics, alerts, and trends that help the team know what is going on in real time and what has taken place in the past.

Metrics also play into the theme of results. Everything in DevOps is driven by results **and metrics become the source of documentation ensuring that DevOps environments are sustainable and extendable**. This removes developers from their own branch (or fork) of code and gets them interested in the quality of the entire production release. It enhances IT's scope from just keeping servers up and running to an interest in how efficiently the code is being deployed.

Tooling: DevOps typically assumes three elements: *culture*, which is represented by the *people*; the *process* implemented by those people; and the *tools* used to execute those processes. Together, they form a hierarchy with tools intentionally placed at the end of the list. It is common for teams to expect tools to automatically create a DevOps operational environment for them. Organizations that lead with tools often find the tools taking over and defining the processes—and ultimately the culture—for the teams, instead of the other way around.

Once you embrace culture, process, and tools, you have DevOps.

Migrating to DevOps

As long as teams seek faster software delivery without sacrificing quality, the processes and people should naturally flow into a DevOps environment. DevOps doesn't have to be some unreachable, complex process either. Rather than attempting to perform a massive transformation, consider setting up DevOps with a building block approach. It is important to remember that DevOps is not an end—it is a means.

We've all heard the infamous "This is the way we have always done things." But those who are reluctant to change should agree that faster software is required to compete. Because all companies' applications have become directly tied to revenue generation, improving their creation is not a choice. On the flip side, even the most modern software development shops will learn they need IT to ensure that, no matter how fast they move, the production to delivery chain itself needs to be sustained.

For existing organizations, there are two options for adopting the DevOps framework: starting out with *new DevOps teams* or *slipstreaming* into existing processes.

New development units: For this to work, upper management needs to commit to building brand-new development units from the bottom up that embrace a modern development culture, process, and tools. This does not mean that existing development teams are eliminated; rather, they often will persist for support and maintenance of legacy application versions. This approach works well when there are brand-new applications or when there is a shift from client-server applications to web application counterparts.

Slipstreaming: This is the process of integrating new practices into an existing environment without stopping current operation or replacing whole chunks of the existing processes at once. In my opinion, slipstreaming is the ideal approach, and by far the less costly of the two. However, to see the results and achieve some of the bigger goals like continuous integration or delivery, it usually takes more time and is threatened by existing habits and priorities.

To start your transition to DevOps, *automate something*. This can be a quick win that builds confidence, and excitement. Take one process with high opportunity and low risk level (like end-to-end testing or functional testing) and automate it.

You could offload functional testing by using test automation scripting to show the immediate benefits of testing code just after a developer commits. These tests should find defects even before manual testing, which should optimize the time spent hunting down defects by the existing testing team. A commitment to adopt automated testing makes front-end development part of quality and it opens the door for focusing on test strategy and exploratory testing.

Which approach you pick is not just a technology choice, it is a strategic choice. Both options imply that there will be changes to the structure of teams. In both cases, IT becomes a services organization and a facilitator while developers gain more responsibility and accountability for software quality in production than they ever had before.

When Things Get Rough

After you have defined your approach, the most difficult part of transitioning to DevOps is dealing with the people involved. In nearly every case I've witnessed, it is difficult to keep the system working well when people leave. The time wasted in meetings explaining and reconciling definitions of various technical dialects can be huge in most organizations. This waste of time will single-handedly halt any move to faster processes using DevOps.

There are various ways to tackle people-related issues head on. But first you have to have critical mass from a team that is empowered and wholeheartedly believes in modern development. You can start by making the pipeline design something everyone can contribute to. Let the entire team decide what to automate first. Break down existing barriers to communication and encourage QA to give IT infrastructure and developers suggestions for improvement. In the most successful modern development shops, QA has an important holistic strategic role as well as a tactical one.

If you work in a large organization, you might want to create a conduit for the DevOps framework. Organizations, such as Wells Fargo, have taken a slightly different shared services approach. The shared services department is usually responsible for DevOps reports to IT, but their customers are really the developers. This organization is responsible for creating and vetting a library of tools and processes, making them available and encouraging developer adoption.

Don't be surprised if you don't claim big wins overnight. And with small successful steps that will motivate the team, things can change quickly.

