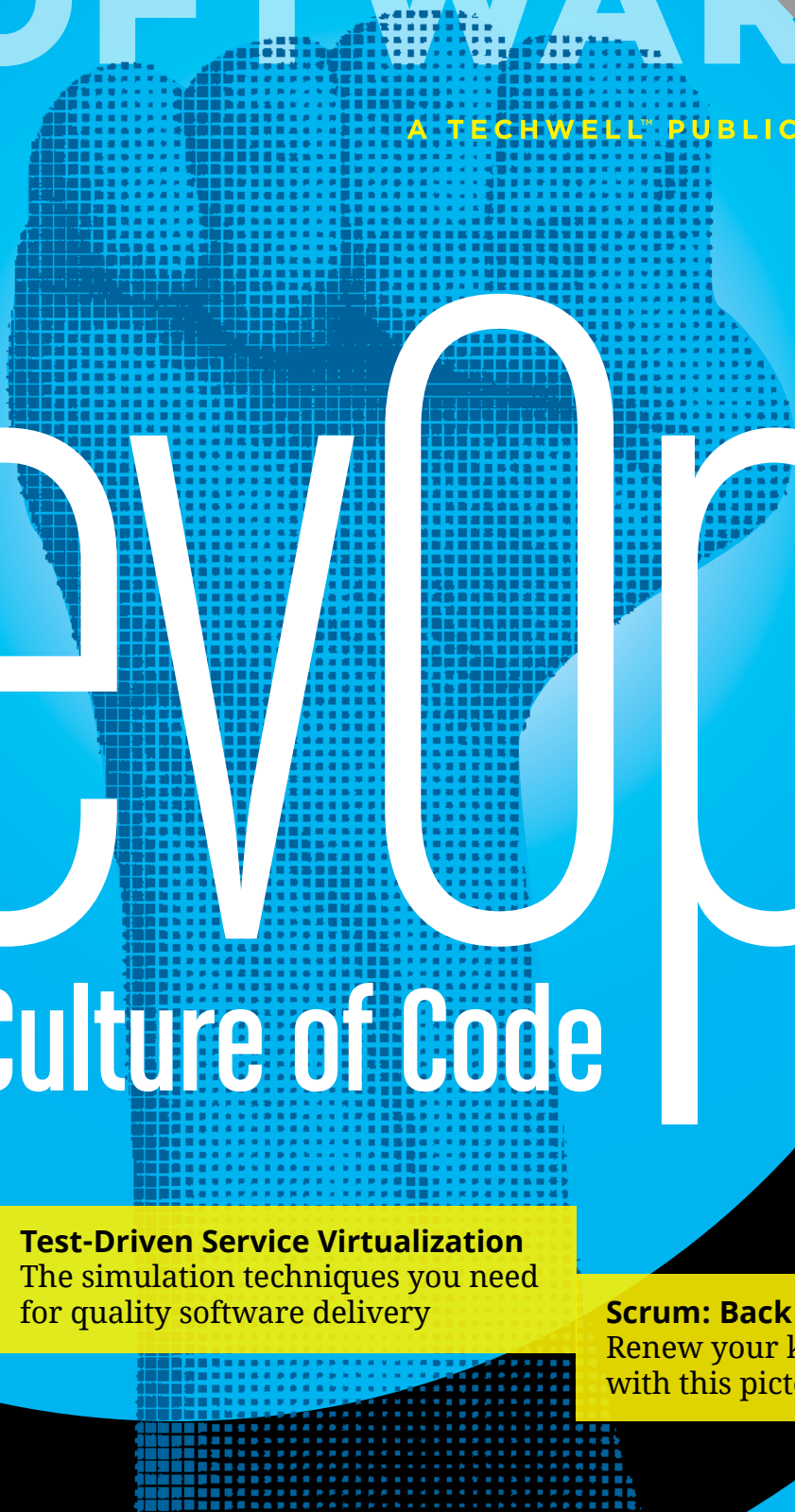
SPRING 2018

[BETTER SOFTWARE]™

A TECHWELL™ PUBLICATION



DevOps and the Culture of Code

Plus

Test-Driven Service Virtualization
The simulation techniques you need for quality software delivery

Scrum: Back to Basics
Renew your knowledge of Scrum with this pictorial "walkabout"

QMETRY

Digital Quality Platform

Jira

TEST MANAGEMENT

TEST AUTOMATION

QUALITY ANALYTICS

Deliver High Quality Software Faster

Test Management that Scales from Manual to Agile & DevOps

Intelligent Test Automation with CI/CD integration for Continuous Quality

AI enabled Real Time Actionable Intelligence for Competitive Edge

Share the Success Of
100,000+ Happy Users!

TRY IT FREE

What Are Past Attendees Saying?

about the TOPICS

"I very much enjoyed being able to cross-attend the varying topics to gain a large content of ideas."

Tim Robert, Systems Analyst, State Farm

about the KEYNOTES

"The keynotes were inspiring! There were several practical talks. Gave me time to think and network to develop actionable takeaways."

Pete Lichtenwalner, Sr. Engineer Manager, Verint

about the SPEAKERS

"Great speakers that show they are passionate about what they do. Plus they are open to share ideas and experiences."

Verita Sorsby, QA Manager, Tio Networks

about the TUTORIALS

"Excellent conference. The tutorials were invaluable to me and my group."

Jennifer Winkelmann, Business Analyst, TD Ameritrade

Agile Dev topic areas:

- Scaled Agile Development
- Agile Testing
- Agile Implementation
- Career & Personal Development
- Agile Teams & Leadership
- And More

Better Software topic areas:

- Digital Transformation
- Process & Metrics
- Software Quality & Testing
- Requirements & User Stories
- Project Management
- And More

DevOps topic areas:

- Architecture & Design
- Configuration Management
- DevOps and Test/QA
- Continuous Delivery
- Continuous Integration
- And More

Agile Dev Better Software DevOps WEST

A TECHWELL EVENT

JUNE 3–8, 2018
LAS VEGAS, NV
CAESARS PALACE

Register by May 4, 2018, with code CWBSM to save up to $600 off your conference*

TO REGISTER, CALL 888.268.8770 | BSCWEST.TECHWELL.COM

*Discount valid on packages over $400

AGILE TESTING DAYS USA

SAVE THE DATE FOR THE INAUGURAL EVENT!

JUNE 25–29, 2018
BOSTON, MA

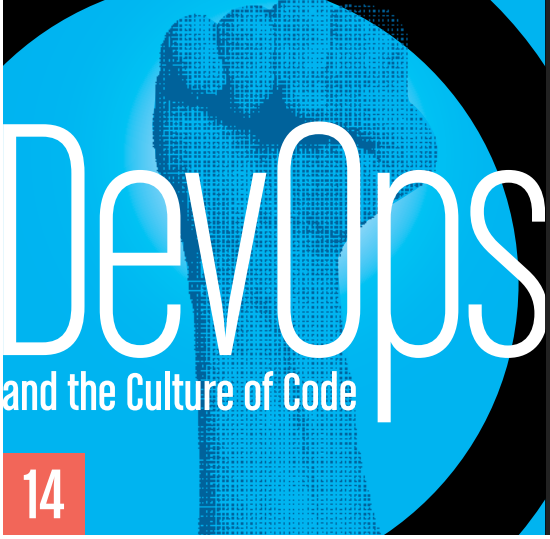
Europe's fun and widely popular agile testing festival is coming to North America as Agile Testing Days USA. This event will feature over 50 of the top agile testing enthusiasts speaking in Boston.

https://agiletestingdays.us

SUPER EARLY LOBSTER SAVINGS WHEN YOU REGISTER BY APRIL 27, 2018

INSIDE

Volume 20, Issue 2
SPRING 2018



14

On the Cover

DevOps and the Culture of Code

Migrating an organization to continuous integration requires adoption new processes, tools, and automation. DevOps relies on dramatic culture change to encourage total transparency and collaboration among all project stakeholders.
by Patrick Turner

Features

19

Test-Driven Service Virtualization

Because enterprise applications are highly interconnected, development in stages puts a strain on the implementation and execution of automated testing. Service virtualization can be introduced to validate work in progress while reducing the dependencies on components and third-party technologies still under development. *by Alexander Mohr*

26

Scrum: Back to Basics

So you think you know Scrum? Using the whimsical notion of farm animals and light-hearted visuals, take a refreshing review of the entire Scrum lifecycle as an intuitive set of roles, responsibilities, and handoffs. Particular attention is placed on what the ScrumMaster and product owner are expected to do at each handoff. *by Brian Rabon*

Columns

09 TECHNICALLY SPEAKING

Building a Test Automation Strategy

QA departments always feel the pressure to start testing quickly, even if the ever-changing software being tested isn't ready. A bought-in test automation strategy can keep a project on track.
by Justin Rohrman

34 THE LAST WORD

The Unspoken Truth about IoT Test Automation

The internet of things (IoT) continues to proliferate as connected smart devices become critical for individuals and businesses. Even with test automation, performing comprehensive testing can be quite a challenge. *by Rama R. Anem*

Departments

SQE TRAINING
A TECHWELL COMPANY

Helping organizations worldwide improve their skills, practices, and knowledge in software development and testing.

Software Tester Certification—Foundation Level
http://www.sqetraining.com/certification

April 9–13, 2018
Virtual classroom

April 29–May 1, 2018
Orlando, FL

May 1–3, 2018
Philadelphia, PA

May 15–17, 2018
San Diego, CA

May 15–17, 2018
Denver, CO

June 5–7, 2018
Nashville, TN

June 5–7, 2018
Jersey City, NJ

June 19–21, 2018
Austin, TX

DevOps Week
https://www.sqetraining.com/training-events/devops-week

April 16–20, 2018
Washington, DC

Lean Kanban Week
https://www.sqetraining.com/training-events/lean-kanban-week

May 7–11, 2018
Seattle, WA

TECHWELL™ events

Conferences
Cutting-edge concepts, practical solutions, and today's most relevant topics. TechWell brings you face to face with the best speakers, networking, and ideas.

STAR EAST
A TECHWELL EVENT
Apr. 29–May 4, 2018
Orlando, FL
LEARN MORE

Agile Dev Better Software DevOps WEST
A TECHWELL EVENT
June 3–8, 2018
Las Vegas, NV
LEARN MORE

AGILE TESTING DAYS USA
June 25–29, 2018
Boston, MA
LEARN MORE

STAR WEST
A TECHWELL EVENT
Sep. 30–Oct. 5, 2018
Anaheim, CA
LEARN MORE

STAR CANADA
A TECHWELL EVENT
Oct. 14–19, 2018
Toronto, ON
LEARN MORE

Agile Dev Better Software DevOps EAST
A TECHWELL EVENT
Nov. 4–9, 2018
Orlando, FL
LEARN MORE

EDITOR'S NOTE

Coworking Communities Are Changing How Software Is Developed

Ken Whitaker
kwhitaker@techwell.com
Twitter: @Software_Maniac

The traditional office workspace is a dinosaur. Software professionals want a different work-life balance. They want the freedom to work from home but still have the experience of working in a team setting. Small-town tech hubs and coworking are the future. Steve Case, cofounder of the Washington, DC-based venture capital firm Revolution LLC, started a crusade with Rise of the Rest®, a nationwide traveling initiative to support and promote entrepreneurs in emerging startup ecosystems.

