

BETTER™ SOFTWARE

A TECHWELL™ PUBLICATION

SEVEN GUIDELINES FOR A GREAT WEB API

TEST AUTOMATION

Perils, pitfalls, and promise

IMPROVE AGILE QUALITY

Take the three-pillar approach to quality nirvana



The Best Place for Software Testing Solutions

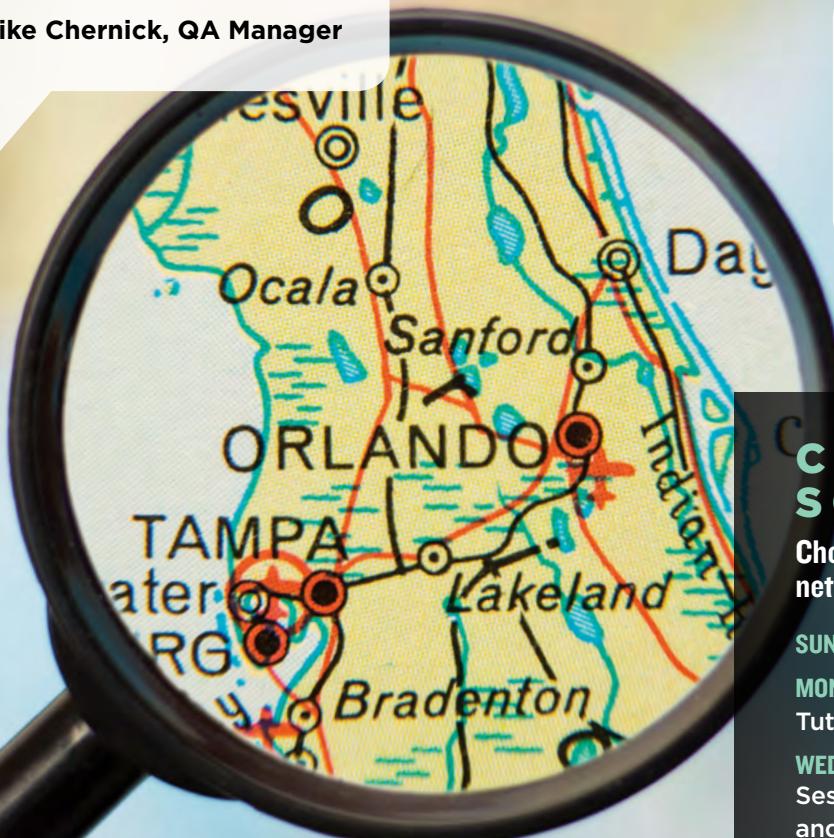
"The most effective training a QA manager can have."

Mike Chernick, QA Manager

Save up to \$400 when you register by March 4, 2016.

"Wonderful first time experience."

Ken Malin, QA Test Analyst



<https://well.tc/STAREAST2016>

CONFERENCE SCHEDULE

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, Women Who Test, & the Workshop on Regulated Software Testing (WREST)

STAR EAST
A TECHWELL EVENT

Orlando, FL
May 1-6, 2016
Renaissance Orlando
at Sea World®

Keynotes by International Experts

STAR EAST
A TECHWELL EVENT



Lessons Learned in (Selling) Software Testing

*Keith Klain,
Doran Jones*



Open Source Test Automation: Riding the Second Wave

*David Dang,
Zenergy Technologies*



Lightning Strikes the Keynotes

*Lee Copeland,
TechWell Corp.*



Telling Our Testing Stories

*Isabel Evans,
Independent Consultant*



The Evolution of a Software Tester: From Novice to Practitioner

*Dawn Haynes,
PerfTestPlus, Inc.*

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

Selenium Test Automation: From the Ground Up

Jeff "Cheezy" Morgan, LeanDog

Get Your Message Across: Communication Skills for Testers

Julie Gardiner, Hitachi Consulting

Critical Thinking for Software Testers

Michael Bolton, DevelopSense

Testing the Internet of Things

Jon Hagar, Grand Software Testing

Plan, Architect, and Implement Test Automation within the Lifecycle

Mike Sowers, TechWell Corp.

Successful Test Automation: A Manager's View

Dorothy Graham, Software Test Consultant



MAY 4-5 THE EXPO

Visit Top Industry Providers Offering the Latest in Testing Solutions

**TOOLS
SERVICES
TECHNIQUES
DEMONSTRATIONS**

TO REGISTER CALL 888.268.8770
<https://well.tc/STAREAST2016>

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

Automated Testing of Desktop. Web. Mobile.



Robust Automation



Seamless Integration



Broad Acceptance



Quick ROI



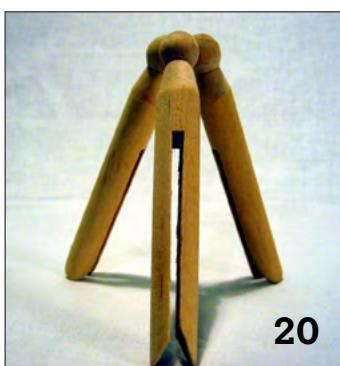
Why Use Ranorex

www.ranorex.com/why





CONTENTS



in every issue

Mark Your Calendar	4
Editor's Note	5
Contributors	6
Interview with an Expert	10
TechWell Insights	34
Ad Index	37

Better Software magazine brings you the hands-on, knowledge-building information you need to run smarter projects and deliver better products that win in the marketplace and positively affect the bottom line. Subscribe today at BetterSoftware.com or call 904.278.0524.

features

12

COVER STORY

SEVEN GUIDELINES FOR A GREAT WEB API

Web APIs have opened up a brave new world for app collaboration. James Higginbotham presents a series of guidelines that every programmer should consider in the design and implementation of a great API developer experience.
by James Higginbotham

20

IMPROVE AGILE QUALITY—THREE PILLARS AT A TIME

A key component to being agile is the adoption of testing from the very inception of the project. According to Bob Galen, to achieve a high degree of quality assurance, there needs to be a careful balance among development and test automation, software testing, and cross-functional team practices.
by Bob Galen

26

MOVING TEAMS TOWARD AUTOMATION: PERILS, PITFALLS, AND PROMISE

There is no magic bullet to create an effective test automation environment. But, Steve Gibson believes that creating a test automation vision, adopting metrics and delivering value throughout a project lifecycle puts an organization on the right path.
by Steve Gibson

30

PEOPLE SHOULD THINK AND MACHINES SHOULD TEST

Testers often develop programmatic tests that mimic manual test conditions. Harry Robinson and Doug Szabo use real programming examples to show how the computer can provide better test coverage than the test author conceived.
by Harry Robinson and Doug Szabo

columns

7

TECHNICALLY SPEAKING

HOW TO ASSESS AND IMPROVE DEVOPS

DevOps can take substantial effort to successfully implement, according to Bob Aiello and Leslie Sachs. By understanding existing development and deployment practices, you'll be able to properly assess the best steps to transition to an outstanding DevOps environment.
by Bob Aiello and Leslie Sachs

36

THE LAST WORD

PLAYING GAMES TO IMPROVE SOFTWARE

You may not have heard about gamification, but instructional designers are now using game principles to help with retention of learned material in many forms of training. Ross Smith and Rajini Padmanaban believe that developers' UX and app design can benefit from gamification.
by Ross Smith and Rajini Padmanaban

MARK YOUR CALENDAR

training weeks

Testing Training Week

<http://www.sqetraining.com/trainingweek>

February 8–12, 2016

Atlanta, GA

Testing Training Week

March 7–11, 2016

Boston, MA

Testing Training Week

April 4–8, 2016

San Diego, CA

Testing Training Week

software tester certification

<http://www.sqetraining.com/certification>

January 26–28, 2016

Houston, TX

February 1–5, 2016

Philadelphia, PA

Advanced Tester Certification

February 23–25, 2016

Little Rock, AR

March 1–3, 2016

Charlotte, NC

March 14–16, 2016

Albuquerque, NM

Denver, CO

March 15–17, 2016

San Francisco, CA

March 29–31, 2016

Vancouver, CA, Canada

Tampa, FL

April 12–14, 2016

Nashville, TN

April 26–28, 2016

Washington, DC

conferences

Mobile Dev + Test

<http://mobiledevtest.techwell.com>

April 17–22, 2016

San Diego, CA

Westin San Diego

IoT Dev + Test

<https://iotdevtest.techwell.com>

April 17–22, 2016

San Diego, CA

Westin San Diego

STAREAST

<http://stareast.techwell.com>

May 1–6, 2016

Orlando, FL

Renaissance Orlando at Sea World

Agile Dev West

<http://adcwest.techwell.com>

June 5–10, 2016

Las Vegas, NV

Caesars Palace

Better Software West

<https://bscwest.techwell.com>

June 5–10, 2016

Las Vegas, NV

Caesars Palace

DevOps West

<http://devopswest.techwell.com>

June 5–10, 2016

Las Vegas, NV

Caesars Palace



Publisher
TechWell Corporation

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

FOLLOW US



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
TechWell Corporation
350 Corporate Way, Suite 400
Orange Park, FL 32073

THE NEW TECHWELL

With the first 2016 issue of *Better Software* magazine, there are more changes going on than just the calendar year.

You may know *Better Software*'s parent company as Software Quality Engineering, or SQE, but we recently rebranded the company as TechWell Corporation. The new name reflects our growing focus on the entire software development lifecycle.

Our TechWell.com website has been completely redesigned to highlight our conferences, training, and online communities: AgileConnection, StickyMinds, and CMCrossroads.

To find *Better Software*, click MORE on the TechWell.com homepage. While you're on the new TechWell site, check out TechWell Insights for timely, tech-focused stories published every weekday.



This issue of *Better Software* is loaded with some awesome feature articles, starting with James Higginbotham's blueprint for writing a great web API. And before taking the plunge into test automation, you'll want to read Steve Gibson's "three Ps:" perils, pitfalls, and promise.

There's no doubt that software development teams have embraced agile. I've heard Bob Galen speak about how to treat quality as a feature on every project. His insightful article about the three pillars of agile quality provides real-world advice for how to balance quality initiatives with your team.

Harry Robinson and Doug Szabo include sample code snippets for tests that can be scripted according to their four rules for test automation. Every software developer and QA engineer will want to absorb their approach.

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 as I enjoyed working with these wonderful authors.

And don't forget to spread the word—SQE is now TechWell Corporation, and *Better Software* magazine has the same great content as ever!

A handwritten signature in black ink that reads "Ken Whitaker".

Ken Whitaker
kwhitaker@TechWell.com
Twitter: @Software_Maniac

Contributors



BOB AIELLO is a consultant and software engineer specializing in software process improvement, including software configuration and release management. He has more than twenty-five years of experience as a technical manager at top New York City financial services firms, where he held company-wide responsibility for configuration management. Bob is vice chair of the IEEE 828 Standards Working Group on CM Planning and a member of the IEEE Software and Systems Engineering Standards Committee (S2ESC) Management Board. Contact Bob at Bob.Aiello@ieee.org or visit cmbestpractices.com.



Agile methodologist, practitioner, and coach **BOB GALEN** helps guide companies and teams in their pragmatic adoption and organizational shift toward Scrum and other agile methods and practices. Bob is the president/principal consultant at RGCG LLC and an agile evangelist at Velocity Partners. Bob is a Certified Scrum Coach, Certified Scrum Product Owner, and active member of the Agile Alliance and Scrum Alliance. Bob's book *Scrum Product Ownership—Balancing Value from the Inside Out* addresses the gap in holistic quality strategies in agile-focused organizations. He can be contacted at bob@rgalen.com.



The director of quality at ReverbNation, **STEVE GIBSON** has a passion for agile development, test automation, and coaching teams in how quality fits into the agile belief system. Of his seventeen years of testing experience, Steve has spent the past seven years honing his test automation craft using open source technologies like Cucumber and Selenium. Steve can be reached at steve@qualitymindset.net.



With experience in architecting, building, and deploying APIs, **JAMES HIGGINBOTHAM** is an API consultant who enjoys helping businesses balance great API design and product needs. As a trainer, James enjoys equipping cross-functional teams to integrate their talents toward building first-class APIs for their product or enterprise systems. He is a frequent speaker at technology events and conferences. You can contact James at james@launchany.com.