Ensuring DevOps Success

Small startups to large organizations have all proven that DevOps works by reducing development costs, improving product quality, and ultimately resulting in happier customers. But DevOps is unique for each organization and is not a one-size-fits-all approach. To provide a successful DevOps environment, you need the right people and the right strategy.

IT and software development must have a mutual responsibility for DevOps success. IT needs to loosen up on the reins and help developers become more aware of what they are already doing. And developers need to better understand how to collaborate with IT operations to keep their environment working efficiently so they can focus on what they do best—coding new features.

DevOps might still sound like hype, but its objectives should be clear within your organization. Without a sound DevOps approach, an organization will fall behind on software quality and practices.

The first step is getting everyone on board and working toward the same goal. **{end}**

chris@fixate.io

GAIN INSIGHT INTO THE MOST ANTICIPATED THOUGHT LEADERSHIP RESEARCH IN THE FIELD OF QUALITY ASSURANCE AND TESTING!

Get a preview of the *World Quality Report 2015-16* at HP Discover, Las Vegas • Tuesday, June 2, 2015 4:30 p.m. - 5:30 p.m. Visit http://h30614.www3.hp.com/Discover/home for more information.

Meet Capgemini and HP subject matter experts at a special session and discover the latest highlights from the upcoming report featuring these key topics:

- Testing and QA budgets
- Digital (SMAC developments and Testing)
- Agile and DevOps
- TCoE models (hybrid models)
- TEM and TDM
- Test and QA across industrialization
- Automation and standardization lifecycle
- Security Testing

To learn more, please visit:
www.worldqualityreport.com or www.capgemini.com/testing-services

ABOUT CAPGEMINI

With more than 130,000 people in 44 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2013 global revenues of EUR 10.1 billion. Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

SOGETI

Six Ways to Use Business Analyst Superpowers in Agile

-by Joy Beatty and James Hulgan

THINKSTOCKPHOTOS.COM

Business analysts (BAs) deal with their fair share of difficult stakeholders, ranging from business stakeholders who won't participate in elicitation, to developers who won't bother reading requirements, to project managers who think BAs are just scribes.

As organizations adopt agile approaches, we are faced with a new kind of obstacle from stakeholders who don't think they need business analysis. These stakeholders might think the BA isn't needed because the delivery team gets to talk directly to customers to understand requirements. Managers may structure their agile teams full of developers, thinking that any one of them can perform the BA role in scrum planning or during any given sprint.

But without someone who is truly dedicated to business analysis work, who is going to determine whether you failed to identify any stakeholders, interfaces, or data objects? Who will do the analysis to figure out the most critical priorities, rather than making decisions based on emotion? Who will do the detailed elaboration to make sure the story can be implemented successfully? Even in agile, you still need someone to perform these tasks.

Who Performs Business Analysis?

Although this varies from organization to organization, the business analysis function can be performed by a designated BA, the ScrumMaster, or the product owner. In some pure forms of agile, all the team members are interchangeable and anyone can act in the role of business analyst, as well as developer, tester, product owner, and so on. We don't really care what you call the role; we just care that someone is performing business analysis.

When BAs do their job well, it seems like they have superpowers. They can predict what details will be needed, they can figure out what is most important at any point in time, and they can bring a difficult stakeholder around to actually work with the team as a contributor (and not as an antagonist). Someone in a dedicated BA role throughout a project's lifecycle can usually provide a more objective perspective than a member of the team who is in the weeds performing specific technical tasks. There are six BA superpowers that are most important to agile and nonagile projects.

Superpower 1: The BA Can Serve on Both the Delivery and Value Teams

The delivery team is responsible for implementing a solution that meets the business needs, and the value team is responsible for defining the value the business is looking for from a solution. In comparison, the value team defines what is most valuable to build, and the delivery team builds the most valuable things first. The product owner and the BA really should stand together on both of these teams.

In many organizations, the product owner won't have the analysis experience to fully discover what is most valuable to develop. For example, the BA can use business objectives models and objective chains to assign dollar values to different features. This benefits the product owner's ability to prioritize user stories for each sprint.

The BA will also use process flows, state models, business data diagrams, and ecosystem maps to ensure that no stories are forgotten.