As Josh Dorfman, the director of entrepreneurship at Venture Asheville and managing director at Asheville Angels, told me "One of the greatest values of coworking is serendipity. It is very likely that you will be sitting down next to others who can benefit your business. I see it all of the time." J.D. Claridge, the CEO of software-powered drone maker xCraft, embraces coworking at the Innovation Collective located in Coeur d'Alene, Idaho. It isn't just because of the amazing espresso bar and the central presentation conference center. "The coworking office environment gives us flexibility," Claridge said. "We use it as a hub to manage a diverse team throughout the US and with contractors in other parts of the world." Living and working where we want to be is a dream come true.

Now, I'd like to switch gears and present the content of the Spring 2018 issue of *Better Software*. Our featured cover article is Patrick Turner's "DevOps and the Culture of Code," which emphasizes the importance of organizational culture required to embrace continuous DevOps operations.

Service virtualization is taking off, and testing requires special considerations. Alexander Mohr walks you through the entire testing lifecycle in "Test-Driven Service Virtualization." After years leading agile teams, I thought I knew Scrum—until I read Brian Rabon's "Scrum: Back to Basics." And if you're interested in test automation, Justin Rohrman gives great advice in "Building a Test Automation Strategy" and Rama Anem shows how intelligent devices need to be tested in "The Unspoken Truth about IoT Test Automation."

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 we enjoy working with these wonderful authors. Don't forget to spread the word and let people know about TechWell and *Better Software* magazine.

BETTER SOFTWARE™
A TECHWELL PUBLICATION

FOLLOW US

PUBLISHER
TechWell Corporation

PRESIDENT/CEO
Alison Wade

DIRECTOR OF PUBLISHING
Heather Shanholtzer

Editorial

***BETTER SOFTWARE* EDITOR**
Ken Whitaker

ONLINE EDITORS
Josiah Renaudin
Beth Romanik

PRODUCTION COORDINATOR
Donna Handforth

Design

CREATIVE DIRECTOR
Jamie Borders
jborders.com

Advertising

SALES CONSULTANTS
Daryll Paiva
Kim Trott

PRODUCTION COORDINATOR
Alex Dinney

Marketing

MARKETING MANAGERS
Cristy Bird

MARKETING ASSISTANT
Allison Scholz

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, Ste. 400
Orange Park, FL 32073

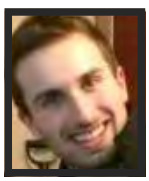
TECHWELL™

CONTRIBUTORS

Rama R. Anem has more than thirteen years of testing experience and is a test architect at Sunpower Corporation. At AMD, she managed the enterprise-wide data platform quality practice and was responsible for building its first center of excellence. At IBM, Rama lead multiple teams and served as one of the brand ambassadors for DB2. In addition to writing articles in a variety of technical journals, she also presents advanced topics at international software conferences and workshops. Rama can be reached at rama.anem@gmail.com.

Starting as a developer, **Alexander Mohr** has been working in software development since 2000 in a variety of roles. Since then he has worked as a product owner, business analyst, test manager, architect, and project manager. Recently, Alexander introduced test-driven service virtualization for a large Austrian telecommunication provider as a groundbreaker for "shift left." Now, he works for Tricentis as an evangelist for service virtualization and agile transformation. Alexander can be contacted at a.mohr@tricentis.com.

Brian Rabon is the president of The Braintrust Consulting Group, a worldwide leader in agile transformations. Throughout his seventeen years of IT industry experience, Brian has applied agile methods in order to successfully deliver working products to his customers. When not in the classroom, Brian can be found around the globe evangelizing the benefits of agility. Brian is the author of *Scrum for the Rest Of Us* and an avid blogger. To contact Brian, please email him at brian.rabon@braintrustgroup.com or join him on LinkedIn.



A longtime freelancer in the tech industry, **Josiah Renaudin** is now a web-content producer and writer for TechWell Insights, StickyMinds.com, and *Better Software* magazine. He wrote for popular video game journalism websites like GameSpot, IGN, and Paste Magazine and now acts as an editor for an indie project published by Sony Santa Monica. 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.

Justin Rohrman has been a professional software tester since 2005. In addition to being technical editor of StickyMinds.com, Justin is a consulting software tester and writer working with Excelon Development. He also serves on the Association for Software Testing board of directors. As president, Justin helps facilitate and develop programs like BBST, WHOSE, and the CAST conference. Contact Justin at rohrmanj@gmail.com.

Patrick Turner discovered a passion for computers at the age of ten on a trip to a local Radio Shack with his brother. He has more than twenty years of experience producing software solutions for a broad range of business needs. In addition to being an accomplished public speaker, Patrick is the CTO of Small Footprint and oversees a unique model of blending experienced American software development professionals with talented Romanian software engineers. Patrick can be reached at pturner@smallfootprint.com.

Building a Test Automation Strategy

TEST AUTOMATION IS OFTEN VIEWED AS THE PANACEA THAT SOLVES ALL QUALITY ISSUES. BUT LIKE OTHER TESTING, TEST AUTOMATION REQUIRES A STRATEGIC PLAN.

by Justin Rohrman | *rohrmanj@gmail.com*

It is the end of a sprint and we still don't know much about the overall quality of the product. For the past decade, the popular approach to this problem has been to batch up the test ideas we created in this sprint, add some from the last few releases, and perform them all. Ideally, we find a bunch of surprising problems that were introduced over the course of the past sprint. This process, known as regression testing, usually takes a few days to perform. From a manager's perspective, the only good thing to do with regression testing is to make it take less time.

I have been employed at more than one company where this scenario leads to a group of testers—usually with little to no programming experience—attempting to build a UI automation solution. After a year of development, they have a build that took three hours to complete and tests that passed or failed unpredictably. Understandably, management is shocked and disappointed when the automation project ends up as a mess.

For most projects, testing with code is better done by the people who are writing the production code.

When testing legacy software, UI automation may make perfect sense as the basis of your automation strategy. [1] However, in most cases, you can find information about your product quality easier and faster by exploring other parts of your product first. Let's take a look at why that is.

Code Design and Feedback

Testers think of automation in terms of testing. Because developers do the vast majority of testing with code, they may have a slightly different take.

Developers are in the business of taking something that works at the moment and then adding a bunch of changes that introduce risk. Whenever code changes, there is potential for something new to go wrong in surprising parts of the product. Developers want quick feedback to know if their code is doing what they expect and to know if they broke anything that was already working.

Some developers will create small tests that run a bit of code to simply check a value. In an ideal world, a developer makes a change and then runs a test. This results in receiving feedback from that test within seconds. If the test fails, they check their code. If the test passes, they can refactor the code to make it perform faster, be more readable, or conform to company code standards.

For most projects, testing with code is better done by the people who are writing the production code. Developers have the context for what will help with their code and are usually motivated to take action when a test fails. They know what parts of the code base have coverage and what needs to be tested.

Providing Automation in Layers

The company I am working with now has its automation spread out over several layers in the product.

At the base is a set of tests built in RSpec that work as unit tests. These check granular things in the code like the status on a checkbox when a page loads and the value of a variable after a calculation is performed. Tests at this level create a nearly instant feedback loop that helps with code design and refactoring, which helps a developer know when they are done with a code change.

On top of that, we have a suite of tests built in Jasmine that validates the UI. These test slightly larger amounts of code and ask questions: Does this button become enabled under certain conditions? Does this page refresh to display a confirmation message after we save?

Another level up from testing UI code are ChromeDriver

tests that run through the Cucumber behavior-driven development framework. These tests check even larger pieces of code and create a feedback loop around simple but important scenarios that a user might perform. These are things that absolutely must work for our software to be useful.

Given more time and resources, we want to add more testing at the service layer. This would be a great place to test a lot of data permutations very quickly.

I often hear from testers that automation makes the computer perform repetitive work so that testers can work on more important, exploratory work. I don't agree. Automation requires a specific skill set and is usually a time-consuming and expensive activity. In my experience, automation helps developers deliver high-quality builds the first time. As an example, automation catches those obscure code issues like field length or null text entry conditions. Layering tests against the unit level, services, and the UI will produce a much better build and make bugs harder to find.

When a tester gets a build to test from development, they should work hard to find problems instead of spending all their time on finding the easy, "low-hanging fruit" defects.

Where Do I Start?

If test automation can help a problem in your development process, the first step is to figure out how to start. My recommendation would be to focus as close to the code as possible, most likely with unit tests at the service layer. Testing at this level is generally fast and relatively easy to create and perform, but it also gives the development team fast feedback about changes they are making in the code. Once that is in place, then and only then, start looking at how to automate testing the UI.

There is usually an optimal way to test each layer in your product. Take a look at your product and your staff. The question of where to start automating should be easy to answer. [BSM]

CLICK FOR THIS STORY'S REFERENCES


GET
INSPIRED
AT THE
PREMIER EVENT
FOR
SOFTWARE TESTING
PROFESSIONALS
STAR WEST
A TECHWELL EVENT
SEPT. 30-OCT. 5, 2018
ANAHEIM, CA
CLICK TO LEARN MORE

How Manual Testers Are Evolving into Automation Engineers

Jennifer Scandariato

Years in Industry: **24**

Twitter: **@JScan**

Interviewed by: **Josiah Renaudin**

Email: **jrenaudin@techwell.com**

"Everyone is driving toward efficiencies and automation with methods such as CI/CD where you test early and often, to reduce the number of defects found late in the software delivery lifecycle."

"Last year when I spoke about the need to upskill testing abilities, there was an attendee in the audience who voiced a concern that if they did that, then perhaps that person would leave his company. So, there is a fear about attrition."

"There is a bias that manual testing is something that provides less value. I believe a strong engineer knows that there's a level of critical thinking that is required to 'break' things and ensure they are built robustly."

"On a great team, you have diverse skillsets where everyone complements each other, and you might even alternate roles such as paired programming where one person is developing and the other performs the validation or peer review."

"I've met with various leaders who think automation is the magic pill, but it's not. How you automate and how much you automate is the key."

"The idea of high performing and best-in-class software as a service (SaaS) is focused for everyone in an agile team and not just the testing arm."

"My belief is that the entire team is accountable for the quality, not just the QA engineer, test engineer, or SDET."

