WINTER 2018

BETTER SOFTWARE™

A TECHWELL™ PUBLICATION

What Testers Need to Know about Continuous Testing

AUTOMATION'S ROLE IN THE FALL OF SOFTWARE TESTING
Are we becoming too dependent on test automation?

BUILDING AUTONOMOUS DEVOPS CAPABILITY IN DELIVERY TEAMS
There is a better way to structure DevOps for success

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 April 4, 2018 with code CWBSM to save up to $800 off your conference*

TO REGISTER CALL 888.268.8770 | BSCWEST.TECHWELL.COM

*Discount valid on packages over $400

Ranorex®

All-in-One Test Automation

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

```
void ITestModule.R
{
    Report.Log(R
    "Website", "O

    Report.Log(R
    "Mouse", "Mo

    Report.Log(R
```

Now with Selenium Integration

- Access Selenium with the Ranorex automation framework
- Address Selenium pain points
- Web testing across all major platforms and browsers
- For testers and developers

www.ranorex.com/try-now

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

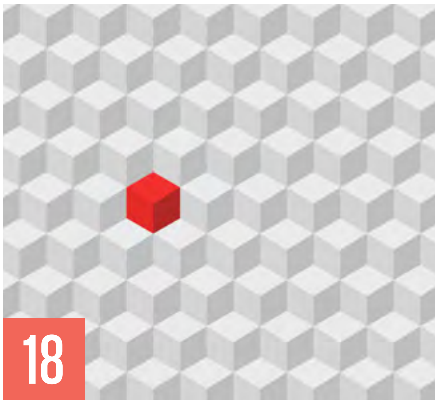
SUPER EARLY LOBSTER SAVINGS WHEN YOU REGISTER BY APRIL 27, 2018

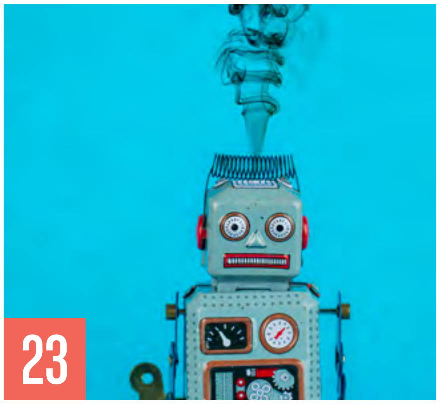
INSIDE

Volume 20, Issue 1
WINTER 2018

On the Cover

Features





Columns

Departments

MARK YOUR CALENDAR

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

Feb. 27–Mar. 1, 2018 Chicago, IL	**Mar. 6–8, 2018** Atlanta, GA	**Mar. 20–22, 2018** Washington, DC	**Apr. 10–12, 2018** Salt Lake City, UT
Mar. 6–8, 2018 Dallas, TX	**Mar. 20–22, 2018** San Jose, CA	**Apr. 10–12, 2018** Detroit, MI	**May 1–3, 2018** Philadelphia, PA

Agile Testing Training Week

http://www.sqetraining.com/agile-week

Feb. 20–23, 2018
Boston, MA

Mar. 20–23, 2018
Chicago, IL

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

2018 Is the Year to Trust Your Customer

Ken Whitaker
kwhitaker@techwell.com
Twitter: @Software_Maniac

In each issue of our magazine we try to provide inspiration for making "better" software. Yet I continue to be discouraged with how software vendors manage end-user licensing. In our industry's quest to protect IP and revenue, vendors often ignore the impact it has on the customer and the end-user. As commercial software apps migrate toward subscriptions, this can be a win for the software provider and a win for the end-user—as long as the app continues to provide value.

But the administration imposed by subscription models on end-users and technical support individuals can be painfully irritating. You, as a software developer or tester, may not even hear about it. Most of us have had frustrating experiences attempting to resolve licensing issues. There is a careful balance between protecting your company's investment and simplifying the end-user experience. Designing a trusting, easy-to-administrate software licensing mechanism cannot be an afterthought.

Enough of my rant. Let's talk about this issue of *Better Software*.