When it comes to elaborating user stories, the BA will use the same business analysis skills to fill in the details necessary for the delivery team. These artifacts include thorough acceptance criteria and models that supplement user stories, such as state models, UI mockups, display-action-response models, and decision models. Being able to step away from the delivery team to communicate with the value team is an advantage for both teams because developers can focus on developing and testing without having to worry about meeting with the value team to clarify requirements. It gives them more heads-down time to accomplish their work. The value team doesn't have to worry about interrupting important work flow and, with a dedicated BA, they now have a trusted team member who can communicate effectively to the delivery team.

Superpower 2: Underneath Our Capes, I'm a BA and She's a ScrumMaster

When I take off my superhero cape, I'm still just a BA—I'm not a ScrumMaster, a developer, or a tester. I can support all of those functions, but I have different strengths. And the same is true with a ScrumMaster. You shouldn't assume the ScrumMaster can do requirements well. These are completely different roles and require different skills. Some organizations attempt to reduce the number of people in a delivery team by combining roles. Instead, why not consider sharing the BA and the ScrumMaster on more than one project? This is where a good BA who is able to shift contexts can really shine by adding insight to both projects based on what she is seeing in the other—especially in related projects.

Attempting to make the ScrumMaster an effective BA will require in-depth education about requirements practices. She will have to learn to focus on understanding the business needs and prioritizing those within the project and process constraints.

Attempting to make the BA the ScrumMaster means the BA will have to learn more about the interworkings of the Scrum process. For nonagile projects, attempting to combine the BA with the project manager role won't work either, for similar reasons.

Superpower 3: I Can Be a Product Owner Proxy

The reality of many organizations is that a product owner (PO) isn't as available as we need them to be— especially for fast-paced agile projects. That could be a result of the product owner not being collocated with the rest of the team, or it could be because the PO has more pressing business duties other than the project. A BA can often perform the product ownership role when the PO is not available, as long as the BA understands the priorities, tasks, and needs from the product owner. A BA can prioritize, answer developer questions, and make decisions to support the product owner.

The BA needs to know his own limitations, though, to bring in the product owner when necessary. When there are multiple Scrum teams engaged, a BA can act as PO on each team, reporting to a chief product owner.

Regardless of the reason or the exact nature of acting as a product owner proxy, the BA and the product owner need to work closely together across a broad range of topics with stakeholders. This requires that the BA maintain a clear list of decisions that are made while acting as a proxy.

Superpower 4: The BA Has Outstanding Analysis and Communication Skills

A BA typically spends her time developing analytical and communication skills to perform solid business analysis. Just because a BA is working on an agile project doesn't mean that analysis and communication stops. In fact, quite the opposite—this analysis is necessary to make sure that the team understands what is most important to concentrate on first. No surprise, but that is why communication is a core agile principle for team collaboration.

By all means, don't treat BAs as scribes. A BA acts as a sounding board, feeds product owners' analysis to make informed decisions, and supplements higher-level thinking with detailed ideas. Having a BA superpower implies being skilled at the thing the ScrumMaster cares most about: blocking and tackling outsiders so that the delivery team can focus on their work.

Superpower 5: Requirements Don't Come with a Super Sign-Off

Complying with the principles of agile, a mindful BA should collaborate with the product owner to avoid documenting too many detailed requirements too soon. Similarly, the rest of the team needs to ensure that they are continuously looking at requirements throughout the project as they are written, prioritized, and elaborated upon. This means that the business stakeholders no longer have to sign off and approve them at the start of a project. The team will have to trust that user stories are created in time, choosing the right items to put in the backlog and focusing on delivering the most value as soon as possible. The BA and product owner keep the longer-term roadmap in mind so that the team has some idea where this ship is heading.

As stories are loaded into the backlog, the team should become familiar with what scope lies ahead. This helps ensure that the team and the BA aren't on completely diverging paths as the project progresses. Before a new sprint commences, product backlog items are prioritized based on what is most important and what produces the best value. The BA should elaborate the most important stories with more detail one to two sprints ahead of when they will be developed so that the team is able to estimate the effort to develop them. The team should become familiar with the stories and request additional details before or within the sprint, as possible.

Superpower 6: Don't Think I've Got Secret Superpowers

Most of us relish how agile changes the cadence of when and how work is defined and performed. BAs no longer have to listen to project managers tell us there are only three weeks to create requirements for an upcoming release. Project managers who have no interest in hearing how long requirements development should take or what scope can be cut to meet their deadlines must assume that the BA can deliver the impossible.