CLICK HERE FOR THE FULL INTERVIEW

LIVE VIRTUAL TRAINING

LEARN ANYWHERE! LIVE, INSTRUCTOR-LED PROFESSIONAL TRAINING COURSES

Live Virtual Courses:

» Agile Tester Certification
» Software Tester Certification—Foundation Level
» Fundamentals of Agile Certification—ICAgile
» Foundations of DevOps—ICAgile Certification
» Performance, Load, and Stress Testing
» Mastering Business Analysis
» Essential Test Management and Planning
» Finding Ambiguities in Requirements
» Mastering Test Automation
» Agile Test Automation—ICAgile
» Generating Great Testing Ideas
» Exploratory Testing in Practice
» Mobile Application Testing
» and More

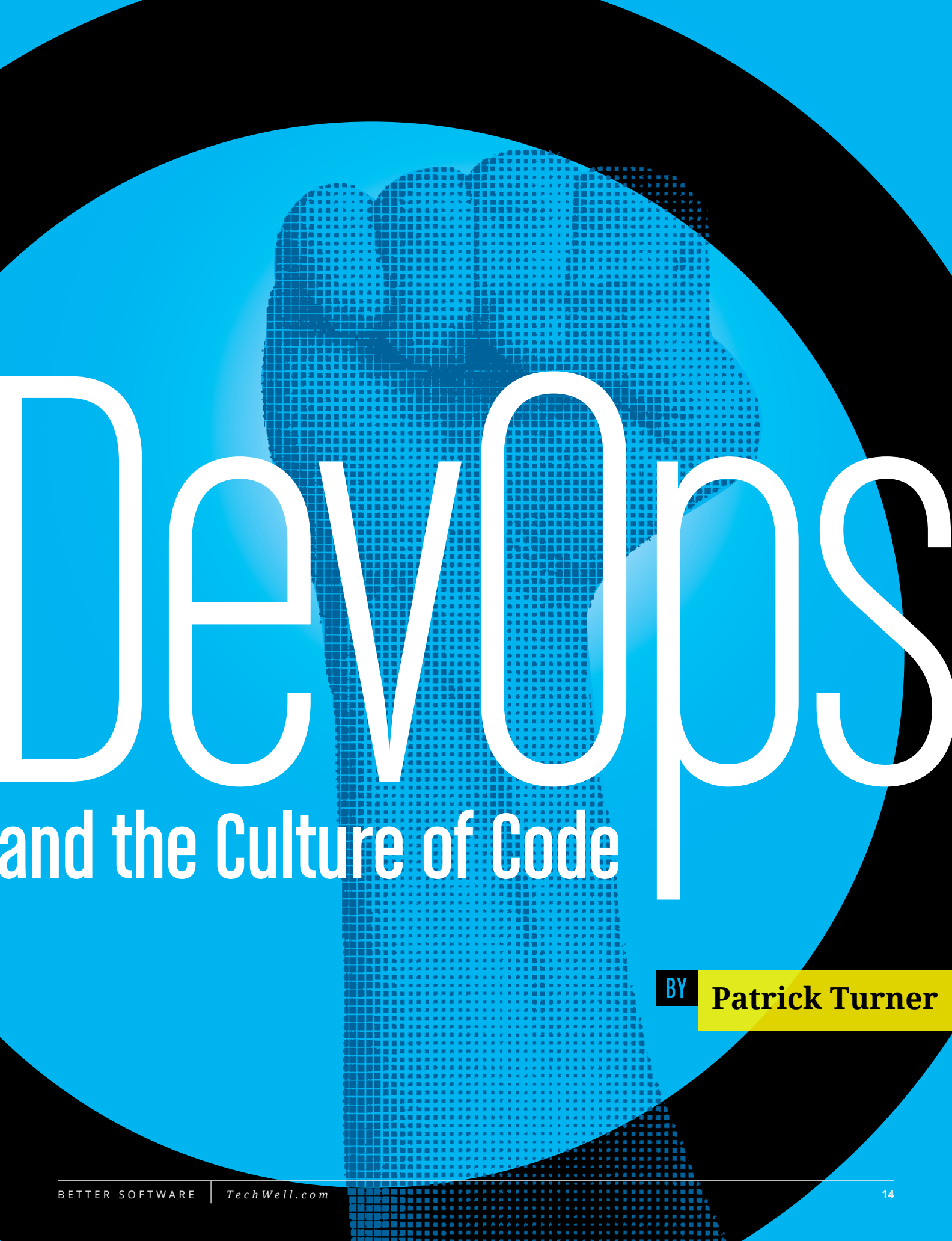
Convenient, Cost Effective Training by Industry Experts

Live Virtual Package Includes:

- **Easy course access:** Attend training right from your computer and easily connect your audio via computer or phone. Easy and quick access fits today's working style and eliminates expensive travel and long days in the classroom.
- **Live, expert instruction:** Instructors are sought-after practitioners, highly-experienced in the industry who deliver a professional learning experience in real-time.
- **Valuable course materials:** Courses cover the same professional content as our classroom training, and students have direct access to valuable materials.
- **Rich virtual learning environment:** A variety of tools are built in to the learning platform to engage learners through dynamic delivery and to facilitate a multi-directional flow of information.
- **Hands-on exercises:** An essential component to any learning experience is applying what you have learned. Using the latest technology, your instructor can provide 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 four hours per day. This schedule makes it easy 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 in small class size to ensure an opportunity for personal interaction.

SQETRAINING.COM/LIVE-VIRTUAL

SQE TRAINING
A TECHWELL COMPANY



DevOps

and the Culture of Code

BY **Patrick Turner**

Over the past twenty-five years, the software development industry has seen a lot of change. But not since the '90s and the prevalence of the web have we seen such a significant shift in our world. Agile changed how the core development team works together, improving collaboration and having a distinctive impact on organizational culture.

DevOps is extending the same cultural shift to the rest of the software development lifecycle and the rest of the organization. It is my experience that DevOps leverages tools and processes to enable teams to row in the same direction more naturally.

Transparency is naturally built in when teams fully implement DevOps best practices. This results in opening up a team's ability to innovate by creating an environment where the feedback cycle and sharing of ideas is transparent, fast, and expected. Everyone takes on the responsibility of quality, as teams are fueled by positive reinforcement rather than cover-ups and catching errors.

We know culture can determine the success of any initiative, but it may hit home more to hear that culture can determine the failure of any initiative, too. DevOps tackles this problem through process, tools, and building a viable team value system.

Extending the Agile Culture Shift

Many technology companies talk about transparency—a fancy term for everyone knowing what's going on. It sounds simple, but getting past organizational inefficiencies and poor communication can be a huge challenge. If you've ever said, "Well, it worked on my machine ..." then you understand how vital collaboration is and how working in silos can damage culture and yield lackluster results. When you work in a silo, company culture suffers because people don't know what's going on and they go numb to the collective goal protecting their own assets.

DevOps is about implementing tools, but the biggest opportunity is the culture shift that occurs when people simply know what's going on. Using traditional software development methods, development doesn't know what to build and can't articulate what to test or how to deploy. QA doesn't know what to test or where to test it, and operations isn't given instructions on how to deploy the software when it's ready. It doesn't take long before the finger-pointing starts. Successful DevOps implementation aligns the culture so collaboration can happen.

As developers, our goal is to deliver the best possible software. Agile taught the development team to work better together, and if you've implemented agile, your team should already be talking to each other and taking ownership of their work.

Similarly, the secret to implementing DevOps is to realize it's not just about tools. It's really more about building a culture of DevOps as an extension of what we've learned with agile by eliminating traditional inefficiencies to streamline the communication and collaboration in an organization.

DevOps integrates and automates processes, creating transparency so people can build the best possible software. Agile allowed for the integration of engineering and quality into the software development process. The next step is for DevOps to help extend that agile culture shift upstream to the product and downstream to the operations organization.

To start, think about how DevOps changes development. With automated builds and continuous integration, developers now must focus on unit tests where code tests code. In fact, unit test coverage is critical to a successful DevOps implementation. This in itself becomes a cultural challenge because developers have grown accustomed to throwing a build over the wall to QA and letting them find the problem. Achieving great unit test coverage requires developers to take a more direct role in ensuring the quality of the software product.

DevOps is about implementing tools, but the biggest opportunity is the culture shift that occurs when people simply know what's going on.

Source Code Management Is the Foundation

Before feature branching, all the developers in a group worked on the same code set. Keeping track of what was ready to go live and keeping the code organized was tough. The communication challenges presented a real problem, especially across distributed teams. Branching source code was common, but the entire development team typically worked in the same branch. As a result, toes were being stepped on regularly and the blame game culture ensued.

With feature branching, development teams can decide which features (or user stories) make sense to the group, keeping them distinct so that go-live decisions become easier to make. It also helps create a self-organizing culture for the code and the teams so they take ownership of their areas. This approach allows the team to move quickly and independently without having to consult with everyone in every part of the development organization before they make decisions.

It usually takes a while for team members to really understand why they should invest time in developing branching strategies as

part of their overall development plan. Just because you've moved your code into Git doesn't necessarily mean that you're solving source code management problems. It's important to spend time on good branching strategies up front and throughout the course of the project, as it makes for cleaner builds. Ultimately, the result is easier deployment and less finger-pointing. This represents a real cultural impact.

In the past, a bug fix meant making a quick—and usually ugly—decision about how to integrate that hot fix into the code and deciding how to make it go live. With feature branching, the process becomes a lot easier by keeping unique parts of the coding process segregated, which ensures that handling even hot fixes becomes less stressful for the team members. It also ensures only clean code makes it to the production environment. This approach helps achieve the goal of delivering high-quality software to end-users.

Using planned branching strategies on a distributed source code repository creates a culture of transparency. It helps ensure that everyone knows what features are going to be in a specific product release. The team's comfort level increases with better odds that only clean code is going into production.

The Role of QA, from Software Breaker to Value Maker

With DevOps, automation enables QA to add more value to the process. It actually changes the QA role and improves the impact they have on the software.

Traditionally, a tester was just a power user—maybe someone from customer service who was pretty good at breaking the application. As a result, testers spent forty hours a week just trying to break the application. They were going through screen after screen and manually hunting down bugs.

These days, however, testing is becoming a more automated process, with test scripts and automated tests able to run end-to-end testing without human intervention. In addition to finding bugs, automation can also be used to test performance and security by letting the system do all of the grunt work of the traditional manual tester.

The automation around DevOps, such as automated regression testing and performance testing, allows QA to focus more on testing new features, which is where they add the most value to the project. QA can spend more time communicating quality feedback with project managers, product owners, and end-users. When the team builds software that users actually want, the customer benefits.

Ultimately, the culture shift is toward QA becoming a true part of the value proposition, not just the folks responsible for breaking the software. With DevOps, QA has the opportunity to find more ways to add value by focusing on other areas, such as user experience acceptance testing.

Continuous Integration and Continuous Delivery Are a Must

When most people think about DevOps, they usually focus on deployment. We see deployment as an activity that happens at multiple points in the process, not just when you are going live. Along with improved collaboration and communication, it's key to the shift to transparency.

The use of continuous integration involves producing a clean build every time new code is ready for testing. The process, simply stated, grabs the code from the source code repository and compiles it into a build ready for testing. That process itself may break the build as it is running unit tests, integration tests, automated tests, and security tests during the build process, so that by the time it gets to QA, it's ready to go. The cultural mindset needs to be focused on delivering code that is always ready for QA and stakeholder review. It eliminates the "it worked on my machine" excuse.

When you incorporate continuous delivery (builds that are automatically deployed to targets), QA can actually test the next build without having to wait on a developer or questioning what's in the build. It's tied to their test case management tool, which is tied to user stories in the practice management tool.

Continuous integration and continuous delivery are yet another way to improve an organization's culture. It eliminates excuses between developers, QA, and system administrators by improving transparency and communication. Everyone knows what's being delivered and where the build will be deployed.

The cultural mindset needs to be focused on delivering code that is always ready for QA and stakeholder review.

Defining the Operations Environment

When it comes to operations, DevOps has a huge impact. Infrastructure-as-code, where an environment is described as code, and containers, which bundle all of an application's dependencies into a package, allow a software architect and system administrator to define the server environment collaboratively. They ensure predictability so that environments remain exactly the same for development, QA, stage, and production. As a result, an application will function the same in every environment, every time. Once you've spent the time writing those initial scripts, one of the beneficial side effects includes the simplification of onboarding new developers.

Orchestration is an approach to systematically managing production environments, including load balancing (which automatically does real-time horizontal scaling), spinning up production instances with new releases, and bringing down instances with old releases in a controlled way. Sophisticated orchestration can even be used to spread your environments across multiple services simultaneously, such as your local data center and design or other target environments.

The beauty of this is that if the server or an entire environment fails, the system can graciously recover. Tools can be implemented that even bid automatically, in real time, for the most cost-effective environment. Rolling out new versions, which is key, via methods such as feature switching and canary staging (a version of software that has not been tested) can be handled automatically.

In a DevOps world, you don't want one person holding everything together. System administrators have to give up some control, and developers have to take on new responsibilities. As a result, tensions between development and operations decrease, eliminating the culture of finger-pointing.