Our featured article by Wayne Ariola, "What Testers Need to Know about Continuous Testing," presents a wonderful introduction to how modern testing is changing. By using test automation and frequent evaluation of business risk, you can determine the best time to deliver.

As a practicing business analyst, Ron Healy shows how agile is perceived in "Agile outside the Development Team." If you are struggling to get nondevelopers to accept your agile practices, this is just the "secret sauce" you need to know. Has the rise in test automation resulted in poor quality? John Tyson seems to think so in his provocative "Automation's Role in the Fall of Software Testing." John gives good advice on how to right the ship. In Miiro Juuso's "Building Autonomous DevOps Capability in Delivery Teams" article, you'll learn the best way to structure your DevOps organization for success.

If you haven't had success providing quality in your software products, consider following Jeffery Payne's advice in "5 Ways to Pair Developers with Testers." In "Adopt an Innovative Quality Approach to Testing," Rajini Padmanaban offers new perspective on testing in production.

If you like *Better Software*, please spread the word via Twitter and Facebook. And let us and our authors know what you think of the articles by leaving your comments. We value your feedback.

BETTER SOFTWARE
A TECHWELL PUBLICATION

FOLLOW US

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
Jamie Borders
jborders.com

Advertising

SALES CONSULTANTS
Daryll Paiva
Kim Trott

PRODUCTION COORDINATOR
Alex Dinney

Marketing

MARKETING MANAGERS
Cristy Bird
Patrice Johnson

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

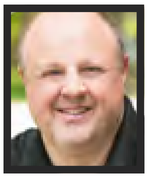
CONTRIBUTORS

Wayne Ariola, CMO of Tricentis, is a recognized thought leader on continuous testing, risk-based testing, service virtualization, and API testing. He has been a contributor to the software testing space for more than fifteen years and in the software industry for more than twenty. Wayne has created and marketed products supporting dynamic software development, test, and delivery. He has driven the design of many innovative technologies and received several patents for his inventions. Reach Wayne at w.ariola@tricentis.com.

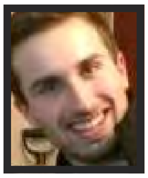
Ron Healy has had a diverse fifteen-year career as a senior business analyst, product owner, entrepreneur, innovator, lecturer, and corporate trainer. Ron has helped organizations with agile software development, e-commerce, Internet of Things, and legacy modernization. He is a keen proponent of using agile techniques only when it makes sense and is a firm believer in lifelong learning. Contact Ron at ronhealyx@gmail.com.

Miiro Juuso is a sysadmin turned software engineer, turned salesman, and returned to sysadmin. As the DevOps lead at AND Digital in London, he helps Financial Times Stock Exchange 100 companies build DevOps capability and deliver better software. In his spare time, Miiro blogs about DevOps tools, methodologies, and practices (getintodevops.com). Reach Miiro at miiro.juuso@gmail.com.

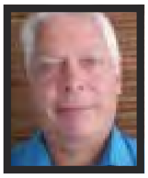
As vice president, **Rajini Padmanaban** leads the engagement and relationship management for some of QA InfoTech's largest and most strategic accounts. She has over sixteen years of professional experience, primarily in software quality assurance. Rajini actively advocates software quality assurance through evangelistic activities including providing insights and blogging on test trends, technologies, and best practices. Contact Rajini at rajini.padmanaban@qainfotech.com.



A frequent TechWell contributor, **Jeffery Payne** is CEO and founder of Coveros, Inc. Since its inception in 2008, Coveros has become a market leader in secure agile principles and recognized by Inc. magazine in 2012 as one of the fastest growing private companies in the country. Jeffery has published more than thirty papers on software development and testing. He has testified before Congress on issues of national importance including intellectual property rights, cyber terrorism, and software quality. Reach Jeffery at jeff.payne@coveros.com.



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.



A software testing and QA professional with more than twenty years of testing experience, **John Tyson** considers himself fortunate to have worked mostly in agile environments. A proponent of lean software development, John makes extensive use of black box and exploratory testing. His clients include startups, the public sector, non-profits, Fortune 500, and multi-national corporations. Reach John at jmt_research@yahoo.com.