When it comes to working ahead on future sprint requirements, please don't tell the BA that the requirements for the next sprint need to be created in four days without any discussion. It is far easier to plan to create the details one to two sprints ahead. That should give the BA plenty of time to elaborate. In my experience, BAs are pretty good at estimating the level of effort to elaborate enough details for a user story (write acceptance criteria and supporting models). The lesson is for the team to not expect the BA to commit to unreasonable planning timeboxes.

Summary

The business analysis function lends itself equally well to agile and nonagile projects. If you are moving to an agile approach, don't ditch the BA. Instead, figure out how a BA can become your project's superhero by helping support your product owner by acting as a product owner proxy, by answering questions from development, by prioritizing backlogs, and by identifying and elaborating on user stories. **{end}**

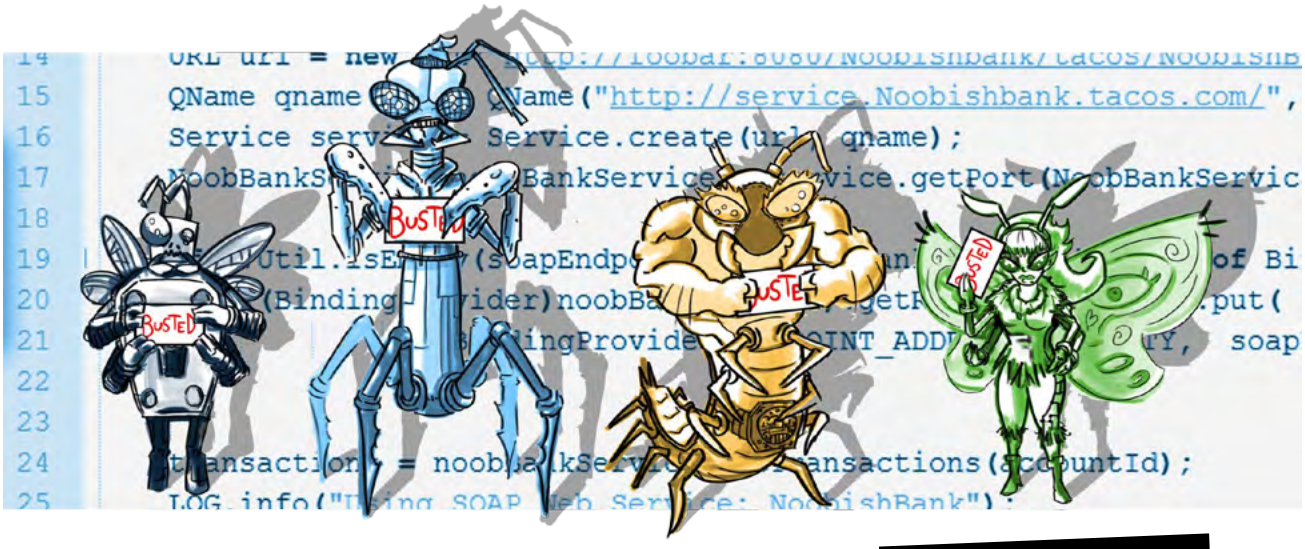
joy.beatty@seilevel.com

james.hulgan@seilevel.com

PARASOFT®

BUSTED!

Software bugs can't hide from **Parasoft's advanced defect detection and prevention technologies**



Parasoft software quality technologies, such as **static analysis, unit testing, peer review,** and **coverage analysis** ensure defect prevention for the next generation of applications.

STATIC ANALYSIS

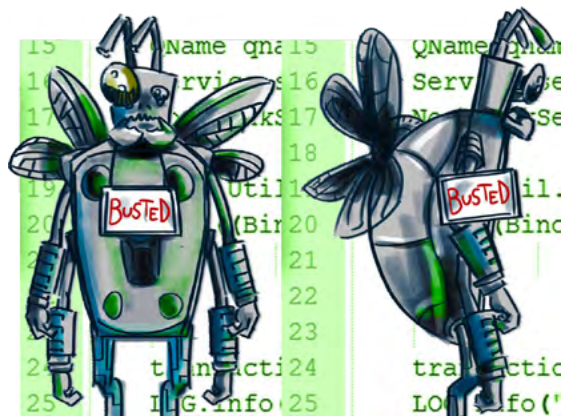
UNIT TESTING

PEER REVIEW

COVERAGE ANALYSIS

Visit **www.parasoft.com/busted** to sign up for a free evaluation.