In addition to orchestration, real-time monitoring is critical. Not only should the traditional system administration role be involved, but a DevOps culture should provide visibility to every stakeholder. Proper orchestration ensures that environments are running, and monitoring should be set at many levels, from the database tables to text verification. The DevOps movement also has resulted in many more sophisticated monitoring tools becoming available.

Ultimately, your goal should be to identify problems before users even know they've occurred, while at the same time ensuring all stakeholders (the product team, advisors, and customers) are all in the loop.

Building the Best Culture

Focus on what you hope to achieve with a better DevOps culture. Should it produce happier, more productive teams? Faster and more frequent delivery? Do you want higher quality releases, better communication and collaboration, or higher employee engagement?

Each of these goals has value. Happy, more productive teams create better software. Faster, higher quality releases are certainly a goal, because no one wants customers to see a bug. Finally, more frequent delivery provides value faster, creating a shorter feedback loop from stakeholders on whether the team is doing the right things.

Finally, DevOps means a serious shift in implementation of tools. A good DevOps team will support the development organization, not drive it. But, ultimately, I'd encourage you to view those tools as a support mechanism for implementing values of positive culture and collaboration. DevOps tools are vehicles for practicing transparency and responsibility so that you can harvest real innovation from your team. Keep those values at the heart of your true goals in delivering the best software, and the tools will help you get there. [BSM]

pturner@smallfootprint.com

TRAIN YOUR TEAM ON YOUR TURF

ON-SITE ADVANTAGE

BRING THE TRAINING TO YOU

Agile Test Automation
Fundamentals of Agile
Foundations of DevOps
DevOps Leadership Workshop
Software Tester Certification
and More!

80+ ON-SITE COURSES

- **25+** AGILE & DEVOPS COURSES
- **40+** TESTING COURSES
- **8** REQUIREMENTS & BUSINESS ANALYSIS COURSES
- **8** DEV & TESTING TOOLS COURSES
- **8** PROJECT MANAGEMENT COURSES
- **7** TEST AUTOMATION COURSES

For more than twenty-five years, TechWell has helped thousands of organizations reach their goal of producing high-value and high-quality software. As part of TechWell's top-ranked lineup of expert resources for software professionals, SQE Training's On-Site training offers your team the kind of change that can only come from working one-on-one with a seasoned expert. We are the industry's best resource to help organizations meet their software testing, development, management, and requirements training needs.

With On-Site training, we handle it all—bringing the instructor and the course to you. Delivering one of our 80+ courses at your location allows you to tailor the experience to the specific needs of your organization and expand the number of people that can be trained. You and your team can focus on the most relevant material, discuss proprietary issues with complete confidentiality, and ensure everyone is on the same page when implementing new practices and processes.

IF YOU HAVE 6 OR MORE TO TRAIN, CONSIDER ON-SITE TRAINING

SQETRAINING.COM/ON-SITE

SQE TRAINING
A TECHWELL COMPANY

TEST-DRIVEN SERVICE VIRTUALIZATION

ALEXANDER MOHR

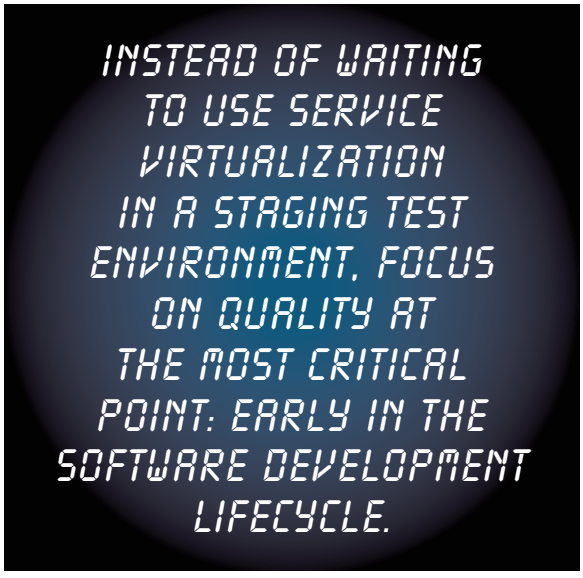
Service virtualization just celebrated its tenth anniversary. The technology is still certainly not mainstream, but maybe it should be. Service virtualization is a simulation technology that lets you execute tests even when the application under test (AUT) and its dependent system components cannot be properly accessed or configured for testing. By simulating these dependencies, tests will encounter the appropriate dependency behavior every time they are executed.

Service virtualization is typically used if dependent system components are unreliable, evolving, unstable, not yet developed, delayed in delivery, challenging to provision (or configure) for testing, or have unsettled scope. It's also used in cases where consistent test data is too complex to generate. In these cases, service virtualization helps avoid delays and overcome blockers in test. But it needs to be positioned more wisely and introduced early not just as a reaction to a symptom. By that time, it is too late.

Staging and Service Virtualization

In larger enterprises, it's still fairly common to have waterfall-like software delivery processes where release cycles last weeks or months. Because applications are now so highly interconnected, it is hard to obtain valuable, quality feedback unless you can interact with an application's dependent systems. Usually, this interaction is achieved through a test lab.

Once the AUT is ready for testing, operations deploys the applications and required dependent subsystems to the test environments. These are protected by gate acceptance criteria and supervised by release management. Testing can start only when all required applications are available and properly configured. However, getting the many moving pieces properly aligned at the same time can be quite a challenge.



One proposed solution is to use environment-based service virtualization to work around any constraints. With this approach, delayed applications are supposed to be simulated until the applications are actually available. In the case of dependencies that are too unstable for testing, service virtualization jumps in as needed in a fallback mode. Service virtualization is also used to reduce interactions with third-party applications that are expensive to access for testing. In practice, I typically see the required simulation artifacts set up by dedicated service virtualization teams that are embedded in the software delivery process right from the planning phase on.

Sounds good, right? But it hadn't been working so well for one telecom provider that approached me for help with service virtualization.

One hundred seventy-five testers verified twenty-five core applications and a total of two hundred applications in three partly mounted test environments. Normally, each system integration and end-to-end test case clearly defines the scope of applications and verified business cases. What the team did not consider was that a simulation fallback temporarily diminishes the testing scope. Tests that ran against the simulation had to be identified and rerun against the dependent system once it was stabilized. The joy of having unblocked test cases was quickly marred by the additional effort required to set the test suite right.

Moreover, predefined service virtualization scenarios required the team to align on decision criteria such as time windows, test data segregation, and source applications. All of this ultimately reduced the available testing time. Using service virtualization as fallback (and without knowing the exact definition of the test drivers) required the team to accurately mirror real application data—in the form of an initial data load and ongoing data sync to ensure that consistent test data was always available. The team gained three days in a four-week test period, but this didn't offset the added costs and efforts. As a result, they considered the service virtualization initiative a failure to that point.

This is one of numerous examples I've seen where this approach to service virtualization hasn't significantly reduced the team's time to market, the testing resources required, or the need to provision and manage test environments. I believe this is because the approach is too little, too late. Instead of waiting to use service virtualization in a staging test environment, focus on quality at the most critical point: early in the software development lifecycle.

To be fair, there are some valid use cases for this approach. If you have an easy way to load, persist, and sync data changes, it can be a viable approach for simulating the behavior of applications that are completely out of the testing scope. However, in reality, the time required for coordination and orchestration makes it hard to reach the break-even point where the value is worth the effort.

Stubs offer another approach to simulation—one on the other side of the spectrum. Stubs allow developers to mimic the interaction of two applications and reduce dependencies from the perspective of their personal desktop. However, those stubs deliver more of a frame than a full request/response structure.

For example, I recently interacted with a team working on the checkout procedure of a shopping cart for a food retailer. The stub provided a successful response from the payment provider, but not the other variants (such as identifying the credit card payment

type, managing an invalid date or CVV, and declined transactions). Guess what blocked the integration tests? A defect in handling declined transactions. It took another two weeks and numerous resources to fix the defect, deliver a new package, align with release management to get the test environment updated, run the gate criteria tests to install the package, have operations installing the new versions, and rerun the tests.

This is just one example of how stubbing tends to provide a false sense of security without really closing the quality gap in a way that's valuable for today's applications and teams.

The Benefits of a Test-Driven Approach

On the other hand, acceptance test-driven development (ATDD), behavior-driven development (BDD), and test-driven development (TDD) have all proven to be magnificent drivers for quality and efficiency. Following TDD principles, developers code unit tests first, followed by classes, functions, and procedures.

Having an automated test case available at the very beginning of your software process provides an efficient foundation for software delivery. In addition, BDD and ATDD expose all possible initial situations, conditions, and expected results for each requirement or user story. By thinking through the process from beginning to end, you ensure that everyone on the team understands what's expected, minimizing unwanted surprises and maximizing the artifact's quality through early availability of test drivers.

Test-driven service virtualization combines the power and benefits of BDD, ATDD, and TDD with service virtualization.

Let's assume that our AUT is an online store. Any arbitrary test driver (UI for the web front-end, API for the online shop middle layer, and even batch files) can be used to validate the AUT. This driver would trigger a process that invokes several service requests on dependent third-party components. These invoke a CRM for customer data, a credit rating, and an order processing system to orchestrate the order once the user submits it.

Test-driven service virtualization combines test variants from the input side with outbound third-party service calls. It uses identical data combinations, allowing testers to execute a fully simulated integration test against the app. These tests cover necessary business variants such as an acceptable or unacceptable credit rating, a declined credit card transaction, or additional shipment fees.

Here are the steps:

1. Identify the initial situation, conditions, and expected results.
2. Combine your test cases as a test driver against your AUT while using service virtualization to simulate dependent applications' interfaces (via request and response scenarios).
3. Following TDD principles, create and provide the test driver and service virtualization artifacts as soon as development starts coding—establishing a full, simulated unit test, system test, and sandboxed integration test for your AUT from the early phases of the delivery cycle.

These steps are essentially the same as those used for many years in aircraft and Formula 1 simulators: Identify the right behavior, then set up tasks by simulating outside conditions. Nevertheless, pilots still get trained on real airplanes, and Formula 1 drivers still drive an incredibly fast two-minute lap at over 200 miles per hour with the real car on a real racing track. It's the same in IT: We execute a fast and efficient end-to-end test to prove the functionality of relevant business processes, and we also test the real-world interface interaction to identify any anomalies that might not have been covered or specified in the simulation.

Getting Started with Test-Driven Service Virtualization

For "greenfield projects" that aren't restricted by constraints of prior work, service virtualization scenarios are set up by using example files or by interpreting definition formats (like OpenAPI, RAML, oData, or WSDL). An alternative option is to intercept the network traffic, record the service traffic, and provide those recordings (in combination with your executed test case) as a service virtualization scenario.