As senior director of engagement, **RAJINI PADMANABAN** leads the engagement and relationship management for some of QA InfoTech's largest and most strategic accounts. She has more than fourteen years of professional experience, primarily in the software quality assurance space. Rajini actively advocates software quality assurance through evangelistic activities, including blogging on test trends, technologies, and best practices and providing insights on software testing to analyst firms such as Gartner and IDC. She can be reached at rajini.padmanaban@qainfotech.com.



A long-time freelancer in the tech industry, **JOSIAH RENAUDIN** is now a web content producer and writer for TechWell, StickyMinds.com, and *Better Software* magazine. He also writes for popular video game journalism websites like GameSpot, IGN, and Paste Magazine, where he writes 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@techwell.com.



For the past thirty years, **HARRY ROBINSON** has developed and tested software at Bell Labs, Hewlett Packard, Microsoft, Google, and several startups. Harry is a pioneer in advanced software test techniques, especially model-based testing and high-volume automation. Recently, Harry contributed a chapter, "Exploratory Test Automation: An Example Ahead of Its Time," to the book *Experiences of Test Automation*. Harry holds two patents on test automation techniques. Harry can be reached at harryr@harryrobinson.net.



LESLIE SACHS is the coauthor of *Configuration Management Best Practices: Practical Methods that Work in the Real World*, a New York state certified school psychologist, and the COO of Yellow Spider Inc. Leslie has more than twenty years of experience in the psychology field, working in a variety of clinical and business settings. A firm believer in the uniqueness of every individual, Leslie has recently done advanced training with Mel Levine's "All Kinds of Minds" Institute. Reach her at LeslieASachs@gmail.com.



Director of engineering at Microsoft, **ROSS SMITH** is a Fellow of the Royal Society for the encouragement of Arts, Manufactures and Commerce, which aims to enrich society through ideas and action. Ross blogs, is an author of *The Practical Guide to Defect Prevention*, and holds six patents. He developed 42Projects, which focuses on management innovation, trust, and the application of games at work. A frequent speaker at the Serious Games Summit and Gamification Summit, Ross works with teams inside and outside Microsoft on deploying games in the workplace. Contact Ross at rosss@microsoft.com.



For more than twenty years, **DOUG SZABO** has been a software analyst, programmer, and, most recently, a software tester. Having transitioned from developing software to testing software in 2007, Doug's current pursuit is intelligent software test automation in the health care sector. Past projects have involved software for a variety of technologies, including GPS navigation, 3D rendering, geographic information systems, handheld mobile computing devices, and data analytics. Reach Doug at doug.szabo@gmail.com.

How to Assess and Improve DevOps

Transforming your organization to effectively use DevOps isn't easy. To properly assess your DevOps environment, get the terminology right.

by **Bob Aiello and Leslie Sachs** | Bob.Aiello@ieee.org and LeslieSachs@gmail.com

DevOps has become a compelling enterprise computing practice, bringing the promise of fast and reliable application deployments—from small bug fixes to major feature enhancements. The focus of DevOps is to improve the communication and collaboration of key groups, including development, quality assurance, and operations. Technology managers at all levels in an organization are insisting on DevOps with all its promise and excitement. DevOps requires applying effective principles and practices that usually take substantial effort to implement successfully. This article will help you understand how to assess your existing practices leading to effective and scalable DevOps.

Getting the Terminology Right

The first step to understanding and applying DevOps is making sure you are using the right terminology. Some managers inaccurately use terms like continuous delivery when they are really describing continuous integration. The difference is that continuous integration combines code from two or more developers, through an automated process that deploys to a test region. Continuous delivery refers to always having your baseline potentially ready to be deployed, often actually delivering changes to production, but hiding them from end-users through a technique called feature-toggle. Some managers have taken to using DevOps terms to mean completely unrelated concepts and that makes it hard to deliver a consistent set of practices throughout the organization.

If you want your DevOps transformation to be successful, start by delivering training so that everyone has a common understanding of DevOps principles and practices. It is also a best practice to begin by assessing your existing practices so that you know exactly what needs to be improved. To successfully implement DevOps, you must first assess your existing practices and then create a roadmap to improve your key functions thereby enabling your team to deliver fast and reliable releases.

Assessing Your DevOps Environment

Successfully assessing and improving DevOps require that you understand your existing practices, identify what needs to be improved, and then choose the right initiatives to assign resources and organizational focus. When we conduct assessments, we meet with a wide array of stakeholders from product and development managers to testers and operations engineers, and we ask them to explain what they feel is being done well and what needs to be improved. DevOps encompasses configuration management practices such as source code management, build and release engineering, automated environment management, along with release coordination and change control—all practices fundamental to an effective DevOps transformation. You may discover that some teams have well-defined automated build and deploy procedures

but need help with basic source code management procedures such as baselining code with tags or merging code correctly on branches. This may indicate that they will struggle with advanced practices, such as feature toggles, too.

In our experience, customers use continuous delivery and continuous deployment interchangeably and insist that these practices be given a high priority even though basic functions such as source code

management, build and release engineering, or even effective change control may be more urgent. Teams implement continuous integration and deploy the code to a test environment, calling this practice continuous deployment. And this can be very misleading. Actually continuous deployment implies that changes are deployed all the way to production and sometimes this may actually not be desirable. For example, banks that bypass change control may find themselves in violation of federal regulatory requirements. Make sure that your assessment identifies key practices that are required in your industry, especially when mandated by federal law.

How good is your source code management? During an assessment, we often learn that teams need help with source code

management practices. We always ask teams if they can prove with certainty that the right code is running in production, which is known as a physical configuration audit. Another key benchmark is whether or not they can easily identify unauthorized changes whether from human error or malicious intent.

Monitoring the environment is another key DevOps function, without which organizations often fail to recognize serious incidents are about to occur. For example, systems running low on memory may trigger messages that alert the operations team

that systems are about to crash. Monitoring memory allows engineers to fix the problem before there is customer impact.

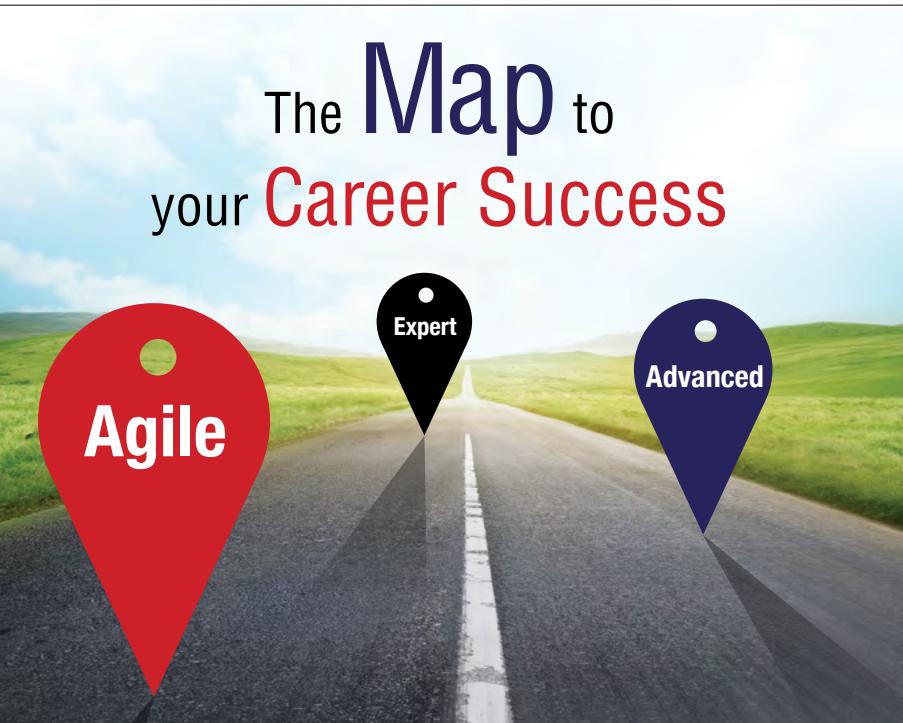
Each step of your application build, package, and deployment should be fully automated and traceable which is a common compliance and audit requirement. DevOps will fail without comprehensive automated application testing. DevOps should enable your organization to deploy code very quickly—even for large systems—on a daily or even hourly basis. Robust, comprehensive automated testing is required to avoid inefficient manual tests that slow down DevOps functions.

Suggested Improvements

Since you do not want to fix things that are not broken, you first need to assess your existing best practices and reach consensus on what can be improved. Understanding what is working well and what needs to be fixed helps to identify a list of tasks that may form the basis of your roadmap to DevOps implementation. We usually recommend starting with a few changes, which helps the team realize that things can actually improve. An added benefit is that your organizational culture should also improve. Focus on specific key initiatives such as ensuring that all broken builds found by the continuous integration server are investigated rather than ignored.

Our focus is on getting deployment procedures documented, reviewed, and fully automated. Agile principles also help drive the DevOps transformation itself by taking an iterative approach to identifying exactly what needs to be improved. As we learn from agile principles, you should be prepared to manage change in your own processes while striving for continuous improvement and operational excellence. [\(end\)](#)

The Map to your Career Success



Agile

Expert

Advanced

Get to the next level in your software testing career.

ISTQB Software Tester Certification has agile, advanced and experts paths that can show you the way, giving you recognition that sets you apart.

If you are ready to take the next step in your career, choose your software testing career path now at www.astqb.org/map.

ASTQB
American Software Testing Qualifications Board, Inc.
ISTQB Certification in the U.S.

Agile Dev Better Software DevOps **WEST**

A TECHWELL EVENT

JUNE 5-10, 2016
LAS VEGAS, NV
CAESARS PALACE

KEYNOTES ANNOUNCED!



DevOps and the Culture of High-Performing Software Organizations

Jez Humble

Humble, O'Reilly, & Associates



The Power of an Agile Mindset

Linda Rising

Independent Consultant



How to Do Kick-Ass Software Development

Sven Peters

Atlassian



The Internet of Things: Through the Mobile Lens

Steven Winter

Trizic

Agile Dev West topic areas:

- Agile Implementation
- Kanban
- Agile Testing
- Agile Techniques
- Scrum
- Agile Requirements
- Agile Leadership
- Agile for the Enterprise

Better Software West topic areas:

- Leading Projects and Teams
- Business Analysis & Requirements
- Cloud Computing
- Going Mobile
- Software Quality
- Design & Code
- Development Tools
- Testing

DevOps West topic areas:

- Enterprise DevOps
- Project Management
- Continuous Integration
- Tools for DevOps
- Process Improvement

LEARN MORE

bscwest.techwell.com

Kevin Rohling

Years in Industry: **6**

Email: kevin@kevinrohling.com

Interviewed by: **Josiah Renaudin**

Email: jrenaudin@techwell.com

“What's differentiating about this next wave [of IoT] is that we're getting to the point where the technology and our own learning around user experience are allowing us to build things that are genuinely useful.”

“What is the UX for IoT? Is it all going to be based on mobile? Are these things going to be self-configuring? Obviously, the more work you push onto the user, the better.”

“The hardest part of getting your device working is getting it set up, getting it configured. Whether it's connecting to Wi-Fi or whatever that happens to be, that experience is your first touch point with your user.”

“At the end of the day, what do consumers want? They want things that just freaking work, and they want it to work as easily as the things that they have in their house do right now.”

“Your oven is pretty easy to figure out, and you know how to open your fridge. You don't expect things in your house to be difficult to use, and that's why these new connected devices have an adjustment period for users.”

“You hear a lot of buzz on media outlets about a lot of really interesting things happening... but you also hear about companies that end up going under before they actually ship a product.”

“At the end of the day, you want users to have to put in as little work as possible.”

“

At the end of the day,
you want users to
have to put in as little
work as possible.

”

For the full interview, visit
<https://well.tc/IWAE18-1>



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.



YAHOO!