Stop by the Parasoft booth to learn more about how Parasoft can bust bugs in your code!



Business opportunities start with a good idea and a good code. Beyondsoft's software development and management services help companies capture new market opportunities by becoming mobile and applying data insights.

Meet us at

BSC/ADC IN LAS VEGAS, JUNE 7-12, 2015

BOOTH # 20

BEYONDSOFT

(877) 896-5859 | info@beyondsoft.com | www.beyondsoft.com

IT SERVICES · APPLICATION SERVICES · PRODUCT ENGINEERING · INFRASTRUCTURE MANAGEMENT · MOBILITY · CLOUD

FAQ

expert answers to frequently asked questions

Does Anyone Really Do All That Is Recommended by the ISTQB Tester Certifications?

by Claire Lohr
clohr@computer.org

The International Software Testing Qualifications Board (ISTQB) gathered groups of subject matter experts to create syllabi (foundation, advanced, and expert) that are comprehensive compendiums of best practices for diverse topics related to software testing. They also created a glossary supporting all the syllabi and devised examinations to be used to certify the knowledge of the test takers.

As of December 2014, ISTQB has issued more than 380,000 certifications in more than 100 countries worldwide, with a current growth rate of more than 13,000 new certifications per quarter. Many people who actively perform some aspect of testing, including test managers, have been exposed to the ISTQB vocabulary and techniques espoused by the various syllabi for these certification exams. But are they using them regularly and deliberately?

What about using all those terms defined in the ISTQB glossary?

With the historically rapid increase in the number of practitioners, each team answered the need to communicate effectively by inventing its own terms. Then some terms became more common, and "standard" definitions were created by organizations such as the IEEE. Most practitioners, however, have never even wanted to see these definitions, much less use them.

Most organizations have some overlap with the ISTQB glossary, but not much, and they may even have conflicting definitions. The glossary is designed to support best definitions, not necessarily common definitions. It will take a very long time for this to evolve into common usage—it is costly for an organization to change from its own well-understood definitions to the "standard" terms. The new ISO testing standards (the ISO 29119 series) are consistent with the ISTQB, so at least the standards world is internally consistent.

Does anyone prepare all those separate documents?

Absolutely not. High-integrity systems (i.e., safety-critical, high-reliability, with a large financial impact) document much of the information suggested, but they combine the documents as much as possible—e.g., test cases, procedure, and log are commonly combined. Many of the less rigorous applications want one test plan containing all planning and execution information, a defect report database, and a test summary report.

Does anyone use all those specification-based black box techniques?

Industry practitioners use the techniques they know about that are applicable to the systems they are testing. Boundary value is universally applied, even if the practitioner does not know that it has a name. Many learn about the decision table during exam preparation training and are eager to apply it in their organizations.

Does anyone use rigorous structure-based white box techniques?

For most of the industry, the extent of white box testing depends on the individual developer. Safety-critical systems commonly mandate rigorous white box testing. Security concerns are increasing the awareness of the need for comprehensive white box coverage, so this will likely be increasing (untested code paths have unknown security).

Does anyone use the experience-based testing techniques?

Finally, a resounding yes. All organizations, with or without rigorous black and white box testing, conduct experience-based testing, frequently at multiple test levels.

The challenge is that software engineering is a new discipline and went from a handful of practitioners in the 1940s to a huge number, affecting every aspect of society today. The short answer to whether practitioners are doing all the ISTQB recommends is: "Not yet." The longer answer, regarding the degree to which the information is in use: "It depends on the need for quality and the maturity of the practitioners for each individual system." It is still helpful to know what best practices are available for planning future improved practices; it is impossible to hit a target if the target itself is unknown. **{end}**

Does Your Code Suffer from Broken Windows?