5 Ways to Pair Developers with Testers

THERE IS AN ART TO TEAM ORGANIZATION, AND YOU'LL GET THE BEST OVERALL QUALITY RESULTS BY PAIRING DEVELOPERS WITH TESTERS.

by Jeffery Payne | *jeff.payne@coveros.com*

Even though the agile movement is nearly two decades old, many organizations still struggle with how to get their developers and testers working together. In my experience, some types of software developers, including front-end, business logic, back-end, and UI/UX designers, have figured out how to comfortably work together. However, developers and testers often are more aligned within their functional silos, causing sprints to operate more like mini-waterfalls than collaborative teams.

Until this gap between developers and testers is closed, teams operating like this will continue to struggle to complete estimated work in sprints. Instead of addressing the underlying gaps between developers and testers, teams often place a bandage on problems by increasing the duration of their sprints or shifting some portion of the testing process out into future sprints. Both of these tactics result in longer feedback loops that only decrease productivity and increase rework. The only way to truly solve this problem is to change the way software developers and testers work.

There is a simple developer-tester pairing approach to solve this problem. Pairing developers and testers on each user story forges stronger relationships, and this collaboration and communication results in better software.

Pairing developers and testers on each user story forges stronger relationships, and this collaboration and communication results in better software.

Based on my experience, I have five suggestions for pairing developers with testers.

1. Define User Story Acceptance Tests

Acceptance tests that satisfy user story acceptance criteria ideally should be created before a story is implemented so its developer can verify that the code works as expected. Software testers are often tasked with defining acceptance tests early in each sprint, either individually or in conjunction with other testers.

Having a developer and tester work together to define acceptance tests is a great way to get them on the same page. This collaboration around each user story results in a clear understanding of what needs to be implemented to satisfy customer needs.

Some who have been schooled on the importance of independence between developers and testers may bristle at the idea of developers and testers working so closely together to define test cases. In practice, the benefits of this collaboration far exceed any danger that developers and testers will miss important defects.

2. Code Review Unit and Integration Tests

Typically, developers are responsible for creating unit tests for their code, and testers are responsible for integration testing new stories with other code. Why not have them help each other in the process?

Many developers are not effective testers and can benefit from walking through their unit tests with a professional tester. Testers can help them with several considerations, such as boundary conditions that may not be fully tested or risky areas of the code that need additional testing, and they also can ask critical questions about the tests that can push the developer to improve their testing approach. In addition, testers can help assure that developers don't focus too much on code coverage (and any other exit criteria used for unit testing) instead of on the quality of the code.

In terms of integration testing, developers often understand the overall structure and design of the application better than testers and can suggest additional integration tests that are necessary to exercise integration points and object relationships.

3. Perform Exploratory Testing on User Stories

Often the amount of testing performed to validate acceptance criteria for user stories is insufficient to ensure quality. Exploratory testing is a great way to supplement acceptance testing and find more bugs prior to code check-in.

Getting developers and testers to perform exploratory testing together before declaring a user story done not only finds additional bugs but also builds a quality culture for the entire team. Exploratory testing helps teach developers how to think critically about testing, and that helps them test more effectively in other areas.

To encourage developers and testers to test together more often, consider including timeboxed exploratory testing as part of the definition of "done" for all user stories.

4. Extend Pair Programming to Include Testers

Since the creation of Extreme Programming, pair programming has been advocated as a way to increase productivity and reduce rework through constant collaboration during code implementation. Development organizations may employ traditional developer-developer pairing, but they often overlook the advantages of having developers and testers pair up.

While testers may not actually code, having a tester listen as the developer talks through what they are implementing has tremendous benefits. This collaboration can help identify coding missteps, uncover ambiguities in understanding, and give the tester more context for how the application works. Set aside some time for pair programming between the developers and testers paired on user stories—it can make a difference in product quality.

If software testers have software development experience, you can take this approach to another level by having developers and testers periodically switch roles while pair programming. Doing so will increase collective code ownership on your teams and improve product quality.