PayPal

mozilla

VISA



sauceLABS

SEVEN GUIDELINES FOR A GREAT WEB API

BY JAMES HIGGINBOTHAM

Browse any technology news website and you will likely read about another product promoting an open web-based API. Companies are making huge investments to expose APIs to both internal and public software developers, resulting in higher conversion rates and increased revenue. Web APIs can be used strategically to extend the ecosystem of your enterprise app by enabling integration with other apps across the enterprise.

Unlike other aspects of software development, APIs at their heart are social and collaborative. They will be shared with other developers, often outside the product team. This means great developer experience is important, as public developers using your API will not have access to your source code, design documents, and database diagrams.

To create a great developer experience, we need to follow some basic guidelines. Each guideline will help shape the direction of your API and perhaps change the way you think about web APIs as a whole.

Guideline 1: Take an API-First Approach

Great APIs strive to make complex problems simple and accessible for developers. They expose technical capabilities and enable collaboration between humans and machines. Yet, most APIs solve a specific problem within an existing application. As a result, the API design is specific to the application and difficult to reuse as future opportunities arise.

Taking an API-first design approach encourages teams to think beyond a specific application. APIs become first-class products within the organization rather than integrated solutions.

APIs can even be viewed as products themselves, with the focus shifting from an internal to an external perspective. We begin to think more about the solution the API could make available to outside developers. This results in better API design and documentation.

For example, companies like Twilio (voice and SMS) and SendGrid (outbound email) have productized their expertise through their APIs. Best Buy developed an open API to allow developers to access their product catalogs and customer reviews, resulting in increased sales. Even the US federal and state governments are releasing public APIs to share more than eight thousand data sets via the data.gov developer portal.

While most of these APIs were designed as products from the beginning, many APIs that were once private are being released as public APIs. Organizations such as *The New York Times*, Thomson Reuters, Brigham Young University, and Uber now provide open APIs for public developers. This means great API design and an API-first approach matters—even if your API is currently used only for internal application development.

Guideline 2: Design Your API from the Outside In

Modern wisdom says developers should use test-driven design (TDD). We write automated tests to ensure that our

software is working and to prevent future regression of bugs. As Tom Preston-Werner writes, “A perfect implementation of the wrong specification is worthless.” [1]

APIs can be regarded as specifications for how to interact with your software system. Attention to the details of the API specifications is important for external developers. Otherwise, poor API design will result in inconsistent responses, diverging error formats, and chatty APIs that require large numbers of HTTP requests.

To prevent these issues from creeping into your API design, adopting an outside-in approach requires modeling the API up front with full documentation.

API modeling: The goal of API modeling is to fully understand and validate the needs of your developers and the end-users, like a UI wireframe. But unlike a wireframe, which focuses on the end-user interaction, API modeling focuses on both developer and end-user goals. Sometimes these goals are the same, but often they are not. API modeling ensures that both sets of goals are met before designing and developing your API. Details on API modeling techniques can be found in the book *A Practical Approach to API Design*. [2]

Documentation-driven design: Documentation-driven design is the discipline of writing documentation as part of API design. Using this approach, teams are able to view their design from an outside perspective. The focus is on the API specification, rather than behind-the-scenes details such as database design. The result is complete API documentation, written up front rather than as an afterthought.

Writing the readme file up front also adds value, as this may be the first experience for developers new to your API. It focuses beyond the specification, on topics such as the API solution, code examples to get started, and links to further resources.

Guideline 3: Write Great Documentation

Because developers won’t have access to your source code, a well-documented API is crucial to developers understanding how to use your API.

Many teams have technical writers who produce beautiful PDF-based documentation. However, PDF or Word documentation can get stale and cause developer confusion. In addition, these formats do not allow for direct API interaction. HTML-based documentation is optimal because developers will have access to the latest updates and can interact with live APIs before a developer writes a single line of code.

Tools such as Swagger, RAML, Blueprint, and I/O Docs provide assistance in documenting APIs. They can also give machine-readable descriptions of your APIs for tool automation. Many of these tools offer interactive capabilities, which allow developers, testers, and technical managers to explore an API from their browsers—no coding required.

Great APIs need well-written documentation, produced thoughtfully by technical writers. Inline code comments often aren’t sufficient, as the focus of inline comments is

on providing understanding for developers with access to the codebase, rather than developers outside the company. Seek out skilled technical writers who understand how to construct complete, understandable API documentation. And don't forget to provide your team with candid feedback regarding the experience of the developer attempting to use the API.

Guideline 4: Create an Intuitive, Consistent Design

As developers use your API, they will come to expect the same experience across all API endpoints. API consistency creates predictability, resulting in a great developer experience. Here are some tips for ensuring a consistent API design:

Avoid abbreviations and jargon: These can confuse developers not familiar with the specific problem domain. For example, use the term "volume" even if the term "power" or "gain" is more familiar to audio engineers.

Use consistent names for resources: Some terms may be interchangeable within your team, but when designing an API, it can lead to confusion. For example, use "accounts" (or "users") consistently throughout the documentation. Avoid mixing the two names unless they mean something different.

Reuse common field names: Refrain from interchanging field names across responses that mean the same thing. For example, don't use `fullName` for one API endpoint and `firstName` with `lastName` for another.

Create URL consistency: Avoid one-off URLs such as `/user/current` when `/user` would suffice. If you choose to use REST constraints, pluralize resource collections (e.g., `/users` and `/users/1234`) and keep singular resources singular (e.g., `/user` for the current user).

Use HTTP content negotiation: If your API will support multiple content types, such as JSON and XML, use HTTP content negotiation to allow your clients to request the content type desired. This is accomplished by using the `Accept` header and returning the content type of responses using the `Content-Type` header.

Consider creating a simple style guide to act as a checklist for consistency during the API design process. Investing time in a style guide will save time in the long run.

Guideline 5: Design for Security Up Front

Too often, security is an afterthought when building software. However, nearly every API will provide access to sensitive data or internal business systems and may share user data. Upfront security considerations prevent poor API design changes that make the API more difficult to use. There are three security concerns that will impact API design:

Protecting data in motion: Most API endpoints will transmit sensitive data. Therefore, teams must secure all data in motion. By using TLS with HTTP, all sensitive information is protected. This includes authentication cre-

dentials and any transmitted data in HTTP headers and request/response bodies.

Preventing data leakage: APIs often begin as an internal solution for a web or mobile application. Over time, they expand to partner integration or open APIs for public developers. It is during this shift from internal to external usage that data leaks are most likely to happen.

As an example, the Tinder API put users of the popular platform at risk through data leakage. Its mobile applications did not display an individual's exact location coordinates, but the API returned specific locations within the response payload. This meant any developer had access to an individual's location because the data was not properly scrubbed for external consumption. [3]

To prevent data leakage, design your APIs as if you were releasing the API to the public—even if you don't plan to do so. Add proper authorization for API consumers to grant or revoke access to specific data fields and endpoints of the API.

Authentication/authorization support: APIs often need to know the identity of the API consumer. This identity may be a specific application developer or an end-user known by the API. There are three common methods to provide identity with an API request:

Password-based authentication: The API consumer sends a username and password, often encoded as Base64 within the Authorization header of each request. When combined with TLS, the credentials are less susceptible to compromise. However, if the password changes, API clients will stop working until updated to use the new password.

Key-based authentication: The API consumer sends a generated API key to identify the application or user making the request. Options for sending the key include using a URL query parameter, using a POST parameter, or via a request header. API keys are better than passwords because they remain the same, even if passwords have been changed.

Token-based authorization: API tokens, unlike API keys, are temporary credentials that must be refreshed after some period of time. Some token-based authorization techniques are also capable of supporting external data access. This includes access to a specific user account on another system, commonly called delegated access. OAuth 2.0 is a common protocol for token-based authorization workflows.

I recommend that developers use a key-based or token-based approach and that they not reinvent the wheel by designing their own authentication or authorization schemes.

Guideline 6: Share Code Examples

While documentation describes what you can do with an API, code examples offer insight into how to use it. Code examples can come in a variety of forms—from a few lines to complete working applications. The amount of code to show depends on your audience.

Teams should provide short, concise code examples that demonstrate how to perform a basic task with the

API. These should eliminate the need for the developer to write code to get started. Instead, offer examples that support simple modifications to parameters for better understanding. Figure 1 shows how Stripe offers a Ruby-based code example to register a new credit card on its service.

```
require "stripe"
Stripe.api_key = "sk_test_BQokikJ0vBiI2H1WgH4olfQ2"

Stripe::Token.create(
  :card => {
    :number => "4242424242424242",
    :exp_month => 10,
    :exp_year => 2016,
    :cvc => "314"
  },
)
```

Figure 1: Code example to access a Stripe API endpoint

Notice how easy it is for developers to quickly copy and paste the code example and try it out. The faster developers can be up and running, the better TTFHW (time to first hello world). API providers like Stripe excel at targeting this metric to just less than five minutes.

As developers become more familiar with your API, they will likely seek out additional code examples. These examples should demonstrate more complex workflows and advanced scenarios. Providing complete demo apps are helpful and can jumpstart a project even faster.

Guideline 7: Provide Helper Libraries

Helper libraries remove the need for developers to write raw HTTP request/response code to consume your API. These libraries, also known as software development kits (SDKs), focus on extending your API to specific programming languages, like the previous Stripe example in Ruby. While API providers are not required to offer helper libraries in multiple languages, the helper libraries do encourage quick adoption. They remove the need to write code for handling the HTTP details for your API.

When deciding which programming languages to target, consider your audience. If you intend to target mobile developers, consider offering helper libraries for iOS and Android platforms. JavaScript is a great choice for those building rich web applications. If your target developers are building server-side applications, consider Java, Python, PHP, Ruby, .Net, and JavaScript.

It is important to note that each helper library should have complete documentation and follow each language's programming idioms. This prevents a library written in Ruby from looking like it was built for a Java developer. This may require hiring outside expertise or using a SDK generation tool such as Swagger code generators or APIMATIC.

Putting It All Together

Designing a great web API involves more than just connecting a database to the web. It requires thoughtful consideration of how developers plan to use the API. By applying these seven guidelines, you will be able to create a well-designed, well-documented API, which should result in a great developer experience for both internal and external developers. **{end}**

james@launchany.com



[Click here](#) to read more at StickyMinds.com.

■ References



A TECHWELL EVENT



A TECHWELL EVENT

APRIL 17-22, 2016 | SAN DIEGO, CA

Get Double the Content for the Price of 1 Registration at the Collocated Mobile Dev + Test and IoT Dev + Test Conferences

Keynotes by International Experts



Mobile and IoT Wins! Now What?

*Jason Arbon,
appdiff.com*

Smartphones now outnumber tablets and PCs combined. Mobile developers and testers make more money than their old-school counterparts. Now that mobile has won the race, a new set of questions arises. How has mobile changed—and how does it continue to change—software technology and the economy? What does this mean to you personally? How does mobile affect technology choices, company strategies, and your career? Jason Arbon shares how to capitalize on the mobile win. Mobile forced a reinvention of how we design, build, and test software. How can these lessons from mobile be applied to web and legacy apps? How about to new IoT apps? And to new apps-as-services? Join Jason for a peek into the future—to the day when an app is just an app, on all platforms, and no one cares about mobile-specific issues anymore... [READ MORE ONLINE](#)



10,000 Years in Your Pocket: The Deep History of Your Mobile Device

*James Dempsey,
Tapas Software*

We live in amazing times with amazing technology all around us. And mobile technology, delivered in iPhones and Android devices, is possibly the most amazing of all. While we designers, developers, and testers strive to make fantastic mobile apps and products, we often spend our efforts fixing the things that are wrong with the mobile experience. Taking a page from recent work in the field of positive psychology, James Dempsey wants us to pause and focus on the positive. Join James to appreciate the deep history of science, technology, and even religion that has led us from communicating with fire and smoke to these wonderful little devices that enhance—and frustrate—us in our daily living. Using calendars, language... [READ MORE ONLINE](#)



The First Wave of IoT—Blood in the Water

*Kevin Rohling,
Emberlight*

In the past two or three years the consumer market has seen the idea of the Internet of Things (IoT) go from a prediction to reality. The first wave of IoT products was largely fueled by the parallel innovation of crowdfunding, which allowed makers and early stage ideas to get off the ground without traditional funding sources. Many feel that the promised innovations from IoT have not yet been realized. Almost weekly another crowdfunded startup announces its closing its doors without ever shipping a product. Products that do ship often offer a poor user experience and are notoriously buggy and insecure. In fact, a recent article—Why Is My Smart Home So {omission} Dumb?—expresses many consumers' opinions about IoT devices. Drawing on his personal... [READ MORE ONLINE](#)



Balancing New Tools and Technologies vs. Risk

*Ellen Shapiro,
Vokal*

Your engineering team wants to dive deeply into the newest programming tool or next generation technology for a mission critical project. How do you balance the promised rewards of a new language, software tool, or hardware technology with the risks of unstable software, hardware that does not work as promised, or new tools that are abandoned? Ellen Shapiro describes how the iOS and Android teams at Vokal approach all the new tools and technologies they evaluate. Discussing manufacturer-built and supported languages like Swift, cutting-edge projects like JetBrains' JVM language Kotlin, Functional Reactive Programming, and the Realm database, Ellen shares how Vokal decides to pursue and test new technologies—and the consequences of those decisions. Once a new... [READ MORE ONLINE](#)

APRIL 17-22, 2016
SAN DIEGO, CA

Mobile Dev + Test 
A TECHWELL EVENT

IoT Dev + Test 
A TECHWELL EVENT

Tutorials

go in-depth with these full- and half-day tutorials

Swift Programming: From the Ground Up

James Dempsey, Tapas Software

Android Development Introduction: A Hands-On Workshop

Ken Kousen, Kousen IT, Inc.