Using the analogy where a home with broken windows increases the possibility of a crime, don't ignore fixing low-severity defects.

by Jennifer Gosden | jennifer.gosden@gmail.com

Low-severity defects are the ugly ducklings of a defect backlog. Dismissed as having little merit, low-severity defects receive scant attention after being initially documented. The existence of these defects in code is considered part of the cost of doing business in software development. They are considered a necessary evil due to tight deadlines and prioritization necessities. The typical defect management approach is to not pay any attention to fixing low-severity defects. Besides, who achieves code that is clean, anyway?

There is evidence that calls into question this approach. As it turns out, fixing low-severity defects may enable teams to achieve long-term excellence in software quality.

Low-severity defects, from the day they are initially recorded, are regarded as a nuisance, of no overall consequence, and of little value. Many defects are closed by the team without a fix ever being made. Closing low-severity defects can become a bad habit. Chances are a new defect will be opened in the future for the same reason by another tester and it, too, will not be fixed.

> "Closing low-severity defects can become a bad habit. Chances are a new defect will be opened in the future for the same reason by another tester."

Even if you believe that these lingering issues don't reflect general product quality, the same resolution process remains. Low-severity defects will always be prioritized for remediation by the developers after higher-severity levels.

This approach is a mistake. It is an antiquated practice based on years of prioritization rules that do not take a holistic approach toward code quality. The rules of prioritization need to change to incorporate the evidence that has come from the implementation of the broken window theory.

This theory illustrates two fundamental concepts:

1. Low-severity defects, when left unfixed, increase the likelihood of medium-, high-, and critical-severity defects.
2. By reducing the number of low-severity defects, applications should have fewer defects overall regardless of level.

The broken window theory is a law enforcement practice stating that by policing minor infractions, the number of major crimes decreases in an area. For example, if there is a broken window in a home and the broken window remains unfixed, the home is more likely to be vandalized as a result. However, if the broken window is fixed, the overall safety of the home should remain sound. The safety improvement of the repaired window provides an additional improvement to the home's quality.

In his book *The Tipping Point*, [1] Malcolm Gladwell discusses how the broken window theory is the reason behind the 81 percent drop in the crime rate for New York City from 1990 to 2009. This is the steepest decline in this time frame in any US city. Despite factors that would lead to a surge in crime, such as rising unemployment, there was a decrease in the rates of homicide, robbery, and burglary when the city focused on the enforcement of misdemeanors. The implementation of similar practices in Albuquerque, New Mexico, and Lowell, Massachusetts, has resulted in corresponding drops in overall crime when focused efforts were applied to clearing trash, fixing street lights, and enforcing building codes.

Extending the broken window theory concepts to code, cleaning up small defects should actually reduce the frequency of the big infractions. The result of mitigating the little things has a residual benefit of preventing errors overall. Cleaning up and preventing the low-severity defects will lead to a lower number of medium-, high-, and critical-severity defects being created in the first place. The cycle will unfortunately require more time to clean up more of the low-severity defects. In my experience, however, this has the benefit that the implementation of new features will occur at a higher level of quality and, subsequently, a higher level of user satisfaction. Clean code is easier to maintain, easier to automate, and easier to add new features than code that is full of defects.

Perhaps you have seen evidence of these patterns in other

situations. When you clean up your desk, closet, or garage, you'll usually take a little extra time and effort to keep it that way for a while. But when you don't focus on it, it drifts back into a disorganized state, resulting in your taking extra time to clean it up again. Instead of allowing the drift backward to occur, consider maintaining it through a little ongoing care and effort.

The same is true with software code.

Taking the broken window theory to heart, expand your testing strategy to attend to low-severity defects as standard practice. If your developers are fixing code that has defects of different levels of severity in the same area, require that low-severity defects are also fixed. Or why not consider hosting an annual hack-a-thon event, with developers cleaning up as many of the low-severity defects as possible in an eight-hour session? Encourage your team to maintain the area of clean code over the long term and watch the quality of your overall product dramatically improve.

Fundamentally it comes down to this: When your users encounter little things that are wrong, how can they have confidence that the important things are done right?

Clean up the broken windows in your house of code. **{end}**

Click here to read more at StickyMinds.com.