5. Approach Test Automation Development Differently

As your codebase grows, it gets more difficult to completely regression test any code changes during sprints without using automation. To support a continuous integration model, automation tests need to be created along with the code it tests instead of after the fact. Consider having your developers and testers work together to automate user story acceptance tests during story development.

Leverage behavior-driven development (BDD) tools such as Cucumber or SpecFlow to provide a way for developers and testers to participate in automation. The software tester takes responsibility for defining the acceptance criteria in a BDD language such as Gherkin that can be automatically executed by one of the tools mentioned above.

The developer creates the fixtures necessary to hook Gherkin tests to the application so the proper methods are executed during test runs. Of course, if your developers and testers all have software development skills, there are other ways to pair and get test automation done.

Start Pairing Your Team

So now that you know how to pair successfully, get started! Pick a task, grab a teammate, and give it a go. Set a goal to try and pair with each of your teammates at some point during each sprint. Make a game of it if it helps. Create a pairing board and track who successfully pairs with everybody else first. Remember, having your developers and testers collaborate day to day on a variety of activities not only builds stronger relationships between team members and breaks down silos but also improves the quality of your applications. You won't be sorry that you did. [BSM]

"I think the average company does not take bug reporting seriously, as you can see if you've ever tried to report a website problem and couldn't find any ways on a website to submit feedback or contact a person who could make a fix."

"I do see that mentality a lot, of shipping first to meet a deadline and fixing problematic issues later. It shouldn't be that way, but the race to release something new takes precedence over the need to have all flaws ironed out."

"If you've got a faulty product, there is a large likelihood your users will just go elsewhere instead of taking the time and effort to tell you about a problem they're experiencing."

Why Bug Reporting Is More Important than Ever Before

Sam Kaufman

Years in Industry:	**12**
Email:	**sam@bugreplay.com**
Interviewed by:	**Josiah Renaudin**
Email:	**jrenaudin@techwell.com**

"Agile is definitely a double-edged sword in regards to bugs. Just the name says a lot about the actual goal, which is shipping a lot of software, fast. There's simply no way to ship software fast without also shipping bugs."

"Modern browsers have incredibly complex diagnostic utilities built right into them, and you just need a tool that can plug into the browser and record all those details whenever a user encounters a problem."

"Rapid development is why a lot of agile shops do focus on writing tests, which do go a long way towards catching bugs before they hit production."

"Showing that you actually care about your customers and their experience using your software is definitely a way to stand out from the crowd today."

CLICK HERE FOR THE FULL INTERVIEW

search

Search

where to learn about the hottest topics in testing

TOP RESULTS:

STAR EAST

A TECHWELL EVENT

STAR*EAST* Testing Conference – April 29–May 4, 2018 – Orlando, FL ...

https://stareast.techwell.com/

Come to STAR*EAST* to get answers to all your toughest software testing questions:

- Testing in DevOps
- Test Transformation
- Test & Release Automation
- Big Data, Analytics, A/I Machine Learning for Testing
- Agile Testing
- Testing for Developers
- Security Testing
- Test Strategy, Planning, and Metrics
- Test Leadership
- Performance Testing & Monitoring

Register early for the best pricing plus a $50 bonus gift card!

SEE PRICING & PACKAGES

What Testers Need to Know about Continuous Testing

Wayne Ariola

Like Lucy and Ethel struggling to keep pace at the chocolate factory in *I Love Lucy*, [1] many software testers scramble to keep pace with accelerated processes—then along comes the supervisor, proclaiming, "You're doing splendidly. Speed it up!"

As expectations associated with testing change, legacy testing platforms simply aren't keeping up due to their heavy approach to testing. They rely on brittle scripts; deliver slow, end-to-end regression test execution; and produce an overwhelming level of false positives. As a result, legacy testing achieves limited success with test automation.