Build Universal Apps for the Windows Platform

Mike Benkovich, Imagine Technologies, Inc.

Develop Your Mobile App Test and Quality Strategy

Jason Arbon, appdiff.com

Advanced Android Development

Ken Kousen, Kousen IT, Inc.

Use Selenium to Test Mobile Web Apps in the Cloud

Brian Hicks, Coveros

Security Testing Mobile Applications

Cliff Berg, Coveros

Testing Web Services and the APIs behind Mobile Apps

Marc van't Veer, Polteq

Super Rad Brainstorming

Jaimee Newberry, SWINGSET, Inc.

Mobile App Project Kick Off: Get It Right the First Time

Jaimee Newberry, SWINGSET, Inc.

Internet of Things: From Prototype to Production

Marc Adams, M2M DataSmart Corporation

Eric King, IoT Smart Labs

Use Mobile/IoT Big Data Analytics to Improve Development and Testing

Jon Hagar, Grand Software Testing

iOS and Swift Quick Start: The Fundamental Pillars of iOS Development

James Dempsey, Tapas Software

Prototyping Wearable Devices Using Android

Lance Gleason, Polyglot Programming Inc.

Top Dev-Ops-Testing Patterns for Mobile and IoT Software

Jon Hagar, Grand Software Testing

**“TUTORIALS AND SESSIONS
WERE GREAT...AND HAVING IT
IN SAN DIEGO WAS PERFECT!”**

—Sandra Nagel, Senior Quality Assurance Engineer, CoStar Group

Mobile Dev + Test



Concurrent Sessions

A TECHWELL EVENT

Mobile Development Topics

Usability vs. Security: Find the Right Balance in Mobile Apps

Can Your Mobile Infrastructure Survive 1 Million Concurrent Users?

Building Connected and Disconnected Mobile Applications

Gradle for Android Developers

Build Smarter Mobile Apps with Real-Time Relevance

Scalable and Collaborative iOS UI Test Automation in Swift

Get Started with Google Fit and Its API

Use the Modern Cloud to Build Mobile Apps

Mobile Testing Topics

Uber's Fascinating World of Inter-App Communications

How to Find Vulnerabilities and Bugs in Mobile Applications

How to Build and Integrate a Framework for Testing Mobile SDKs

Shift Left Mobile Application Testing

Test Infrastructure for Native and Hybrid iOS and Android Applications

Integrate On-Device Test Automation into the Dev-Release Pipeline

Implement Combinatorial Test Patterns for Better Mobile and IoT Testing

Innovations in Mobile Testing: Expanding Your Test Plan

MobileDevTest.TechWell.com
To Register Call **888.268.8770**

"I LEARNED A TON, MET GREAT PEOPLE, GREAT PRESENTATIONS IN EXPO OF AVAILABLE PRODUCTS AND SOLUTIONS WE COULD USE."

—Chris Mosconi, Under Armour Connected Fitness

"I THOUGHT MOBILE DEV + TEST WAS WELL RUN WITH GOOD SPEAKERS IN A NICE LOCATION, THE SPEAKERS WERE VERY WILLING TO ANSWER QUESTIONS."

—Catherine Casab, Complete Genomics, Inc.

IoT Dev + Test



A TECHWELL EVENT

Concurrent Sessions

IoT Development Topics

Turning Smartphones and Smartwatches into Hotel Keys

Guerrilla Test & QA: The Mobile of the Internet of All the Things!

Wearables: Testing the Human Experience

The Internet of Things in Action: Anki's OVERDRIVE Racing Game

Bring Team Interaction into the Living Room

Bending Roku to Your Testing Needs

Apple Watch, Wearables, and Mobile Data—with IBM MobileFirst

IoT Integrity: A Guide to Robust Endpoint Testing

IoT Testing Topics

The 4th Industrial Revolution and IoT Predictions: A Software Perspective

IoT Scalable Deployments with M2M Cellular Networks

Rapid Application Development for Raspberry Pi

Making IoT Enterprise Development Simpler

Hardware Solutions to Start—and Fast-Track—IoT Development

Future Perspective: Cloud Connectivity in an IoT World

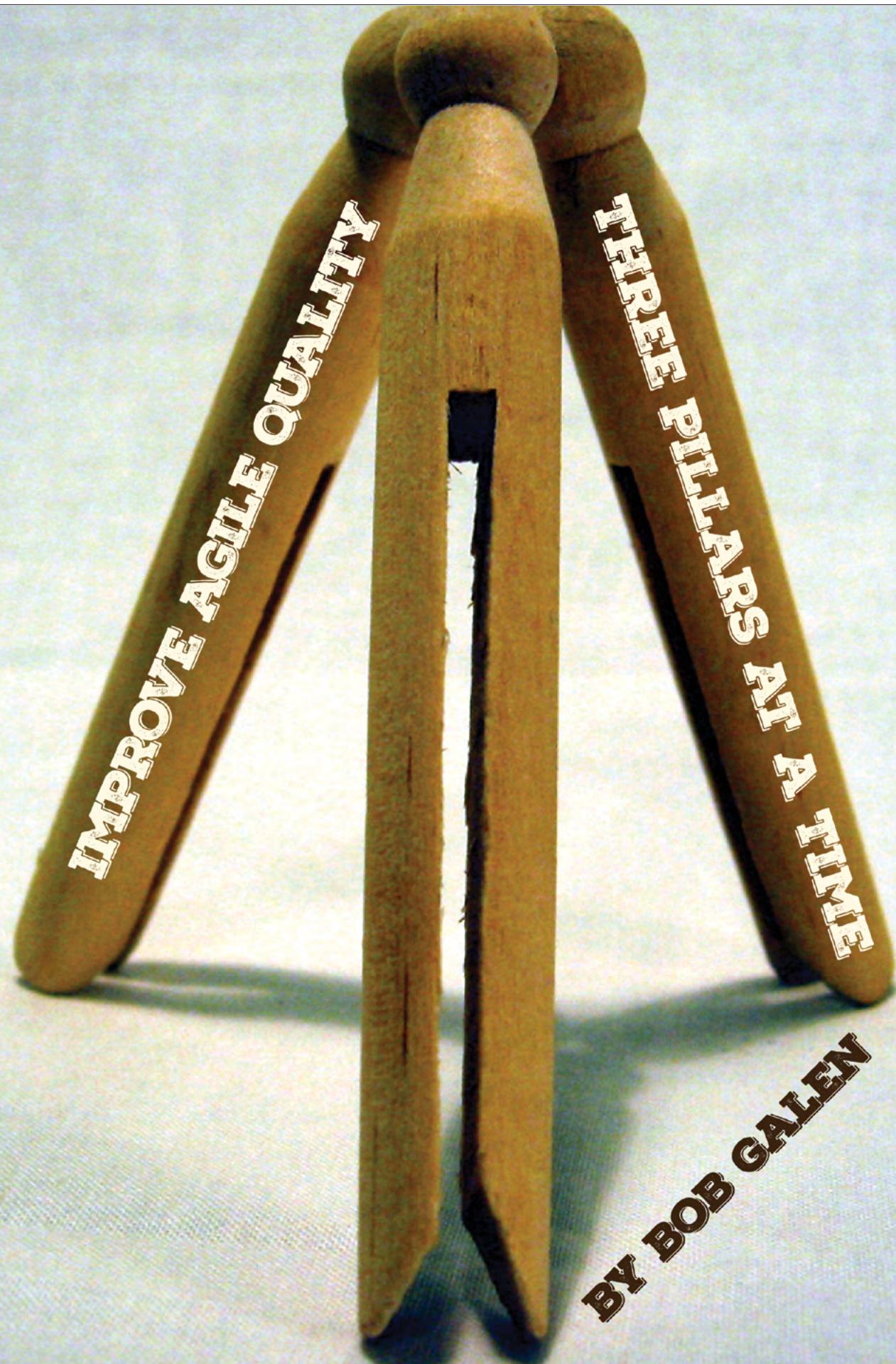
Prototype the Internet of Things with Javascript

IoTDevTest.TechWell.com
To Register Call 888.268.8770

Who Should Attend Mobile Dev + Test & IoT Dev + Test

- Development managers
- IT directors & CTOs
- Mobile designers
- Project managers & leads
- QA managers & analysts
- Software architects
- Software developers & engineers
- Software & test managers
- Test practitioners & engineers





A few years ago, I performed agile-focused coaching and training for an organization. From the outside looking in, it appeared to be an experienced agile organization. It was a global financial firm that delivered their IT projects via highly distributed teams. They seemed to be fairly committed and rigorous in their application of agile methods: They had internal coaches in place, had implemented Scrum, and had been leveraging extreme programming (XP) technical practices for a couple of years. A wide variety of tools were in place for application lifecycle management (ALM), software development, and software testing support.

Too Narrow a Focus

I noticed that the firm had committed to behavior-driven development (BDD) leveraging Cucumber, an open source tool that runs automated acceptance tests. Teams were creating thousands of BDD automated tests while developing their software. From the teams' perspective, there was incredible energy and enthusiasm. Literally everyone contributed tests, and they measured test coverage daily.

These tests were executed as part of their *continuous integration* environment, so visible indicators of coverage and counts were everywhere. I could tell that everyone was focused on increasing the number of automated tests. There was a unity in team goals and focus.

However, a few days into my coaching, I was invited to a product backlog refinement session where a team was writing

and developing user stories. I expected to be an observer, but I quickly realized that the team did not know how to write a solid user story. In fact, they could barely write one at all. After this shortfall became clear, they asked me to deliver an ad hoc user story writing class. Afterward, the team was incredibly appreciative, and everyone seemed to understand the important role stories play in developing BDD-based acceptance tests.

It became obvious over the next several days that the organization was at two levels when it came to their agile quality and testing practices. Everyone was either overcommitted—as in the example of BDD and writing automated Cucumber tests—or undercommitted—as in the struggle to construct basic user stories. The organization lacked balance across most of the agile practices. They also didn't realize that it is the interplay across these practices that drives the effectiveness of agile testing and quality.

To help them improve, I prepared the three pillars of agile quality and testing model to illustrate the balance that is critical across practices. Figure 1 shows the types of activity and focus within each pillar.

The Three Pillars of Agile Quality and Testing

The three pillars form the basis of a balanced quality plan. Let's explore each pillar in more detail.