This recording option is a perfectly efficient way to get started for "brownfield projects" that are based on legacy applications. In this situation, recording re-

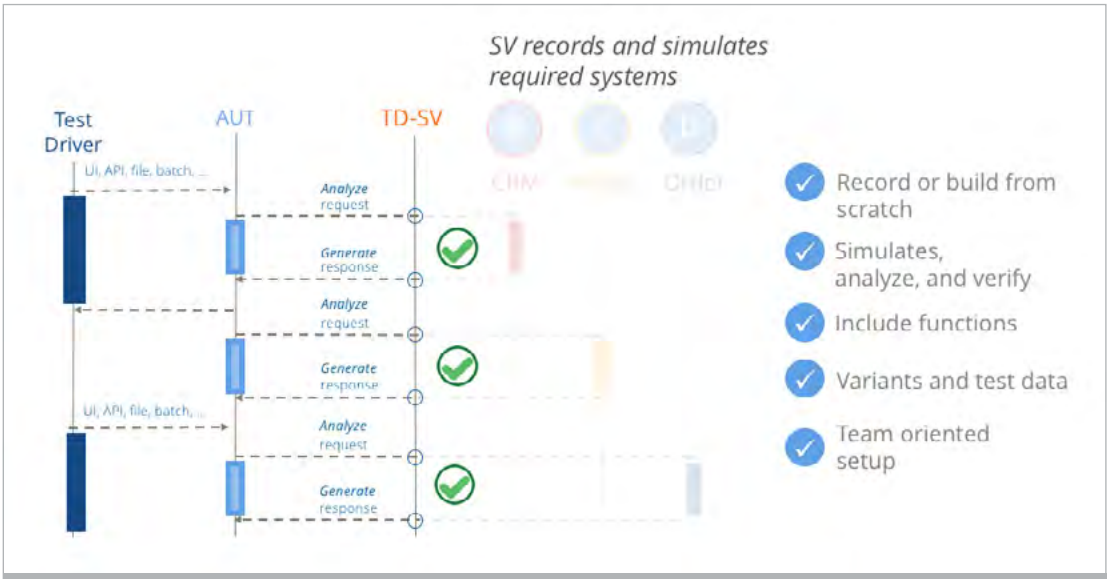


Figure 1: Test-driven service virtualization approach

verse-engineers a process when documentation is not available. With service recording, the quality of the results depends on the test drivers that are selected. This may require considerable research to identify the input variants that lead to specific outputs from an AUT.

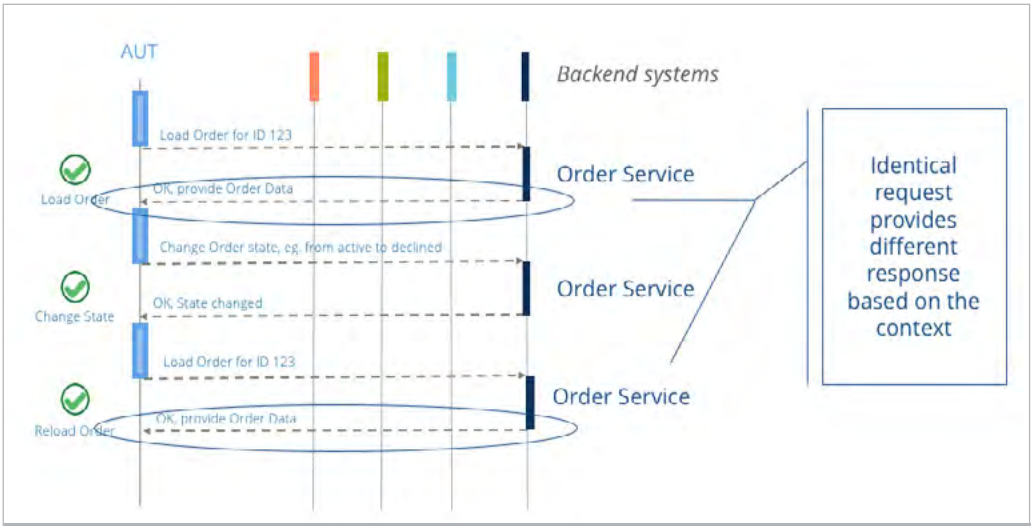


Figure 2: How stateful scenario simulation works

Once the AUT's schema and request data are verified, data can be stored, reused, or modified for later service virtualization scenario steps. It is possible to embed string, mathematical, and external functions to make test scenarios more realistic. Advanced testers use dynamic pattern matching systems to provide different response scenarios, structures, or failure simulations.

Stateful scenario simulation is a test process against an AUT. It requires an order to be created, then that order is either reused or declined.

The simulated back-end system's response varies based on where you are in the process. If the service request says, "Give me data for the specific order 123 after it has been created," it responds with an order state of "active." After the order has been confirmed (or declined due to a negative rating), it responds to the same request ("Give me data for the specific order 123"). This results in a final state of either confirmed or declined. The same request leads to different responses based on the process phase. If the AUT doesn't persist that order data, the identical data can be used over and over again, relieving you from creating additional test data in dependent third-party applications.

When using sandbox testing with stateful scenario simulation, test-driven service virtualization provides the AUT modified data during the test process, without persisting real data changes in third-party components. Assume a test driver that needs to add a new order, process the order, approve the order, and add components to the order. The AUT retrieves appropriate test data during the test process, but in a way that's fully independent of any prior or ongoing data preparation and modification in external components.

The test data repository either holds test data variants or creates appropriate test data on demand, as shown in figure 3.

Test data include technical data mappings in the AUT's database, e.g., customer-related data (1). The specific data record "John Peter Doe, United States" (2) is used to execute tests against the application under test via the UI (3). Next, the outbound service request (4) includes "John Peter Doe, US." The service virtualization framework "knows" which record to expect, verifies against "John Peter Doe, US" (Did the AUT send the correct data?) and responds with the required customer detail data. The test driver can then verify this and proceed.

In combination with a dynamic data management repository that is used consistently by test driver and service virtualization scenarios, test-driven service virtualization doubles its efficiency while keeping data in sync. This ensures that the test driver and its simulation scenario use identical data, which reduces the need for data preparation in external components.

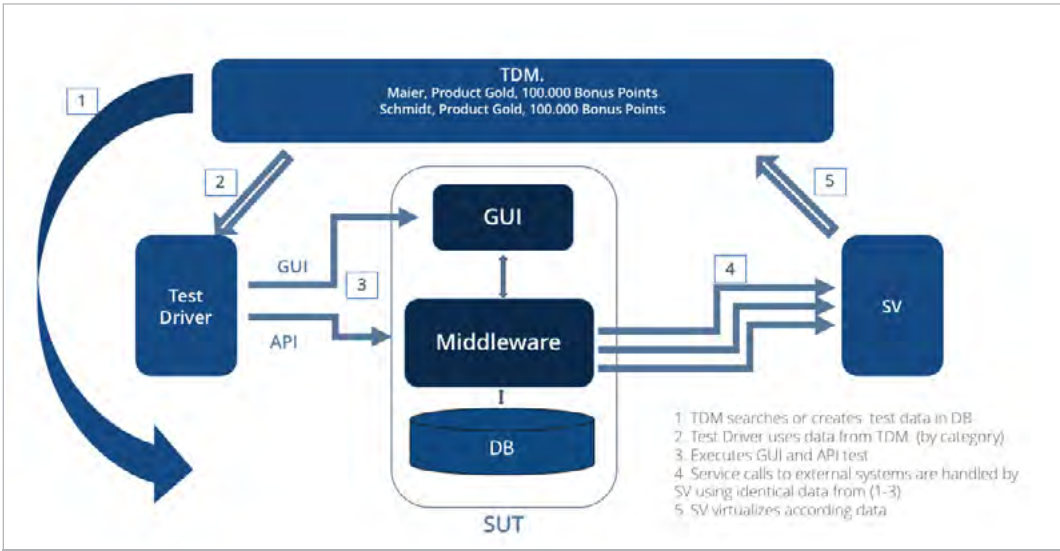


Figure 3: Combining a test data repository with a test driver and test-driven service virtualization

The Scope of Service Virtualization

Once you start with service virtualization, you will have to determine what data and service variants to use for the simulation and decide if the service or consumer side defines the scope. One simple request-response pair of your PurchaseOrder-Service could be too little, but a full data copy of 56 million records of your CRM application would skyrocket your setup and maintenance costs through the roof.

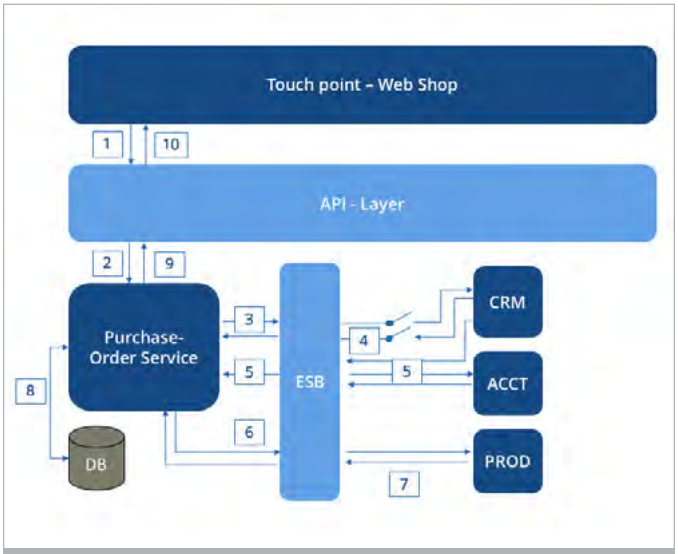


Figure 4: A simple microservice with a web store connected to an API-Gateway

With test-driven service virtualization, your test cases define the scope. As a result, it does not matter if you start with a server-side approach. In one case, the service provider sets up the service virtualization variants, then makes them centrally available and reusable throughout the company and to external partners. In other cases, the service consumer creates selected variants based on its usage, then makes the artifacts centrally available and reusable for consumers—and, at the same time, extendable and maintainable by consumers or service providers.

Let's assume the flow shown in figure 4.

A simple microservice environment consisting of a web store connected to an API-Gateway forwards service requests and responses via http to an internal purchase order service. The internal services are connected via a lightweight enterprise service bus (ESB). The ESB provides partly asynchronous messaging via queuing. The full order process requires an account service, a product service, and an available CRM service. The CRM is basically protected by a circuit breaker, which responds with an error to the client system if the CRM is not available. Once the order has been created, the order service sends status and results back to the web store.

1. A user enters their data, address, product, and options in a web application, then the data is sent to an order service via RESTful service (1, 2).
2. The purchase order triggers internal services via RabbitMQ, JSON/AMQP to CRM to create or retrieve customer data (3, 4), and the CRM service is protected via a circuit breaker that returns an error if the CRM becomes unstable. An asynchronous account creation is triggered, then the account service notifies the order service event based via RabbitMQ (5).
3. The order service retrieves product data (6, 7), verifies all data, and stores the created order in its database (8).
4. The result is handed back (9, 10) to the touchpoint.

The goal here is to set up a stateful simulation scenario to correlate input and output parameters, which get verified inside the order service.

To achieve this, start by defining the initial situation, conditions, and results (local or international customers, consumer or business service). Next, identify the possible conditions that are valid options, as well as those that trigger error conditions (such as an unavailable CRM service). Finally, identify the outcomes correlated to successful and unsuccessful orders. For example, an unsuccessful order might be correlated with a message such as "Sorry, something went wrong. We apologize for the inconvenience. Please try again."

Sandbox testing, another important test technique, allows the order-service team to retrieve full integration test results on a daily basis (figure 5).

The AUT is encapsulated in a test-driven approach using test drivers and test-driven service virtualization artifacts simulating dependent applications, both of which use the same test data repository.