According to industry sources, the overall test automation rate is well below 20 percent and feedback I've received shows that the results of test automation are just "so-so." [2]

Traditional Testing Isn't Enough

Recent changes across the industry demand more from testing while making test automation even more difficult to achieve. There are several reasons for this:

- Application architectures are increasingly more distributed and complex. They embrace cloud, APIs, and microservices, creating virtually endless combinations of different protocols and technologies within a single business transaction.
- Thanks to agile, DevOps, and continuous delivery, many applications are now released anywhere from every two weeks to thousands of times each day. As a result, the time available for test design, maintenance, and especially execution decreases dramatically.
- Now that software is the primary interface to the business, an application failure is a business failure. Even a seemingly minor glitch can have severe repercussions if it impacts the user experience. As a result, application-related risks have become a primary concern for even nontechnical business leaders.

We need to transform the testing process as deliberately and markedly as we've transformed the development process.

Given that software testers face increasingly more complex applications, they are expected to deliver trustworthy, go/no-go decisions at the new speed of modern business. Traditional testing approaches won't get us there. We need to transform the testing process as deliberately and markedly as we've transformed the development process. This transformation requires a different approach altogether: continuous testing.

What Is Continuous Testing?

Continuous testing is the process of executing automated tests as part of the software delivery pipeline. It provides rapid feedback on the business risks associated with a software release candidate.

Test automation is designed to produce a set of pass/fail data points, correlated to user stories or application requirements. Continuous testing, on the other hand, focuses on business risk and provides insight on whether the software can be released. To achieve this shift, we need to stop asking "Are we done testing?" and instead concentrate on "Does the release candidate have an acceptable level of business risk?"

Table 1 shows the key attributes of continuous testing.

Assesses business risk coverage as its primary goal
Establishes a safety net that helps the team protect the user experience
Requires a stable test environment to be available on demand
Integrates seamlessly into the software delivery pipeline and DevOps toolchain
Delivers actionable feedback appropriate for each stage of the delivery pipeline

Table 1: Five key characteristics of continuous testing

Continuous Testing Is More than Test Automation

The differences between continuous testing and test automation can be grouped into three categories: risk, breadth, and time.

BUSINESS RISK SHOULD BE CONSTANTLY EVALUATED

Businesses today not only have exposed many of their internal applications to the end-user, but also have developed vast amounts of additional software that extends and complements those applications. For example, airlines have gone far beyond exposing their once-internal booking systems. These systems now let customers plan and book complete vacations, including hotels, rental cars, and activities. Exposing more innovative functionality to the user is now a competitive differentiator. However, there is a major downside. This additional functionality can dramatically increase the number, variety, and complexity of potential failure points.

Large-scale software failures can have such severe business repercussions that application-related risks have become prominent components of a public corporation's financial filings. [3] Given that notable software failures resulted in an average 4.06 percent decline in stock price, it's not surprising that business leaders are taking note. This equates to an average $2.55 billion loss of market capitalization, and management expects IT leaders to take action.

If your test cases weren't built with business risk in mind, your test results won't provide the insight needed to assess risks. Most tests are designed to provide low-level details on whether user stories are correctly implementing the requirements—not high-level assessments of whether a release candidate is too risky to release. Would you automatically stop a release from taking place based on test results? If not, your tests aren't properly aligned with business risks.

This doesn't mean that low-granularity tests aren't valuable. Instead, it suggests more action is needed to stop high-risk candidates from going out into the wild unchecked. Table 2 shows what testers need to do in order to address risk.

Understand the risks associated with the complete application portfolio
Map risks to application components and requirements (which then are mapped to tests)
Use a test suite that achieves the highest possible risk coverage with the fewest test cases
Always report status that shows risk exposure from business, technical, performance, and compliance perspectives

Table 2: What testers need to do to address risk properly

THE BREADTH OF TEST COVERAGE COUNTS

Even if a business manages to steer clear of large-scale software fails that make headlines, seemingly minor glitches can still cause trouble. If any part of the user experience fails to meet expectations, you run the risk of losing that customer to a competitor. You also risk brand damage if that user decides to expose issues to social media.