Development & Test Automation: This pillar represents the technology side of quality and testing. It is not focused on

Three Pillars of Agile Quality and Testing

Development & Test Automation

- Pyramid-based strategy: (Unit + Cucumber + Selenium)
- Continuous integration
- Attack technical infrastructure in the backlog
- Visual feedback—dashboards
- Actively practice ATDD and BDD

Software Testing

- Risk-based testing: functional & nonfunctional
- Test planning @ release & sprint levels
- Exploratory testing
- Standards—checklists, templates, repositories
- Balance across manual, exploratory & automation

Cross-Functional Team Practices

- Team-based pairing
- Stop-the-line mindset
- Code reviews & standards
- Active done-ness
- Aggressive refactoring of technical debt
- User stories, "Three Amigos" conversations

- Whole team ownership of quality
- Knowing the right thing to build; building it right
- Agile-centric metrics
- Center of excellence or community of practice
- Strategic balance across three pillars: assessment, recalibration, and continuous improvement

Figure 1: High-level view of the three pillars of agile quality and testing

testing and testers. Activities include tooling, execution of automated tests, continuous integration, XP technical practices, and support for ALM collaboration tools. It emphasizes Mike Cohn's agile test automation pyramid as the strategy for developing automation in most agile contexts. [2]

Often, the technical aspects are where organizations gravitate first, due to our affinity for tools that can help us solve technical challenges. An important way to think about this pillar is that it is foundational, i.e., the other two pillars are built on top of this tooling.

While this pillar includes testing, tooling, and automation, it inherently includes all tooling related to product development across an agile organization. It provides much of the glue in cross-connecting tools and automation that leads to overall work efficiency and improved quality.

Software Testing: This pillar focuses on the testing profession. A large part of that focus is toward developing solid testing practices. It relies on a team's past testing experience, skills, techniques, and use of tools. This is where agile teams move from a superficial view of agile software testing (test-driven development, acceptance test-driven development, and developer-based testing) toward a more holistic view of quality.

Software testing embraces exploratory, functional, and non-functional testing. This pillar emphasizes exploratory testing because it's an underutilized technique that provides exceptional value within agile contexts. Because agile teams often forget about nonfunctional testing, this pillar provides a gentle reminder to constantly consider all areas of testing.

In addition to broad test reporting, this is where testing strategy, planning, and governance activities are performed. By ignoring traditional testing with all of its process focus and typical lack of value, the software testing pillar should provide a framework for effective professional testing, broadly and deeply applied within agile contexts.

Cross-Functional Team Practices: Finally, this pillar focuses on cross-team collaboration, team-based standards, quality attitudes, and building things properly. Of the three pillars, consider the soft skills area that provides guidance for how each team should operate.

This pillar encompasses traditional code and document reviews that should be highly valued and practiced. Activities include the pairing of team members, formal architecture reviews, design reviews, code reviews, and test case reviews. In addition, the best testing teams insist on rigorous inspection as established by the team's definition of done.

As part of understanding what doneness really means, cross-team physical constraints, conventions, and agreements need to be established with total team commitment.

Foundational Practices

Beneath the three pillars are some foundational principles and practices. For example, it is crucial that quality and testing become everyone's job—not just the job of testers. I find too many agile teams relegate the ownership of quality and testing to the QA department. Continuously coaching the adoption of whole-team quality ownership should be an ongoing focus.

Another component of the three pillars framework is a thread that permeates through the pillars and the foundation. It embraces the core idea that each agile, self-directed team has a basic responsibility to build the right things (customer value) and to build them properly (design, construction integrity, and quality).

Crosscutting Concerns

Beyond the individual pillars, the real value of the three pillars framework resides in connecting crosscutting concerns. In the earlier example of the team struggling with user stories, the results would have been better if they had made the following cross-pillar connections:

- Pillar one implies a balance in your automation strategies, so the BDD work should have been offset by a focus on unit test and UI-based automation. It also implies that acceptance tests would have been connected to the customer via the user story, as-is common practice.
- In pillar two, there are implied standards for software testing. In this case, there would be norms for story writing and acceptance criteria. I would also look for some lifecycle examples of well-written stories that led to automation. This would have served to connect pillar one and two.
- Placing an emphasis on pillar three would have prevented the customer collaboration from being dropped. Independent of the automation, the team would be involved in three amigos conversations that strove to elicit functional value from customer and team collaboration. It would also have emphasized having well-defined stories before writing BDD scripts.

The three pillars are not a guarantee of integrated thinking when implementing quality and testing practices in agile contexts. But they are a strong reminder of gluing crosscutting concerns.

Moving Forward

There are three key takeaways for an organization to adopt the three pillars of agile quality and testing.

- Quality and testing is a whole-team strategy.
- Quality and testing practices intersect across technology, testing, and soft skills.
- A holistic model is needed to create more crosscutting connections for your quality and testing strategies.

Use a balanced, three-pillar approach to effectively transform your organization's agile quality initiatives. [\(end\)](#)

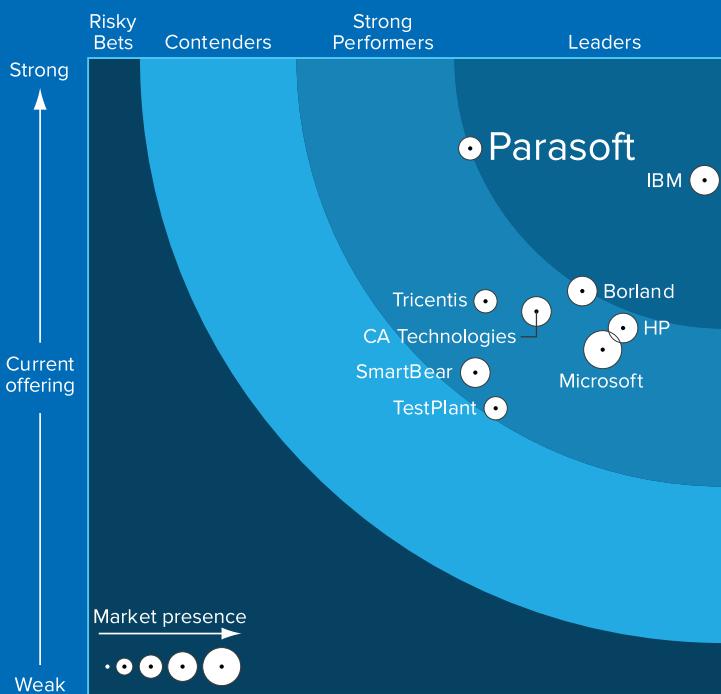
bob@rgalen.com



[Click here](#) to read more at StickyMinds.com.

■ References

Parasoft Service Virtualization



Forrester recognized Parasoft for the strongest current offering for Functional Test Automation

The top award-winning Service Virtualization technology



Download the Forrester Research and Service Virtualization Guide:
www.parasoft.com/bettersoftware_SV

Testing Training Weeks

Featuring up to 16 specialized classes, SQE Training's Testing Training Weeks offer software professionals the opportunity to maximize their skills training by building a customized week of instruction. Courses are offered for QA, test, and other software professionals across the development life cycle and include topics such as agile, DevOps, mobile, test automation, test management, and more.



SOFTWARE
TESTING



AGILE SOFTWARE
DEVELOPMENT



SOFTWARE
SECURITY



MOBILE DEV &
TESTING



TEST
MANAGEMENT



DEV OPS

Software Testing Training Week classes are led by leading subject matter experts, including Rick Craig, Dawn Haynes, Claire Lohr, Dale Perry, and Rob Sabourin. Attendees get both one-on-one interaction with instructors and opportunities to network with software professionals.

SAVE UP TO
\$250
WITH EARLY
BIRD PRICING

“Hands on, interactive training that truly demonstrates how to be successful with agile.”

—Bobbi Caggianelli, Se2, *Fundamentals of Agile Certification—ICAgile*, 2015 Testing Training Week

“The instructor taught very enthusiastically and informatively. Very fun learning experience.”

—Mike Wilkinson, IMO, *Performance, Load and Stress Testing*, 2015 Testing Training Week

“The information has provided an excellent strategy for growth in our environment.”

—Sue Maddock, Bravo Wellness, *Essential Test Management and Planning*, 2015 Testing Training Week



The more training you take the greater the savings!

Maximize the impact of your training by combining courses in the same location. Combine a full week of training for the largest discount!

2016 SCHEDULE

TESTING TRAINING WEEKS

February 8–12 Atlanta, GA
March 7–11 Boston, MA
April 4–8 San Diego, CA
June 13–17 Chicago, IL

August 22–26 Dallas, TX
September 19–23 Washington, DC
October 17–21 Tampa, FL
November 7–11 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 Management and Planning*°		Measurement & Metrics for Test Managers*°	Leadership for Test Managers*°	Test Improvement for Agile**
Risk-Driven Software Testing**		Performance, Load, and Stress Testing*°		

*Not available in Atlanta. °Not available in Dallas.



Project
Management
Institute

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

MOVING TOWARD AUTOMATION

PERILS, PITFALLS, AND PROMISE

BY STEVE GIBSON

There's nothing in the Agile Manifesto or its twelve principles that dictates teams must leverage test automation to be successful. Yet, if you've done any form of agile or adaptive development, you know firsthand how important it is. Agile promotes fast feedback loops. Test automation, when done skillfully, provides exactly that. Running automated tests repeatedly over time provides a heartbeat of quality of your application, informing everyone of your application's stability as you add or remove features from your code base.

There is a path to automation nirvana. It leads to your teams' building valuable, reliable automation. Unfortunately, it's beset on both sides with problems that could turn a valiant effort into a pile of frustration and wasted effort. Learn how you can avoid several obstacles that would get in the way of your achieving test automation.

Absence of Vision

Without a doubt, the most perilous of test automation pitfalls is an absence of vision. From a coherent vision comes a detailed roadmap—something tangible your teams can begin to dissect and understand. Without that shared understanding, teams often are adrift during a project lifecycle. One team might have a different strategy than the next, though they may sit in close proximity. Other teams might be duplicating efforts. Another might have gone off in the wrong direction entirely. Worse yet, teams might be building something unmaintainable that will eventually be neglected or abandoned.

The vision should be simple and state its goals clearly. The primary goal is to build a regression suite that gives your teams confidence that bugs will be found when they're refactoring the application. Refactoring old code can be very dangerous without an automation safety net. It can have unforeseen quality consequences, especially when dealing with crusty code that is overly complex.

Development teams and their product owners need the peace of mind that they can ruthlessly refactor the application without negatively impacting customer experience.

Another important goal is that your automated test suite should liberate your testers from having to repeatedly execute regression tests by hand. That freedom will allow your team to focus on more insightful exploratory testing. If you've ever run a manual regression multiple times, you know it's a repetitive activity that eventually dulls the senses and numbs the mind. Liberated testers are happy testers, and happy testers find better bugs.

A solid automation strategy should detail your expectations for unit, functional (service), and UI tests. Because each type has its own cost and return on investment (ROI), be specific about the amount of time your teams should devote to each. Mike Cohn's automation triangle visually represents the proper ratio for each type of test. [1]

Do not underestimate the importance of a cohesive, actionable strategy. It will be your technical guide on a journey that requires expertise, dedication, and teamwork.

Siloing

I've seen three organizational configurations in which teams create test automation. The first and most damaging is the lone-wolf approach. The responsibilities of creating and maintaining tests fall on a single person, usually the most technical QA engineer on the team. This individual has additional responsibilities for manual testing along with everyone else.

The automation tool used in the lone-wolf approach was likely acquired by management. The third-party tool generates record-and-replay automation and can be created quickly, but those tests are generally fragile by nature. They inextricably bind the user interface (UI) elements to the test logic, thereby ensuring that any slight change in the UI means cascading test failures.

Test breakages of this magnitude mean that many tests will have to be re-recorded, yet there's not enough time available to fix them because of other testing responsibilities. As a result, UI playback tests remain in a failed state. Because of a lack of confidence in the results, the regression suite eventually will get mothballed. The organization will come to associate UI test automation with failure. Convincing the team otherwise will be an uphill battle.

The next approach is one in which there's a small, separate automation team that works apart from other teams. Siloed teams don't socialize their best practices with the rest of the organization. The teams also will find that as their test count grows, maintenance becomes unsustainable. Instead of developing new automation, they're fixing tests and patching the framework.