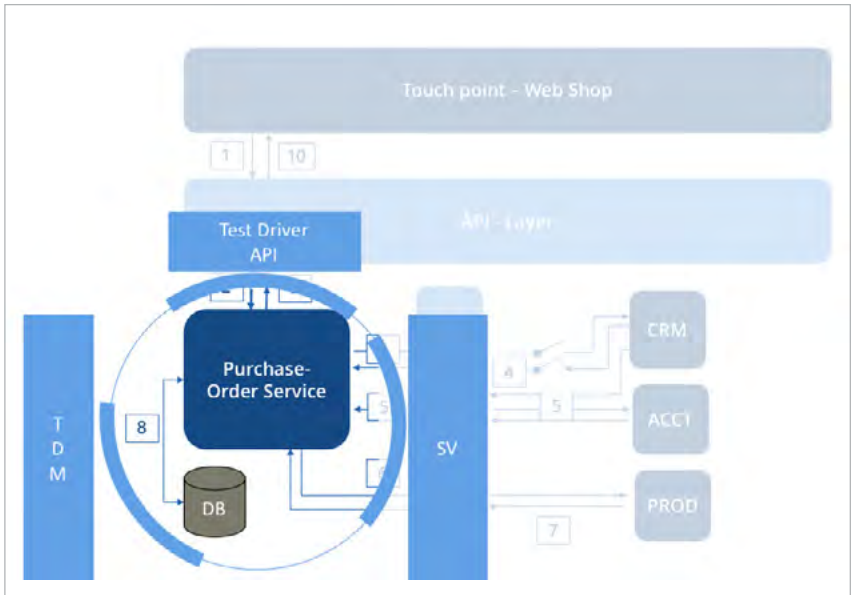


Figure 5: How sandbox testing works

The API definition is loaded for the API purchase order service (2) and combined with the data repository. The three following data definitions for CRM, PROD, and ACC are used to create a stateful service virtualization scenario: CRM request and responses (3, 4), asynchronous account creation (5), and product request and responses (6, 7). The scenarios are applied with the test data repository, which allows you to verify the outbound order requests for the right input data and defines the right JSON response variants for national, abroad, or active circuit breaker (error condition). The identical data repository is applied to the purchase order response (9) to verify the result.

Creating those test scenarios right from the beginning allows an efficient and fully simulated integration test from the very first line of code.

TEST-DRIVEN SERVICE VIRTUALIZATION IS A DIFFERENT APPROACH THAT APPLIES SIMULATION PRIMARILY AS AN ENABLER FOR GREATER TEST COVERAGE AND IMPROVING COLLABORATION ACROSS AGILE TEAMS.

The Benefits of Test-Driven Service Virtualization

As a project manager in the enterprise telecom marketplace, I've seen firsthand how applying this strategy reduces the cost of fixing defects and quality assurance. There are also other benefits, including controlling project risks, lowering hardware costs, and reducing license costs.

On several projects, we've measured a 25 percent reduction in the overall development effort, a 45 percent reduction in testing time, and up to a 65 percent reduction of the operational and overall management costs that typically take place with staging test processes. In my experience, once service virtualization has been established at a broader scale, it reduces hardware and license costs up to an additional 40 percent.

For example, I set up a test-driven service virtualization approach for a shopping application that had seven forms and eleven back-end service calls. It took the team less than three hours to create the automated UI test driver and only thirty minutes to create the service virtualization scenario. Once that foundation was set, test cases could be run completely decoupled from any of its dependencies. As a result, the team had full quality feedback at their fingertips every single day during development. Otherwise, they would have had quality feedback three to four weeks later, that feedback would have been based just a single execution, and they would have needed additional resources to set up the staging environments. Considering the reduced wait time as well as the benefits of early defect detection, they estimate that this approach helped them get the project delivered twenty times faster.

Is Test-Driven Service Virtualization for You?

Service virtualization has predominantly been used as a patch to prevent blockers in staging test environments. This approach is indeed viable in specific scenarios, but it's typically applied too late in the software process to deliver the promised benefits.

Test-driven service virtualization is a different approach that applies simulation primarily as an enabler for greater test coverage and improving collaboration across agile teams. With this focus, service virtualization assets are developed and used from the earliest phases of the software delivery cycle. It provides developers an always-on virtual test environment and simulates sandboxed integration testing.

Although there is a learning curve involved in getting up to speed with test-driven service virtualization, the time to value exceeds that of traditional approaches to service virtualization, and the rapid quality feedback is not able to be estimated. [BSM]

a.mohr@tricentis.com


MARK YOUR
CALENDAR
STARCANADA
IS COMING
OCT. 14-19
WWW.WELL.TC/STARC18

SCRUM: BACK TO BASICS

BY BRIAN RABON

After pioneering the agile leadership movement and putting on many Scrum workshops, I have noticed a trend. Many of the workshop attendees have never experienced agile, and one of my tasks has been to present how the agile movement positively influences leadership. To my delight, there is something insightful about re-examining the basics—things we often take for granted. This is especially true for any Scrum practitioner who has been using advanced techniques for a while.

Regardless of whether you are an agile veteran or someone who is just hearing about Scrum for the first time, this back-to-basics article explains what Scrum is and why you should be using it to build better software.

An Introduction to Agile

Probably the best definition of Scrum I have heard is "Scrum is a fun and profitable way to get work done." Essentially, Scrum is a team-based approach to building a product. Before I get into the fundamentals of Scrum, let's examine why it is considered one of the best examples of any agile process.

Agile, as applied to an alternative way of developing software, was coined in February of 2001. Frustrated with the prevailing software development paradigms, a team of seventeen software development experts met and hashed out what is now known as the Agile Manifesto. [1]

The Agile Manifesto defines what agile software development should be all about. This includes collaboration, accepting change, face-to-face interaction, and the benefit of a functional product over predefined plans, documentation, or rigid processes. Agile is sometimes referred to as a lightweight process because of its minimalistic approach, but this doesn't mean that we don't plan or create documentation. Although agility can be tailored to meet the needs of any work effort, truly agile teams never compromise its core values.

The primary features of agile support frequent and rapid change. Instead of defining the detailed end-game for a product prior to starting work, agile focuses on setting a vision and getting started with incremental development of small pieces of functionality. Thus, a client or customer can change their minds along the way with minimal disruption or loss of work already performed on the product. Agile is an empirical process (think inspect and adapt), whereas traditional methods are based on a defined process (a plan-driven approach).

Critics believe that agile ignores good design principles and process. Proponents, on the other hand, believe that you get exactly the product customers need with incremental development, inspection, adaptation, and review.

Agile itself is not a discipline or a set of practices. Rather, it is a philosophy for iterative and incremental product development. The implementations of agile promote teamwork, collaboration, and adaptability throughout the lifecycle of product development. Some of the popular agile frameworks are Scrum, Extreme Programming, kanban/lean, Crystal, and feature-driven development.

Scrum is an agile method emphasizing the values and principles of the Agile Manifesto, with a focus on commitment, focus, openness, respect, and courage. As Scrum is a way to get work done, much of its tenets are based on the concept of "Keep it simple, stupid" (KISS).

The approach relies on a few concepts while letting an organization fill in the gaps. According to a VersionOne survey, Scrum is the most popular of the agile approaches, with 58 percent of respondents who use agile reporting that they practice Scrum or a Scrum–Extreme Programming hybrid. [2]

The Pigs and the Chickens

Who would have imagined that farm animals would be used to describe Scrum projects? The analogy goes like this: A pig and a chicken are planning to open a restaurant together, but they can't decide on a name. The chicken wants to call it Ham and Eggs, but the pig has concerns. The rub is the pig would be committed, but the chicken would only be involved (figure 1).

There are key differences between being *committed* and being *involved*.

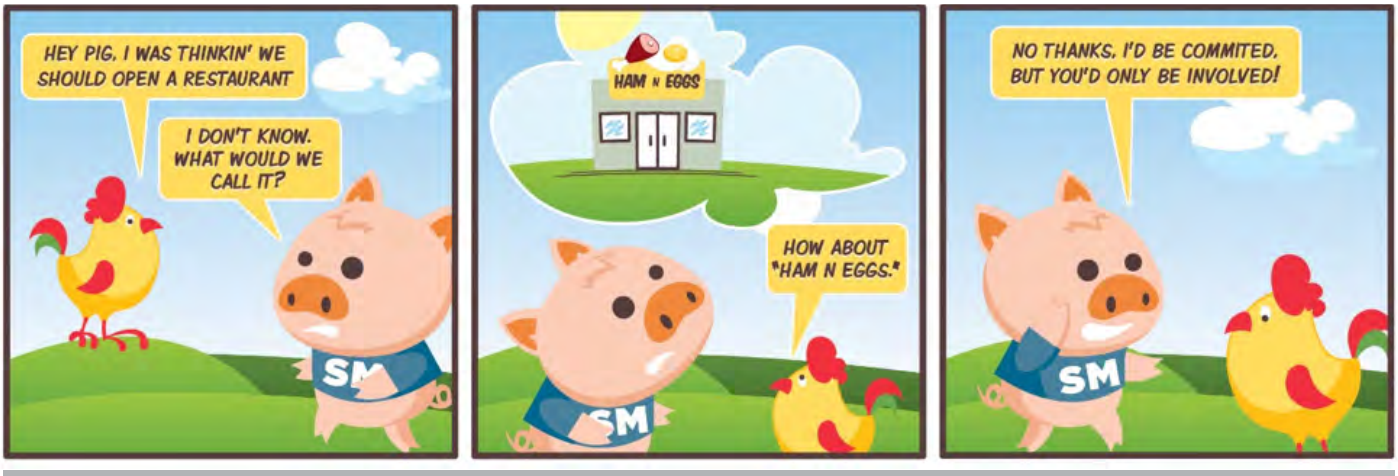


Figure 1: Committed or involved?

What does it mean to be committed? In Scrum, we like to talk about the Scrum team as pigs—they have to be committed to the product development effort. The Scrum team is in the trenches every day, making it happen. They are fully committed to the outcome of their effort. They definitely have "skin in the game."

Conversely, stakeholders (chickens) are only involved in the product development effort. Stakeholder input is requested for strategic planning purposes, during product development as needed, and during sprint reviews. Because stakeholders are not in the trenches every day, they are considered critical bystanders.

These definitions are not always clear, and there needs to be a word of caution.

Anyone involved with software development sometimes abuses the notion of pigs and chickens. Teams have been known to ostracize and isolate stakeholders from their meetings. I have witnessed individuals belittling stakeholders with comments such as, "You are a chicken and you are not allowed to speak." Please remember that any development effort is a partnership. The role of the chickens is no less than that of the pigs. It's just different.

By recognizing this fact, and treating stakeholders with respect, the team can be more successful.

The Scrum Walkabout

Figure 2 shows the best way I've found to present the overall Scrum process.

Throughout the Scrum project lifecycle, there are three primary roles:

Team members (TM): The group of individuals who get the work done

Product owner (PO): Represents the stakeholders who have influence on or are impacted by the team

ScrumMaster (SM): The "grease" that keeps everything running smoothly

Everyone else is considered a stakeholder whose input is valued and needed throughout the process. Everyone's ideas for the product go into what we call the product backlog. You can think of the product backlog as a dynamic entity—an iceberg. New ideas are constantly coming in and old ideas are falling off the bottom, melting away.

The product owner maintains the product backlog, solicits and takes in new ideas, refines existing ideas, and keeps them in priority order based on everyone's feedback.