Just knowing that a unit test failed or a UI test passed doesn't tell you whether the overall user experience is impacted by recent application changes. To protect the end-user experience, run tests that are broad enough to detect when an application change inadvertently impacts functionality that users have come to rely on. There are several techniques, shown in table 3, that I've found invaluable when addressing testing breadth.

Define and execute complete end-to-end tests that exercise the application from the user's perspective
Provide integrated support for all technologies involved in critical user transactions (web, mobile, message/API-layer, SAP and packaged apps, etc.)
Simulate service virtualization for dependent components required to exercise complete end-to end transactions that aren't either available or configurable for repeated testing
Ensure that tests and service virtualization assets are populated with realistic and valid data every time the tests are executed
Perform exploratory testing to find user-experience issues that are beyond the scope of automated testing (e.g., usability issues)

Table 3: What testers need to do to address breadth of testing

TESTS SHOULDN'T IMPACT TIME TO MARKET

As the speed at which organizations ship software has become a competitive differentiator, the vast majority of organizations are turning to agile and DevOps to accelerate their delivery processes.

When automated testing emerged, it focused on testing internal systems that were built and updated according to waterfall development processes. All systems were under the organization's

control, and everything was completed and ready for testing by the time the testing phase was ready to start. Now that agile processes are becoming the norm, testing must begin in parallel with development. Otherwise, the user story is unlikely to be tested and deemed "done-done" within the extremely compressed iteration time frame.

If your organization has adopted DevOps and is performing continuous delivery, software may be released hourly—or even more frequently. In this case, feedback at each stage of the process can't just be fast; it must be nearly instantaneous.

If quality is not a top concern for your application (e.g., if there are minimal repercussions to doing a rollback when defects are discovered in production), running some quick unit tests and smoke tests on each release might suffice. However, if the business wants to minimize the risk of faulty software reaching an end-user, it needs a quick way to achieve the necessary level of risk coverage and testing breadth.

For testing, there are several significant impacts:

- Testing must become integral to the development process (rather than a "hygiene task" tacked on when development is complete)
- Tests must be ready to run almost as soon as the related functionality is implemented
- The organization must have a way to determine the right tests to execute at different stages of the delivery pipeline (smoke testing upon check-in, API/message layer testing after integration, and end-to-end testing at the system level)
- Each set of tests must execute fast enough that it does not create a bottleneck at the associated stage of the software delivery pipeline
- A way to stabilize the test environment is needed to prevent frequent changes from causing an overwhelming number of false positives

Table 4 summarizes what testers need to do to address time pressures.

Identify which test cases are critical for addressing top business risks
Define and evolve tests as the application constantly changes
Rebalance the test pyramid so that most tests execute at the API layer, which is at least a hundred times faster than UI test execution
Integrate tests into the delivery pipeline
Run distributed tests across multiple virtual machines, network computers, or in the cloud, as appropriate
Enlist service virtualization and synthetic data generation or test data management so that testing doesn't need to wait on data or environment provisioning

Table 4: What testers need to do to address time pressures

Set Up Your Team for Continuous Testing Success

If you only take away one idea from this article, remember these two algorithms:

```
Test automation ≠ continuous testing
Continuous testing > test automation
```

Even teams that have achieved fair levels of success with traditional test automation tools hit critical roadblocks when their organizations adopt modern architectures and delivery methods:

- They can't create and execute realistic tests fast enough or frequently enough
- The constant application changes result in overwhelming numbers of false positives and require seemingly never-ending test maintenance
- They can't provide instant insight on whether the release candidate is too risky to proceed through the delivery pipeline

It's important to recognize that no tool or technology can instantly give you continuous testing.

It's important to recognize that no tool or technology can instantly give you continuous testing. Like agile and DevOps, continuous testing requires changes that impact people, processes, and technology. Trying to initiate necessary changes in people and processes when your technology is not up to the task will be an uphill battle from the start, as will only providing new tools without trying to explain the purpose behind continuous testing and getting your teams on board. In my experience, this ultimately fails.