They don't have the bandwidth to innovate or to share their knowledge and expertise with other teams. Even more dangerous, their separation may lead to a sense of domain ownership over all test automation and a subtle unwillingness to share. The most effective way to get traction with test automation is to make everyone responsible. In agile, there's a notion of doneness criteria that defines how work is deemed to be complete. The entire team agrees to, and is responsible for, doneness.

One example of a doneness criterion is that teams need to manually test the features they develop. Testers orchestrate the testing activities across all team members, including developers. Weaving test automation into your doneness criteria is a great way to make everyone responsible for product quality.

Nontechnical testers can help write the front-end of the tests, while developers or test "automators" write the actual automation code. The goal is to get the whole team to feel a sense of ownership. Everyone's sharing the responsibility for developing, maintaining, and fixing tests increases efficiency by removing the personnel bottleneck. It encourages collaboration and sharing best practices.

Having a unified vision and ubiquitous understanding of the automation strategy are prerequisites for this team-based approach.

The Silver Bullet

Test automation serves as a black box for nontechnical managers. Their understanding of the subject is limited, and their free time to investigate solutions is almost nonexistent. There are products on the market that claim to provide the ultimate test-automation solution—the silver bullet. The perception is that nontechnical testers can start using the product immediately and build a robust automation suite quickly and effortlessly. Managers might get a sense that because no specialized training is necessary, the software's cost is justified. Marketing websites for these products claim to remove the complexity of test automation and replace it with an intuitive user interface.

Make no mistake, test automation is complex. Be wary of products that claim to dramatically simplify the process of creating automated tests. As proprietary test count rises, teams become more tightly coupled to the automation tool. What if you've built hundreds of tests only to discover the tool doesn't deliver on its promises? For cost-conscious organizations, discussions around a rip-and-replace change can be difficult due to the sunk costs involved. Managers concerned about perception are unlikely to admit failure, and teams are left with no choice but to continue painting themselves further into a corner.

It takes courageous leadership to admit there are mistakes in the tool selection process. From a technology standpoint, choosing a third-party testing tool requires fully adopting their architecture. Do you want an enhancement? Submit a feature request and get in line with the rest of the user base. Meanwhile, you're hacking your tests to make them work.

The growing technical debt will lead to widespread test fragility. As fragile tests grow in numbers, the cost of maintaining them becomes unsustainable. If you decide to switch to another test tool, there won't be a graceful exit strategy. It's not going to be as simple as exporting your tests from one tool and importing them into the next.

The Numbers Game

Those unfamiliar with the intrinsic value of test automation attempt to glean that information from a handful of metrics. They place focus on the number of tests written and pay attention to the number of bugs the automated tests find. While those metrics may be interesting, they only tell a small portion of the story.

Teams without test automation pay a recurring cost each time the regression suite is run manually. As your codebase grows, so does your regression suite. As a result, that recurring cost continues to rise. A meaningful metric is "How many person-hours are we saving each week by running the automation suite?"

As the team is able to execute more exploratory testing, they're presumably finding more bugs. How many bugs are found using exploratory testing that wouldn't have been found if the team were running the regression suite manually? It's an interesting question, but it's impossible to answer. It's even more difficult to assign a dollar value that could show ROI. Automation provides value, but in my experience it's difficult to measure.

A team with a reliable safety net is bolder. A product owner may be more than willing to take chances and allow teams to be more adventurous with refactoring code. The resulting refactoring could increase quality. It might even improve the app's user experience, which could increase signups and conversions. More users could result in more revenue. It's impossible to determine ROI using this hypothetical situation. It underscores the point that not everything that has value is easily measurable.

Leaders who focus on the wrong metric incentivize the wrong thing. If a team calculates a key performance indicator as total test count, the team will most likely write more tests. The total test count only indicates the distance traveled since introducing test automation; it doesn't tell you how much closer you are to your destination.

And a high test count doesn't equal better coverage; it just means the team has found a way to game the system.

An example of a meaningful test automation metric is the percentage of the manual regression suite that has been automated. That gives the team an understanding of progress. It also answers the question of how much more effort is required to completely automate manual regression tests.

Not Delivering Value Quickly

Automation is worthless without a properly architected framework. The term framework encompasses all the support code that helps make your tests valuable. Developing a proper framework architecture takes time. It's an adaptive process that evolves with the needs of your team. There are always enhancements to code, components to update, and refactoring to perform.

Due to the pace of software development, it's imperative that the teams writing test automation deliver value as quickly as possible. If your organization has sprint reviews, demo the progress you've made.

Don't wait until your test framework is mature to start writing tests; you can run tests using a walking skeleton of a framework. From a value perspective, the worst thing you can do is spend weeks developing a framework without any tests to go along with it. A framework without tests—no matter how elegant the architecture—provides little to no value. As a result, stakeholders might begin to doubt their investment.

Starting out, your focus should be on automating the manual smoke test. Using a tool like Jenkins, tie the run of these tests to a deployment of code to your test environment. Your next goal should be to automate the full manual regression suite, starting with the high-value components. Features such as signup, conversion, login, billing, and payments are high-value targets and need to be thoroughly tested.

After adopting an automation strategy, commit to delivering at least a handful of tests at the end of your first sprint. Maximize transparency by showing your test results on an information radiator (like a TV mounted on a wall) located somewhere near the teams.

At regular intervals, reflect on your progress and make course corrections to increase the value of your test suite.

Missing the Value Proposition

The goal of any team's automation effort is to create tests that are valuable. Valuable tests are stable, fast, properly focused, and easily maintained. These type of tests are written by teams that have been automating tests for a long time. Over time, teams gain an understanding of what should be automated and what should not. Experienced test teams view automation as a scalpel, not a hammer. They're careful about adding tests to the suite due to increased maintenance costs and extended overall execution time.

When assessing new features to automate, successful teams ask themselves, "Is the juice worth the squeeze?" Is the test being automated actually better left as a manual test? Is the automation too complex and time-consuming to write given the limited time we have in a sprint? Is the feature being developed already saturated with automation? Does the automation's execution rely on the timing of some external moving piece that we cannot control? If the answer to any of these questions is yes, pick something else to automate.

Successful teams are quick to delete tests that don't provide value. They're even quicker to delete tests that fail randomly for no obvious reason. Those tests have a high maintenance cost. Observers of the information radiators might see the tests staying in a failing state a disproportionate amount of time. Eventually your organization will grow to distrust your automated tests and will ultimately discount whatever value they provide as a whole.

Test Automation: A Success

Test automation, just like software development, is not easy. There are significant costs involved. There are many potential wrong turns and false starts. Know that with each valuable automated test you write, your teams become incrementally more liberated.

Most importantly, don't forget to empower your teams. Allow them to self-organize around a vision by providing them a clear strategy. Give them the freedom to explore new technology and decide for themselves which automation tools they want to embrace. Being part of the decision-making process means they'll be more invested in their own success.

Your team might just find the million-dollar bug that would otherwise have remained hidden. How's that for ROI? [{end}](#)

steve@qualitymindset.net



[Click here](#) to read more at StickyMinds.com.

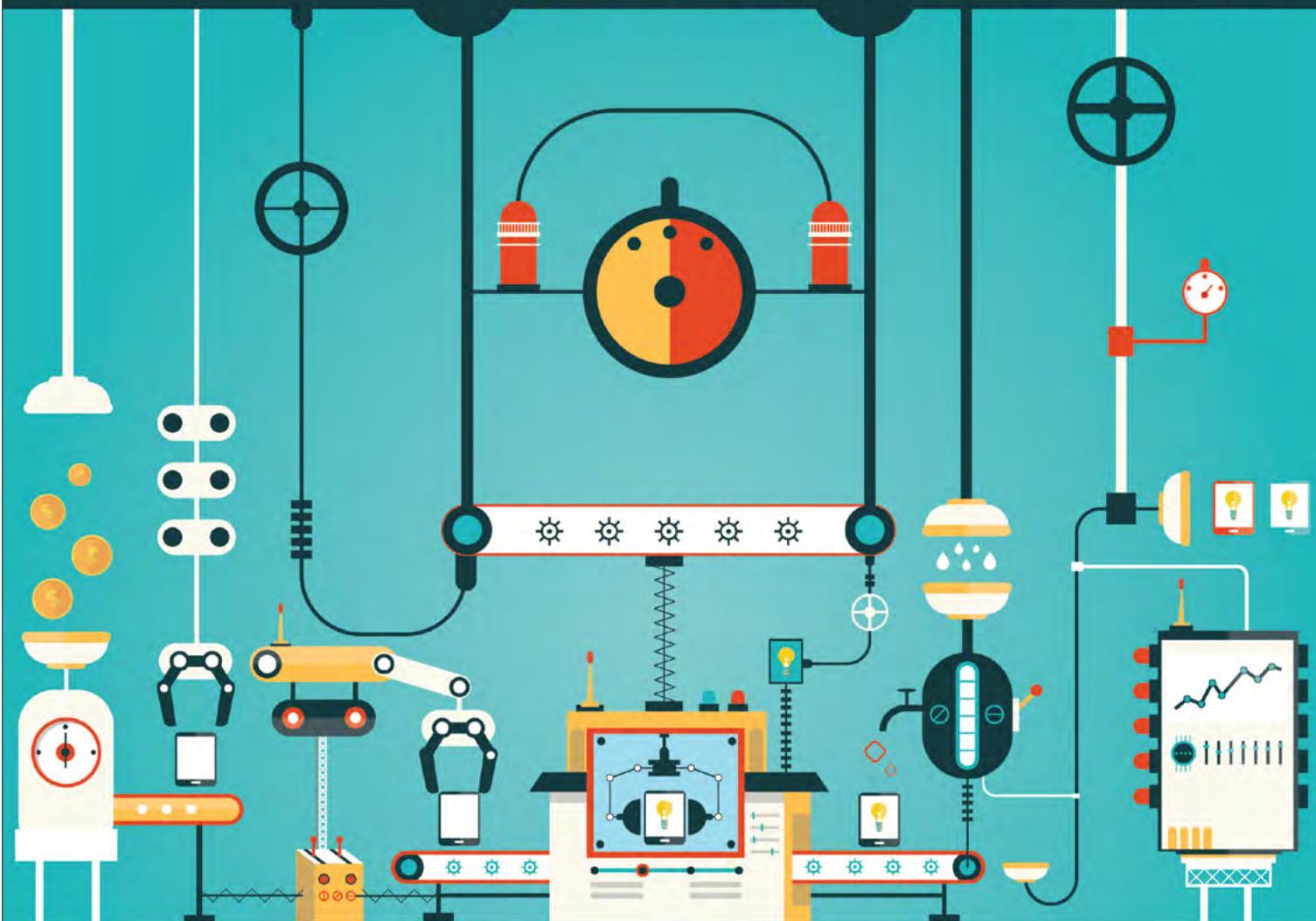
■ References

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 relevant 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 Insights** features the latest stories from conference speakers, SQE Training partners, and other industry voices.
- Visit [StickyMinds.com](#), [AgileConnection.com](#), [CMCrossroads.com](#), or [TechWell.com](#) to sign up for our weekly newsletters.

PEOPLE SHOULD THINK AND MACHINES SHOULD TEST



BY HARRY ROBINSON AND DOUG SZABO

How many tests can you create and execute in an eight-hour workday? Five? Ten? Twenty-five? In a single day, your computer can generate and run tens of billions of tests. That's nine orders of magnitude more testing than humanly possible, all for the cost of electricity to keep the machine going. That's power. That's leverage. And that's why you should have a computer do your testing.

Four Steps to High-Volume Test Automation

We use an approach we call Behavior/Inputs/Outputs/Steering (BIOS) for creating our high-volume automated tests:

1. Describe the desired behavior of the system under test. What is the system supposed to do?
2. Choose how to generate inputs. How can we mechanically generate input values?
3. Design a way for the computer to verify system outputs. How can we mechanically verify results?
4. Steer the test generation toward areas you want to test. How can we focus the testing on areas we think are important?

Example 1: Testing a Square Root Function

Let's apply the BIOS approach to testing a simple square root function.

STEP 1: BEHAVIOR: WHAT IS THE SYSTEM SUPPOSED TO DO?