- References

index to advertisers

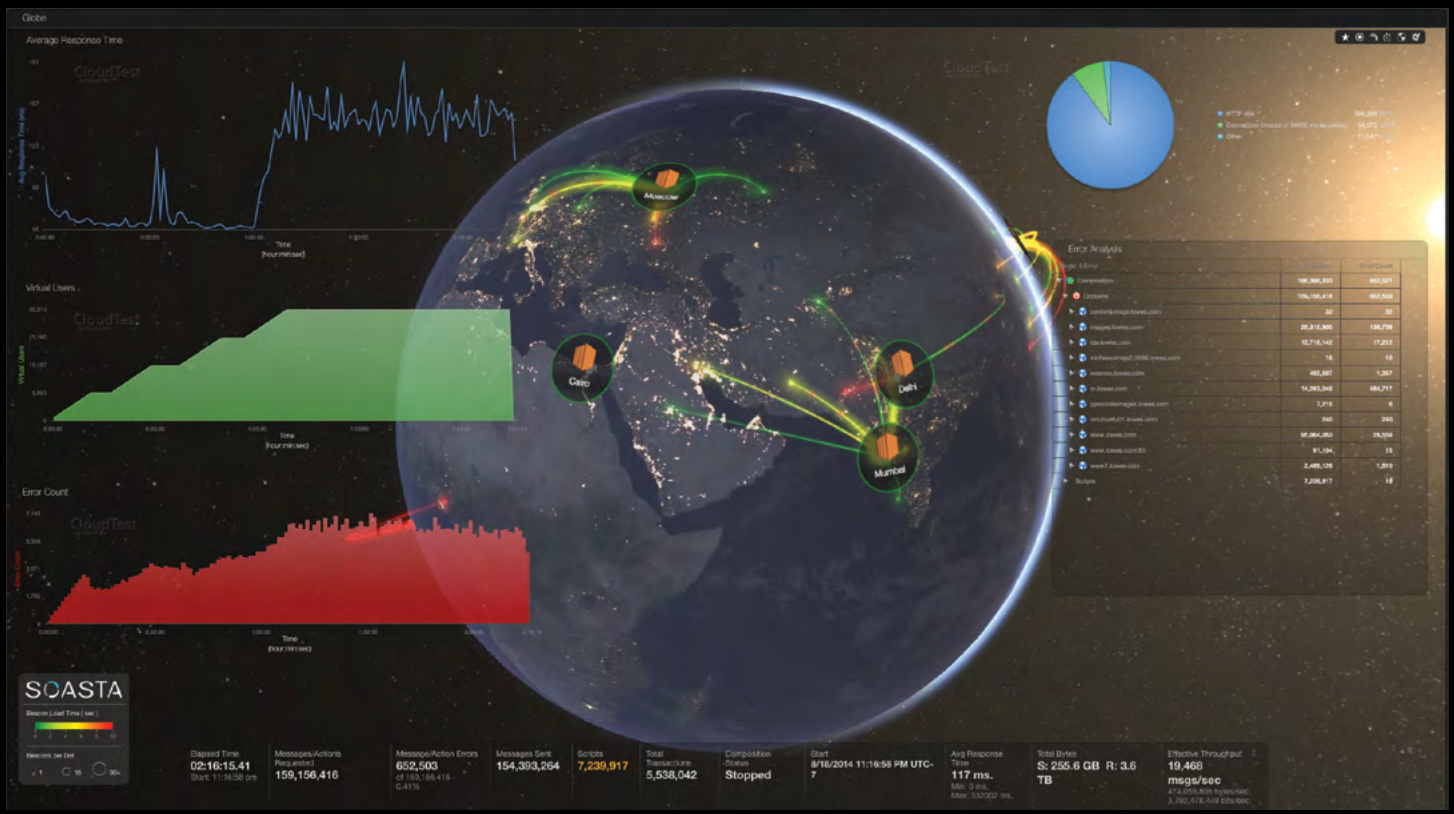
Display Advertising
advertisingsales@sqe.com

All Other Inquiries
info@bettersoftware.com

Better Software (ISSN: 1553-1929) is published four times per year: January, April, June, and September. Print copies can be purchased from MagCloud (http://www.magcloud.com/user/bettersoftware).

SOASTA

Modern apps demand modern testing



Optimize your results with the power of real user data

Monitor Real Users	Measure at Cloud Scale	Optimize Continuously

Learn how: www.soasta.com