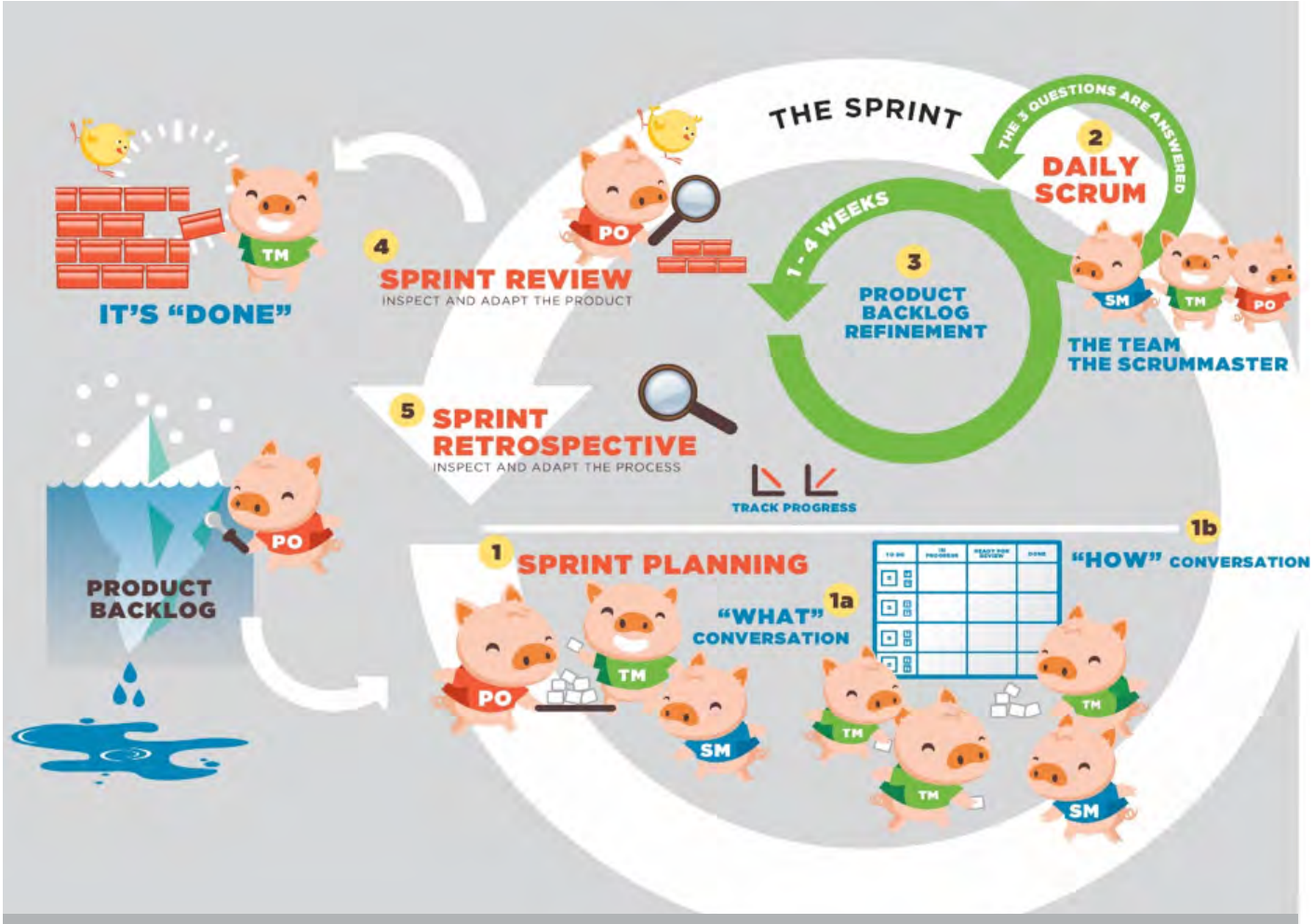


Figure 2: The Scrum process is highly iterative

Stakeholders usually want to get the product built as quickly as possible. Contrary to a traditional waterfall approach where one development phase follows another, Scrum iterates in cycles. This is accomplished by the team executing sprints. A sprint is a period of time, typically two to four weeks, at the end of which we expect to have a potentially shippable product increment that is a functional piece of the final product.

Each sprint begins with a sprint planning meeting. In this meeting, there are two primary conversations:

1. The first conversation is when the product owner presents the highest-priority product backlog items to the team. The team is then expected to figure out which ones they can realistically get done.
2. The second conversation is when the team dives into the technical work and identifies the tasks necessary to complete the agreed-upon product backlog Items. Once the scope of the sprint is set, the team is now ready to start building.

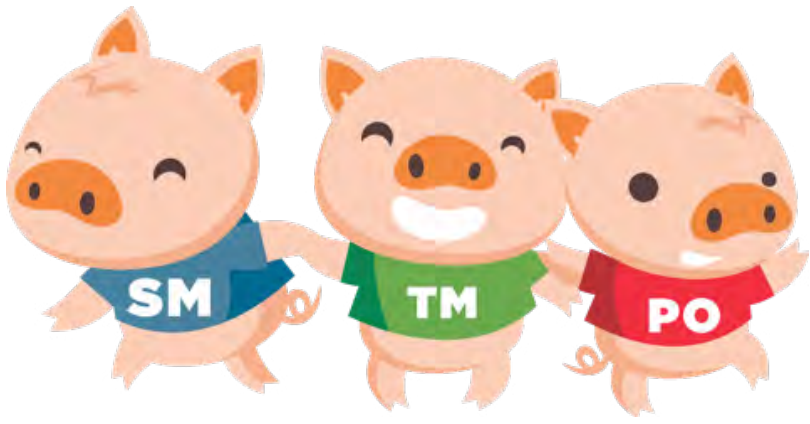


Figure 3: There are three roles in Scrum

The heartbeat of the sprint is the daily scrum meeting. This meeting is an opportunity for the team to come together briefly to discuss their progress, ask for help, and synchronize their efforts.

THE OUTPUT OF EVERY SPRINT IS A POTENTIALLY SHIPPABLE PRODUCT INCREMENT, ANOTHER BRICK IN THE PROVERBIAL WALL.

The team may incorporate other tools in order to promote transparency, like burndown charts and scrum boards.

At the end of the sprint, it's time to formally inspect and adapt what the team produced in order to improve. This takes place in a meeting called the sprint review. This is an opportunity to invite the stakeholders to see a demo and offer their feedback. The feedback from this meeting becomes new product backlog items. As an aside, savvy teams are constantly inspecting and adapting with real-time feedback from their product owner and stakeholders during the sprint. The output of every sprint is a potentially shippable product increment, another brick in the proverbial wall. This represents a vertical slice of functionality that is fully tested, documented, and ready to release to production. Whether to give this functionality to the stakeholders then becomes a business decision to be made by the product owner.

The directional arrows in figure 2 show that sprints move forward, one after the other, until the product owner calls the development effort "done." This can happen when a targeted date has been reached, the budget has been depleted, or enough business value has been delivered to meet the needs of the stakeholders.

Now that you have a basic overview of Scrum, let's examine a few parts and pieces in more detail.

The Roles in Scrum

The three Scrum roles each have different responsibilities that fit into the overall process. A team is ideally composed of five to nine members—studies have shown that seven is just about perfect. The team should be cross-functional, self-organizing, and self-managing. Teams that exhibit these characteristics tend to operate at maximum efficiency.

The product owner serves as the liaison between the team and the stakeholders. To the team, the product owner is the voice of the stakeholders, representing their needs, wants, and desires for the product.

The product owner has strategic oversight of the product from the organization's perspective; they own the return on investment for the product. They are involved in product planning through visioning, road-mapping, and release planning. In general, the product owner works with stakeholders and project sponsors to perform strategic planning.

The product owner is also responsible for the product backlog. They own it, maintain it, and prioritize it. They always assure that the needs of the stakeholders are being presented to the team

for implementation within the sprints. ScrumMaster is one of the most vital roles on a Scrum team. The ScrumMaster facilitates the Scrum process as servant leader. A ScrumMaster also acts as an "information radiator" to stakeholders and clears roadblocks out of the team's way.

The ScrumMaster serves to help the team effectively execute sprints and the entire Scrum process. While the ScrumMaster is not the manager of the team members, they guide the team in their execution of the Scrum process, coaching, cajoling, nudging, and sometimes escalating to human resources when necessary.

Ceremonies in Scrum

We've already touched on the four main Scrum ceremonies, or meetings: sprint planning, the daily scrum, sprint review, and the sprint retrospective. Let's examine the purpose of each of these ceremonies.

SPRINT PLANNING

In sprint planning, the work for the next sprint is determined. Generally, there are two separate conversations in this meeting, each of which is half of the meeting's allocated time (figure 4).

The first conversation should be attended by the entire Scrum team. The items from the product backlog are selected by the team for inclusion in the upcoming sprint. This is known as the "what" conversation.

The second conversation is when the team meets to finalize the sprint backlog. This is done by decomposing the selected product backlog Items into tasks and estimating each task in ideal man-hours. This "how" conversation is technical in nature and doesn't require the product owner's full attention.

DAILY SCRUM

The daily scrum meeting, often called the daily standup, is the most tactical of all the Scrum meetings.

THE SCRUMMASTER SERVES TO HELP THE TEAM EFFECTIVELY EXECUTE SPRINTS AND THE ENTIRE SCRUM PROCESS.

This meeting is held each workday during the sprint and is attended by the team members, the ScrumMaster, and the product owner. Although morning meetings are often preferred, the best answer to when a daily scrum should be held is "whatever time of day the Scrum team can commit to coming together for fifteen uninterrupted minutes."

During the daily scrum, each attendee traditionally answers these three questions:

1. What did I do yesterday?
2. What am I going to do today?
3. What are my roadblocks?

Savvy teams change these questions if there are better alternatives.

SPRINT REVIEW

At the end of each sprint, the team should have a working product, known as a potentially shippable product increment. In the sprint review, the stakeholders get to see what was accomplished, hear what work was accepted or rejected, and provide feedback and new ideas.

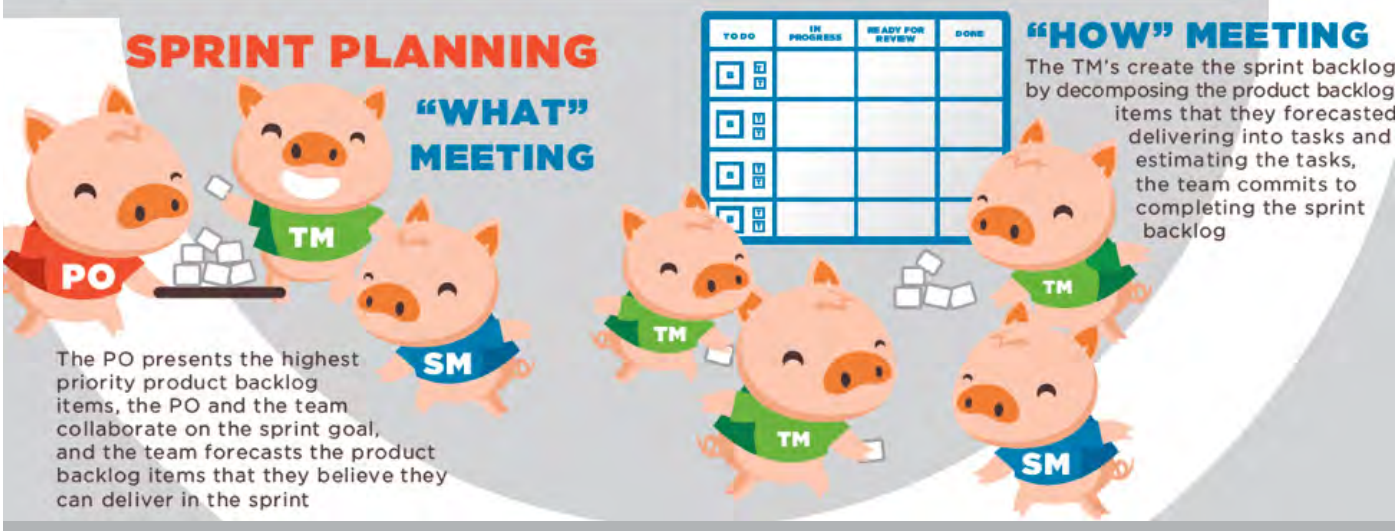


Figure 4: Sprint planning answers the questions *What?* and *How?*

AFTER THE STAKEHOLDERS HAVE LEFT THE ROOM, THE KEY INDIVIDUALS (TEAM MEMBERS, PRODUCT OWNER, AND SCRUMMASTER) MEET TO INSPECT AND ADAPT THE PROCESS.