The square root of a number is a non-negative value that, when multiplied by itself, gives the number. For example, the square root of 16 is 4.

The input domain for this particular function is all floating-point numbers from 0 to 1 billion.

STEP 2: INPUTS: HOW CAN WE MECHANICALLY GENERATE INPUT VALUES?

From those floating-point numbers, our test program selects two different sets of inputs:

1. Proceed systematically through a subset of values (such as 0, 1, 2, 3, ...)
2. Choose sample values throughout the range

STEP 3: OUTPUTS: HOW CAN WE MECHANICALLY VERIFY RESULTS?

Because a square root is a non-negative number that, when multiplied by itself, gives the input value, our test program verifies two outcomes:

1. The output is greater than or equal to zero
2. Squaring the output produces a number very close to the input value (allowing for computer precision)

STEP 4: STEERING: HOW CAN WE FOCUS THE TESTING ON AREAS WE THINK ARE IMPORTANT?

Our highest priority for these tests will be whole numbers, followed by other floating-point inputs. For the moment, we don't care about invalid inputs such as negative numbers or non-numeric strings. Based on our BIOS steps, we design two test programs:

1. Test program A systematically walks through all whole numbers from 0 to 1 billion
2. Test program B randomly samples floating-point numbers between 0 and 1 billion

Test Program A

Because we have only a billion whole numbers to test, we can simply walk through the list. A program based on the pseudocode in figure 1 is straightforward to implement and finishes in about ten seconds on a typical computer.

```
Number = 0 // from the lowest number in range
while (Number <= 1 billion) // to the biggest number in range
{
    Root = SquareRoot(Number)
    if (Root is negative or the difference between Number and Root*Root is too big)
        report an error
    Number = Number + 1 // get the next whole number
}
```

Figure 1: Pseudocode to systematically test whole numbers from 0 to 1 billion

Test Program B

Most programming languages support sixteen digits of precision, so there are about ten quintillion floating-point values between 0 and 1 billion. That's way too many to walk through. Figure 2 shows how to randomly sample values throughout the input range. Because there is no hard stopping point to our input generation, we will let the program run for as long as we don't have anything more important for our computer to do. If randomized inputs make you uneasy, take comfort in knowing that a typical computer can test tens of billions of square roots per day, which is great coverage, no matter how you look at it.

```
while (the computer has time available)
{
    Number = a random floating point number between 0
            and 1 billion
    Root = SquareRoot(Number)
    if (Root is negative or the difference between
        Number and Root*Root is too big)
        report an error
}
```

Figure 2: Pseudocode to test floating-point numbers between 0 and 1 billion

What did we learn from testing the square root function?

- The test programs took only a few minutes to design and write but provided outstanding test coverage because the computer did the grunt work for us for free.
- Generating billions of test cases relieves us of the burden of trying to guess a specific value that might cause the function to show an error.
- We do not store test cases; we generate them as needed. This means we don't maintain a big inventory of static tests.
- We can steer the testing in new directions as required.

Example 2: Testing a Sorting Routine

Now that we've seen the basics of high-volume testing, let's see how it fares in a real-world example.

In 2004, a library of sorting routines called Nsort appeared

on the web. [1] The library was well received and was eventually recommended in the book *Windows Developer Power Tools*. [2]

In 2013, we wondered whether a simple high-volume approach could find useful bugs in these published routines. We tried out high-volume testing on the fifteen sorting routines in the Nsort library, as well as a related implementation from another website. We revisit that experience here, following the four BIOS steps.

STEP 1: BEHAVIOR: WHAT IS THE SYSTEM SUPPOSED TO DO?

Given a set of elements such as {5, 2, 6, 3, 1, 4}, a sorting routine should return the same elements arranged in order (e.g., {1, 2, 3, 4, 5, 6}).

STEP 2: INPUTS: HOW CAN WE MECHANICALLY GENERATE INPUT VALUES?

We can generate sets of various sizes by starting with an ordered set (e.g., {1, 2, 3, 4, 5, 6}) and shuffling that set into a random order, such as {5, 2, 6, 3, 1, 4}.

STEP 3: OUTPUTS: HOW CAN WE MECHANICALLY VERIFY RESULTS?

Our verification step leverages the fact that shuffling a set is easier than sorting it. When the shuffled set created in step 2 is handed to a sort routine, the resulting set should be identical to the original, ordered set.

STEP 4: STEERING: HOW CAN WE FOCUS THE TESTING ON AREAS WE THINK ARE IMPORTANT?

For this example, we are interested in sets of different sizes. We will systematically increment set size from one element to five thousand elements. At each set size, we will perform several shuffles before moving on. We are not currently interested in duplicate elements, and we will limit the sort to increasing order.

Test Program C

Figure 3 shows pseudocode for testing the sort routines. Small sets sort quickly, and it is possible to generate all permutations. Large sets take longer to sort and have many more permutations, so the number of shuffles tested will be limited.

As we kicked off the test runs for the sixteen different sort routines, we wondered whether such a simple approach could find useful bugs. We were soon pleasantly surprised to find significant bugs in four of the routines, as shown in Table 1.

Why You Should Think and Let Your Computer Test

Most test automation today is stuck in the slow lane, with computers rigidly following scripts handcrafted by humans. High-volume automation breaks that mindset, allowing computers and humans to exploit their different strengths. Human ingenuity and computer horsepower can produce cheap, powerful testing that needs to be the next step for the software industry.

Isaac Asimov sounded a similar note thirty years ago in his essay “Intelligences Together” [3]:

“Computers ... are extraordinarily good in some ways. They possess capacious memories, have virtually instant and unfailing recall, and demonstrate the ability to carry through vast numbers of repetitive arithmetical operations without weariness or error...

“The human specialty ... is the ability to see problems as a whole, to grasp solutions through intuition or insight; to see new combinations; to be able to make extraordinarily perceptive and creative guesses.

“Each variety of intelligence has its advantages and, in combination, human intelligence and computer intelligence—each filling in the gaps and compensating for the weaknesses of the other—can advance far more rapidly than either one could alone.”

Asimov’s insight rings true in software testing. No tester can execute a billion test cases by hand, and no computer can think up good test cases by itself. But testers and computers working together form a very powerful team.

We invite you to try out this style of testing yourself. A program to test the ShearSort routine is available for download. [4]

```
setSize = 1;
while (setSize <= 5000)
{
    create an ordered list of size setSize
    for (some number of iterations)
    {
        shuffled list = shuffle(ordered list)
        sorted list = sort(shuffled list)
        if (sorted list != ordered list)
            report an error
    }
    increment setSize
}
```

Figure 3: Pseudocode to test a sorting routine

Sort Routine Name	Example Input	Incorrect Output	Code Issue
FastQuickSort	1, 4, 5, 3, 2, 6	Unhandled Exception	Error decrementing array index
OddEvenTransportSort	2, 3, 1	2, 1, 3	Condition statement used "<" vs "<="
ShearSort	1, 2, 3, 4, 5, 6	1, 2, 3, 5, 4, 6	Condition statement used "<" vs "<="
ShearSortB	9, 2, 7, 4, 5, 6, 3, 8, 1	1, 2, 4, 3, 6, 7, 5, 8, 9	Condition statement used "<" vs "<="

Table 1: Summary of bugs found by automated high-volume testing of sorting routines

Drop the file contents into a Visual Studio Console project and see for yourself how easy and effective this style of testing can be. {end}

harryr@harryrobinson.net
doug.szabo@gmail.com



[Click here](#) to read more at StickyMinds.com.
■ References

WANTED! A FEW GREAT WRITERS!

I am looking for authors interested in getting their thoughts published in *Better Software*, a leading online magazine focused in the software development/IT industry. If you are interested in writing articles on one of the following topics, please contact me directly:

- Testing
- Agile methodology
- Project and people management
- DevOps
- Configuration management

I'm looking forward to hearing from you!

Ken Whitaker

Editor, *Better Software* magazine

kwhitaker@TechWell.com

Featuring fresh news and insightful stories about topics that are important to you, TechWell Insights on TechWell.com is the place to go for what is happening in the software industry today. TechWell Insights' passionate industry professionals curate new stories every weekday to keep you up to date on the latest in development, testing, business analysis, project management, agile, DevOps, and more. The following is a sample of some of the great content you'll find. Visit TechWell.com to read more TechWell Insights.

DevOps Begins with Developers

By Adam Auerbach

The DevOps movement is driving changes to our organizational structures and amount of automation, and accelerating the delivery of high-quality features to our customers. While all this is awesome, there is a ton of work required to make it happen, with a high change curve to overcome. Typically it involves changing methodologies, organizational design, and technologies. But, perhaps the most critical change is to the way the developer works.

Going from waterfall to agile is a change for all team members because it asks you to communicate and work closer as a group. Agile asks everyone to be good team members and start providing more transparency. This can be a little painful for everyone, but for a developer who enjoys the heads-down work of writing code, this is just the beginning of the challenges.

Keep reading <http://well.tc/3wd7>.

A Tester's Guide to Dealing with Scrummerfall

By Bob Galen

If you've been a tester on an agile team, you've probably experienced "Scrummerfall" behavior—a cross between Scrum and waterfall.

You can actually tell in sprint planning. If the developers are planning larger tasks that take nearly the entire sprint to complete, then you're probably in Scrummerfall. You'll hear lots of grousing about the complexity of their work and how it can't be broken down any further.

There isn't really any collaboration. Developers grab their stories and tasks in the beginning of the sprint, and the testers grab theirs. After that, it's every man for himself. If you're lucky, you'll get something to test when the tasks are 80 percent complete. If you're unlucky, you'll need to wait until the very end, which usually butts up against the end of the sprint. The developers consider their part done, then they throw the work over for testing.

Keep reading at <http://well.tc/3wd2>.

Important Questions to Ask Yourself before Committing to Agile

By Josiah Renaudin

Agile isn't a process that can be instituted into your team or organization overnight. This iterative, test-as-you-go methodology is something that requires dedication in order to effectively and successfully enact it, and this sort of fundamental shift can be frightening.

Will my team succeed under agile? Is it worth the time investment to find out? Is agile even right for my company? A head full of indecision is a common occurrence as you inch closer and closer toward a resolution, so in order to lessen this fear and push forward with a clearer mind, you need to ask yourself a few important questions.

Keep reading at <http://well.tc/3wd6>.

"What Is Code?" How I Explain What I Do

By Michael Sowers

You're at a party and someone asks you what you do. You answer that you're a developer, or you're a software tester. "What is that?" the person asks, and you begin what is usually a much longer answer than the listener can bear. Soon, the person interrupts and says, "Hey, I'm thirsty. Let's get a drink!" Anything to get you to stop all the technobabble.

The work we perform is complicated by the objects (modules, methods, applications, systems) that we work on. Yet, given the ubiquitous nature of software and our passion for our profession, it's easy for us to conclude that everyone understands (or should understand) software. However, using our software engineering community lingo to explain what we do gets us "deer in the headlights" looks quickly!

I recently read the Bloomberg article "What Is Code?" It's a long but good article that uses examples and interaction to describe what code is and how it works.

Keep reading at <http://well.tc/3wdu>.

The Challenge of Saying "I Don't Know"

By Naomi Karten

While on a flight, my seatmate asked me what I was reading. I told her, "I don't know," and chuckled because it was the truth. I was reading a book titled *I Don't Know: In Praise of Admitting Ignorance (Except When You Shouldn't)*. What struck me in reading the book is how, according to research, even young children can't "shake the idea that admitting not knowing is bad." Something in our culture or upbringing conveys the message that it's not OK to reveal we don't know something.

Of course, sometimes the reason for this resistance is obvious. When a friend spouts a reference that everyone else seems to know, admitting that you don't can be embarrassing. And when management asks why the project is late, saying you don't know can be a career-limiting response.

The fear of consequences is a powerful motivator, so we pretend we do know or strive to detract others from realizing that we don't.

Keep reading at <http://well.tc/3wdb>.



LIVE
VIRTUAL
TRAINING

ATTEND LIVE, INSTRUCTOR-LED CLASSES VIA YOUR COMPUTER.



Live Virtual Courses:

- » Agile Tester Certification
- » Fundamentals of Agile Certification—ICAgile
- » Testing Under Pressure
- » Performance, Load, and Stress Testing
- » Get Requirements Right the First Time
- » Essential Test Management and Planning
- » Finding Ambiguities in Requirements
- » Mastering Test Automation
- » Agile Test Automation—ICAgile
- » Generating Great Testing Ideas
- » Configuration Management Best Practices
- » Mobile Application Testing
- » and More

Convenient, Cost Effective Training by Industry Experts

Live Virtual Package Includes:

- **Easy course access:** You attend training right from your computer, and communication is handled by a phone conference bridge utilizing Cisco's WebEx technology. That means you can access your training course quickly and easily and participate freely.
- **Live, expert instruction:** See and hear your instructor presenting the course materials and answering your questions in real-time.
- **Valuable course materials:** Our live virtual training uses the same valuable course materials as our classroom training. Students will have direct access to the course materials.
- **Hands-on exercises:** An essential component to any learning experience is applying what you have learned. Using the latest technology, your instructor can provide students with hands-on exercises, group activities, and breakout sessions.
- **Real-time communication:** Communicate real-time directly with the instructor. Ask questions, provide comments, and participate in the class discussions.
- **Peer interaction:** Networking with peers has always been a valuable part of any classroom training. Live virtual training gives you the opportunity to interact with and learn from the other attendees during breakout sessions, course lecture, and Q&A.
- **Convenient schedule:** Course instruction is divided into modules no longer than three hours per day. This schedule makes it easy for you to get the training you need without taking days out of the office and setting aside projects.
- **Small class size:** Live virtual courses are limited to the same small class sizes as our instructor-led training. This provides you with the opportunity for personal interaction with the instructor.

Playing Games to Improve Software

Gamification applies game mechanics to non-game situations and improves user engagement, quality, and engineering processes.

by Ross Smith and Rajini Padmanaban | rosss@microsoft.com and rajini.padmanaban@qainfotech.com

Many dynamic changes have taken place in the world of software development in the past several years. The rise of the Internet and mobile devices—along with the competitive fervor of iterative development—has dramatically changed the world of software development. Engineering processes are being altered in new and exciting ways to meet an exploding demand for software products and services.

We want to explore how the incorporation of game mechanics, or gamification, can help improve software quality and the user experience.

Data Everywhere

While the concept of software quality and user experience seems random and unstructured, there have been many efforts to define approaches for modeling and assessing these concepts. For example, the original ISO/IEC 9126 software engineering product quality standard defined a structured model for software quality and user experience.

The challenge has been around how to balance unstructured user behavior with structured data that comes from an engineering process. Traditionally, dynamic user behavior has been evaluated with inputs from usability study observations or ad hoc user feedback. Today, software and services generate telemetry data that more accurately represents what users actually do. This telemetry supports post-usage analysis, and if it's sufficiently real-time, it can be used to tune the user experience, target advertising, and even suggest friends on social media.

Gamification

In the past few years, the term *gamification* has sprung up to describe the application of game mechanics to non-game situations. Gamification has had a checkered history, with many game developers disliking the term because games have not been effectively built or have often been applied very haphazardly—without mapping outcomes to quality goals. Others feel that companies have misapplied gamification techniques to achieve short-term gains without regard for their users.

If games are being leveraged to bring more users in to evaluate a product and provide feedback, simply awarding a badge for an accomplishment will not bring sustainable results around product quality. When used appropriately, game mechanics can improve user engagement, software quality, and engineering processes. An example is when a communication application is tested via a game to see who communicates with the highest number and greatest diversity of people.

“When a user voluntarily enters the magic circle of a software experience, game mechanics can be applied to direct unstructured future behavior.”

Structured and Unstructured Data

Structured data is highly organized data that resides in fixed fields in a record, file or database. This is typically data found in relational databases, with normalized tables that can be searched and joined easily. This leads to important issues about structured data: what fields and how it is stored, such as data type (date, numeric, string) and restrictions on the data (number of characters). Unstructured data is data with little or

no degree of organization that does not reside in a traditional row-column database. It is hard to compile or draw conclusions from unstructured data. Unstructured data often include text and multimedia content, such as emails, word processing documents, photos, videos, and webpages.

Solid engineering practices and user experience methods rely firmly on structured data. Engineering teams look to performance lab data, defect density, code coverage, and static analysis tool results that are organized to describe software's quality. Structured online service telemetry data allows large-scale sites to tune user experience, target advertising, encourage friend referrals, and keep users engaged.

The challenge is that user behavior—the activity or task that comes next—tends to be unstructured. If your test case is “add a contact, then immediately make a call,” you can't direct a Skype user to do this in order to serve your quality goals. User behavior is unpredictable. Likewise, software developers cannot instruct their beta users to install a new version and try new features that generate telemetry data to refine the service.

Games and Play

In his book *Homo Ludens*, Dutch philosopher Johan Huizinga talks about the “magic circle of play.” [1] This is a place outside the ordinary world that is entered voluntarily with new rules that turn normal activity into something playful. In his book *The Grasshopper*, Bernard Suits presents a great example using the game of golf. [2] The goal of golf is to start at the tee and put the ball in the hole for eighteen holes. Outside the magic circle, the easiest way to meet this goal is to pick up the ball, walk from the tee, and drop it in the hole. But inside the magic circle, players add artificial obstacles: They wear plaid pants and use golf clubs to hit the ball from a tee. With new rules, golf becomes fun—and for some, even addictive.

How Gamification Relates to Structured Data

When a user voluntarily enters the magic circle of a software experience, game mechanics can be applied to direct unstructured future behavior. Games have rules that can guide a user’s unstructured behavior toward the activity that generates structured data. This data is then used to improve software quality or provide a more engaging user experience.

Let’s say, a word building game is used to test for validations of text that a user would provide in a form. When left without any rules, you may not be able to bring structure to the data that is coming in. However, if your game has rules around how many times a certain character can be entered, what kinds of capitalization to use, or what special characters are allowed, the user would structure his inputs based on the rules to be able to proceed further with his play. This will make the data more structured and more fun for the user who is mapping his inputs to game rules. This is magic circle thinking!

Conclusion

Gamification and the use of structured data have a strong connection. Structured data is critical to assessing all phases of software development, release, and user engagement. Agile and iterative development processes require big data to inform decisions about where to invest. To know what to build or improve on to keep users engaged, services look at structured data from A/B testing, deployment rings, and experiments. The challenge is that user behavior is unstructured. Service providers and development teams can’t tell users what to do next.

However, games and play offer an alternative. A game is fun because it is voluntary and outside our normal world. When players enter the magic circle of play, they have a new level of engagement and will follow rules to be successful. When we bring together the world of play and the world of software engineering, we can leverage game mechanics, gamification, and solid game design. This magic circle thinking provides a unique framework for software developers, online service providers, and customers to generate structured data from unstructured behavior in a way that can help improve the quality of the user experience.

Delivering a wonderful, intuitive, and engaging experience to our users is really what building great software is all about. In a highly competitive world, it is important to build an edge (and even some unexpected magic) into our software that can shape the industry for the coming years. Gamified data collection can help us provide those magical experiences. {end}



[Click here](#) to read more at StickyMinds.com.

■ References

index to advertisers

Agile Dev, Better Software & DevOps West	http://adc-bsc-west.techwell.com	9
ASTQB	http://www.astqb.org/map	8
Mobile Dev + Test & IoT Dev + Test	http://mobile-iot-devtest.techwell.com	16
Parasoft	http://www.parasoft.com/bettersoftware SV	23
Ranorex	http://www.ranorex.com	2
Sauce Labs	http://saucelabs.com	11
SOASTA	http://soasta.io/cloudtestnow	Back cover
SQE Live Virtual Training	http://www.sqetraining.com/training/delivery-options/live-virtual	35
SQE Training	http://www.sqetraining.com/trainingweek	24
STAREAST 2016	http://stareast.techwell.com	Inside front cover

Display Advertising

advertisingsales@TechWell.com

All Other Inquiries

info@bettersoftware.com

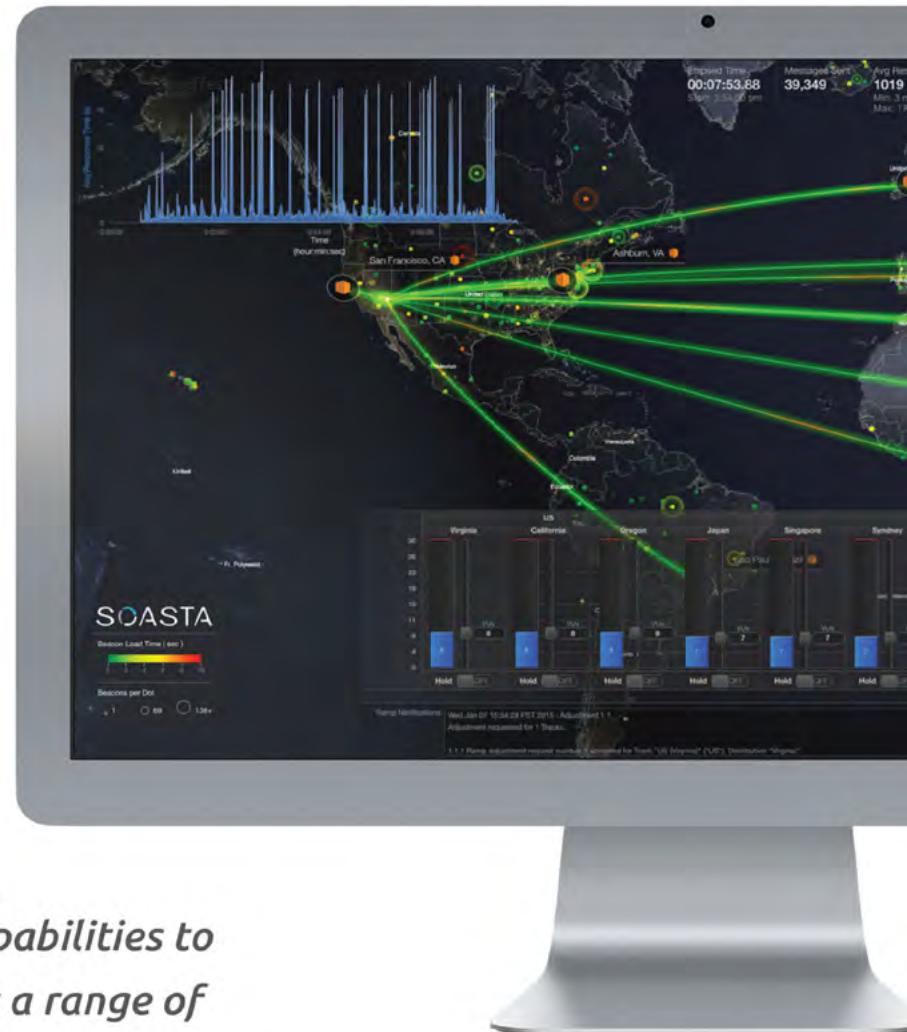
Better Software (ISSN: 1553-1929) is published four times per year: January, April, July, and October. Print copies can be purchased from MagCloud (<http://www.magcloud.com/>) user/bettersoftware). Entire contents © 2016 by TechWell Corporation, (350 Corporate Way, Suite 400, Orange Park, FL 32073), unless otherwise noted on specific articles. The opinions expressed within the articles and contents herein do not necessarily express those of the publisher (TechWell Corporation). All rights reserved. No material in this publication may be reproduced in any form without permission. Reprints of individual articles available. Call 904.278.0524 for details.

In today's online world, milliseconds mean \$millions.

Are Your Apps Performing at Their Peak?



- **Monitor** real user behavior & business metrics in real-time
- **Test** continuously across web & mobile applications to ensure availability
- **Optimize** via powerful predictive analytics & data visualization for fastest resolution



"SOASTA, with its unique capabilities to correlate information across a range of metrics and excellent real-time dashboard technology, enables us to spot issues, make smarter decisions and align our efforts to achieve the best possible performance and business outcomes."