SPRINT RETROSPECTIVE

At the completion of the sprint, a retrospective meeting is held. After the stakeholders have left the room, the key individuals (team members, product owner, and ScrumMaster) meet to inspect and adapt the process. The Scrum team looks at what worked well and what needs improvement, and they leave the meeting with tasks to make changes for their betterment. This meeting benefits the Scrum team directly, which ultimately benefits the product and stakeholders as a whole.

Once You Scrum, You'll Never Look Back

I often get asked about what specific benefits my team can expect to see when we implement Scrum. While only occasionally realized, executives want to hear examples like:

- Faster delivery
- Less waste
- More productivity

If we don't always achieve these big ticket items, what are the benefits? I find that Scrum teams are happier, build better products, and succeed more often. There's a lot more to share, but I am out of space in this article. [3]

I hope you enjoyed this back-to-basics look at Scrum. [BSM]

brian.rabon@braintrustgroup.com

CLICK FOR THIS STORY'S REFERENCES

Featuring fresh news and insightful stories about topics important to you, TechWell Insights is the place to go for what is happening in the software industry today. TechWell's 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 for the full stories and more!

5 Tips for Choosing Your First Agile Project

By Jeffery Payne

When transitioning to agile, applying agile methods to a single project is a great way to get started. However, care must be taken to ensure the project you choose is appropriate—it shouldn't be too large, take too long, or be too risky. Here are five tips to help you pick the right project for your agile pilot.

Read More

Think through System Changes to Anticipate Quality Issues

By Payson Hall

When you replace or significantly modify components of a larger system, too frequently we focus on whether the code we are building functions correctly. This is important, but it's also short-sighted. It's easy to introduce errors because we are changing interactions. Coding bugs are only one quality problem.

Read More

Continuous Exploratory Testing: Expanding Critical Testing across the Delivery Cycle

By Ingo Philipp

Continuous testing entails executing automated tests to obtain rapid feedback on business risks. Where does that leave exploratory testing? Obviously, it doesn't make sense to repeat the same exploratory tests across and beyond a sprint, but exploratory testing can be a continuous part of each software delivery cycle.

Read More

5 Myths and Misconceptions about Leadership

By Naomi Karten

It's a common myth that leaders are born, not made. Even so-called natural leaders have plenty to learn about handling the kinds of challenges and problems they'll have to face, and many others grow into the role. Let's explore this misconception and four others to learn that anyone with the drive can be a leader.

Read More

An Agile Approach to Change Management

By Steve Berczuk

Many organizations are reluctant to introduce new tools or technologies, or even to update existing ones. The reason is often framed in terms of risk management, but agile teams already have the tools to manage the risk of change: testing and experiments. These approaches together eliminate gaps in risk identification.

Read More

Why the Gig Economy Thrives in the World of DevOps

By Josiah Renaudin

Even if the industry is booming, it's not easy filling the full-time DevOps roles. Every software team is vying to find the perfect person to come in and establish a culture to promote improved software release cycles, software quality, security, and rapid feedback on product development. But it's not easy.

Read More

4 Trends You'll See in the Tech Workplace in 2018

By Beth Romanik

A new year means new technologies changing how we work, and the software industry is affected by these shifts more than most. Let's look at four trends we're likely to see in tech workplaces in 2018: continuing education, artificial intelligence and machine learning, data privacy, and more employee interaction.

Read More

A Tester's Guide to Choosing a Programming Language

By Justin Rohrman

Many testers want to learn a programming language, but how should they decide which one? Justin Rohrman suggests finding an authentic problem to solve and moving from there to determine which language would be best. You can also ask developer coworkers for suggestions and help—take advantage of available resources.

Read More

Continuous Testing, Continuous Variation

By Hans Buwalda

With the arrival of continuous integration/continuous delivery (CI/CD), the notion of continuous testing is taking center stage. Knowing that comprehensive tests are running smoothly can be of benefit for the CI/CD pipeline. Using the repetitive character of CI/CD for testing can be a way to address issues.

Read More

4 Ways to Use Virtual Reality in Your Workplace

By Anthony Coggine

Businesses are adopting virtual reality as a means of strengthening marketing tactics, increasing collaboration, and connecting with consumers. For those new to VR, it's important to understand how a virtual world could be used in your day-today operations. Here are four ways virtual reality will impact the workplace.

Read More

How Testers Can Collaborate with the ScrumMaster

By Michael Sowers

ScrumMasters serve the team by providing facilitation and coaching, but they also have many challenges. Those in testing roles are in a good position to collaborate with the ScrumMaster to improve agile processes. Here are some ways testers can partner with, support, and assist the ScrumMaster—and the rest of the team.

Read More

Application Release Automation: Why the QA Pro Should Care

By Tracy Ragan

The speed of testing depends on a consistent software release process that can provide critical information when reporting issues. QA pros will benefit from a new set of DevOps tooling called application release automation, which drives continuous release deployment and provides visibility about what was deployed.

Read More

6 Skills Needed for Exceptional Exploratory Testing

By Nicholas Roberts

While anyone can claim to be an exploratory tester, only those with a set of honed skills will discover hard-to-find bugs that could impact your mobile app or website. Exploratory testers must possess these six skills if they are to find the edge cases that could derail a successful software release.

Read More

Why Software Testing Is Key to DevOps

By Alan Crouch

One of the major reasons organizations adopt DevOps practices is to accelerate delivery of software to production. However, many fail to include quality components in their practices. Continuous deployment without quality is just delivering continuous bugs. Here's why software testing is an essential part of DevOps.

Read More

Scrum Isn't the Only Path to Agility

By Tom Stiehm

Scrum can really help a team to become more agile. But that doesn't mean it is the only way for a team to become agile. Agile is all about self-organizing teams collaborating to find what works for them, so if a nontraditional approach helps your team get started, then you're just forging a new path to agility.

Read More

Testing the Requirements: A Guide to Requirements Analysis

By Evgeny Tkachenko

Everyone knows testing requirements is important, and everyone says they do it, but it seems like no one knows exactly how. The best way to solve this problem is to introduce a requirements analysis stage that has to be done before coding starts. No one knows a product as well as a tester who works with it every day!

Read More

Testing Next-Generation Digital Interfaces

By Amir Rozenberg

With chatbots, facial recognition, voice integrations, and more, digital interfaces have a complex software side. With concrete examples from the market, Amir Rozenberg offers new approaches for embedding quality and test activities into the development cycle when dealing with this new generation of digital interfaces.

Read More

The Unspoken Truth about IoT Test Automation

LESSONS LEARNED FROM TRADITIONAL TEST AUTOMATION TECHNIQUES REQUIRE A MUCH DIFFERENT APPROACH WHEN TESTING CONNECTED, SMART DEVICES.

by Rama R. Anem | *rama.anem@gmail.com*

The internet of things (IoT) phenomenon has opened a new part of a market that will grow dramatically in the next few years. Forecasts show up to fifty billion connected devices by 2020, all of which will demand new development tools, frameworks, and testing techniques. [1]

What Is the IoT?

The IoT is a network of connected electronic devices that incorporate sensors, actuators, and any object that can send or receive data. Each "thing" in this network has a unique identifier to send or receive data or commands.

Lower costs, higher computing powers, and availability of Wi-Fi and other wireless networks make these kinds of electronic devices popular tools to make our lives easier and more comfortable. IoT devices can save money, as well, if they are integrated as part of a smart home. One example is the Nest thermostat, which can be controlled remotely and adapts to your lifestyle.

IoT networks generate lots of data, which impacts the way that data is processed and stored. This means IoT development and testing is strongly linked to the methods used to handle big data.

Emulating an environment is more critical for hardware tests, while infrastructure emulation is required to test the device's firmware and software.

The IoT Requires Different Test Techniques

Hardware QA approaches vary depending on the type of hardware. For physical products, you use physical property tests, calibers, and hot/cold rooms, depending on the device. For electronics, tests include "bed of nails" tests and visual inspection before shipping out. A typical example is to run display segment tests to verify the quality of a display and its connectivity.

Tests for software also vary depending on the platform, type of application, and reliability requirements. For example, typical test scenarios and user flows will be very different for web-based, mobile, and desktop applications. For Java, there can be some surprises with system performance when the garbage collector starts running. Reliability tests (load tests and stability tests) are rarely performed on desktop applications, but they are a must-have for server-side code.

Why does the IoT require such special treatment? IoT testing combines both software and hardware tests. In many cases, it also depends on infrastructure, other devices in the network, and environmental factors. This results in a greater count and complexity of test scenarios, as end-to-end tests have many more links on average. Long end-to-end tests require either very specific test conditions to validate every module or a clever logging system that improves the testability of the IoT solution. Attention to this type of testing usually reduces costs and training times for QA in the long run.

IoT Testing and Test Automation

I'd like to think that IoT testing, like other application testing, can always be automated.

In the traditional waterfall development model, automation may not have as significant an impact on timeline and product quality as it has had on delivering SaaS products in the agile era. Today, new functionality is being added faster, release timelines are more aggressive, and there is simply no time for manual testing. The product lifecycle is different and requires frequent regression tests. It would be a nightmare if we couldn't use test automation.

Testing for hardware and software products is well documented and has many approaches that have proven to be highly efficient. But IoT testing is a relatively new task.

One of the main challenges for the IoT is emulating its environment and infrastructure. Emulating an environment is more critical for hardware tests, while infrastructure emulation is required to test the device's firmware and software. Here are a couple of examples.

- Emulation of IoT infrastructure: This may be the easiest task of all because you can use existing services that create IoT brokers using different protocols. Write simple stub code to test connectivity and ensure two-way communication. Tests that enable an IoT device to send data to a server every five minutes is an example of one-way communication. An example of two-way communication is a server being enabled to send commands to an IoT device that will process and respond to each command.
- Emulation of IoT network: To test an IoT server, you need at least one emulated or real client. Functional tests can be performed once there is access to raw data sent by the device. If the system supports many device models and configurations, it may become difficult (or just inefficient) to keep them all in a lab for testing. A lab set up to support a wide range of configurations may not be sufficient for all nonfunctional testing, and this test environment will not prepare a system for actual load conditions. There are two choices you can make: buy more hardware (switches, extension cords, and other things), or think of a way to simulate the IoT device network.

The usual recommendation shouldn't limit simulation tests to nonfunctional features. However, it is a great chance to do end-to-end testing by simulating multiple devices. This approach can be used to validate how the overall system processes data from all possible device combinations.

Nonfunctional tests covered by a simulator could include:

- Load tests that show how much load the system can take and where the bottlenecks are
- Scalability tests that show the ability of the infrastructure to adapt to increasing load
- Interactivity tests that show how the system handles real-time activity and delayed two-way communication

Going Forward with Test Automation

Each project requires special consideration to support your specific testing goals. Some investments to increase testability of the solution will likely give benefits in the future, when the number of clients grows. For the IoT now, however, using a mix of test automation to validate functional and nonfunctional testing is a must. Avoid manual testing techniques at all costs! [BSM]

CLICK FOR THIS STORY'S REFERENCES

Better Software (ISSN: 1553-1929) is published four times per year: January, April, July, and October.