If your organization is starting or scaling continuous testing automation efforts, there are two recent research studies by Gartner and Forrester Research for you to review. [4, 5] Both reports provide insight into continuous testing and test automation trends as well as how the top continuous testing tools compare. [BSM]

w.ariola@tricentis.com

CLICK FOR THIS STORY'S REFERENCES

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
» Fundamentals of DevOps Certification—ICAgile
» 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

AGILE OUTSIDE THE DEVELOPMENT TEAM

BY RON HEALY

The story of how the Agile Manifesto came about has faded into legend. More than fifteen years later, a whole generation of programmers currently practicing their craft in some form of agile-like methodology may have no idea that there was ever such an era as BA (before agile).

Most of us, however, wonder about the non-agile mind-set of those outside the development team. Archaic processes that are routinely wrapped around projects in enterprises are as quaint and unfathomable to agile software developers as sundials are to the smartwatch generation. They're sort of related—but just barely.

Recognizing the Benefits of Agility

Because of perceived problems with waterfall projects, the Agile Manifesto was conceived by software engineers, for software engineers. Like most buzzwords, the term agile has recently become commoditized and homogenized to the point where it is often thrown around by people who have no idea what it really signifies.

The history of the Agile Manifesto includes two phrases I think are illustrative: [1]

> "A bigger gathering of organizational anarchists would be hard to find …"
> "Agile approaches scare corporate bureaucrats …"

Despite the fact that I am not technically gifted enough to be a software engineer, I regularly work closely with software teams. As a business analyst, I have found that it is important to be comfortable when challenging the status quo and dealing with unpredictability. This might explain why the Agile Manifesto appeals to me and why the groundbreaking philosophical shift that came out of that meeting scared process-driven, plan-hungry, and milestone-focused project people.

AS A BUSINESS ANALYST, I HAVE FOUND THAT IT IS IMPORTANT TO BE COMFORTABLE WHEN CHALLENGING THE STATUS QUO AND DEALING WITH UNPREDICTABILITY.

Software development teams have always been quicker than corporate bureaucrats to identify and adapt the benefits of agile. Eventually, though, management started to get it. Agile is all about completing work early with a focus on effective communication, feedback, and delivery. This often translates to higher return on investment. Senior managers liked this, and some started to believe that every part of the software development process must

become agile. But organizations accustomed to structured IT projects thought they could achieve this by doing little more than wrapping agile development in familiar, non-agile corporate planning processes.

Unfortunately, it isn't that simple.

Worse, some people began to believe that they could devise a super-agile methodology that was perfect for all kinds of software development projects—which is the antithesis of agile—and doomed both to failure and to rejection by agile proponents. As a business analyst experienced in both modern and traditional projects, this was painfully obvious to me.

Exploring Hybrid Agile Alternatives

Software developers are usually not interested in Gantt charts or budgets, whereas project managers and executives live with these concepts and associated artifacts. Because the decision-makers and project planners are generally in the latter group, it is not uncommon to see agile ceremonies, practices, terms, and buzzwords used in what are essentially project-centric, even waterfall processes. These are sometimes referred to as hybrid waterfall-agile methodologies—"wagile" for short—to make them appear as if they fit a deliberate structure, when they actually don't. On a humorous note, some people use the term "frAgile" to refer to these approaches, which isn't far from the truth.

Even in these so-called agile environments, delivery deadlines are sometimes defined months in advance. Budgets and resources are estimated based on some vague collection of sentences and aspirations in a high-level, requirements-like document.

Typically, the plan will even specify the number of sprints included! Someone signs off on this fictional plan without having any idea how it is to be implemented. As ludicrous as this sounds, this scenario is probably familiar to most of you.

Then, when the technical team gets their hands on the plan and starts tearing up the Gantt chart with their pesky objections and warnings, stakeholders are often shielded from the noise. Even when concerns are escalated, stakeholders often point to the signed-off plan and insist it get delivered, along with anything else that might get added along the way. After all, isn't the freedom to add requirements to existing projects exactly what agile is all about?

Agile evangelists might talk about transitioning an entire organization to agile. However, all they can realistically hope to do is introduce an agile mind-set to those outside the development team so they appreciate the benefits of agile and adapt whatever agility makes sense for them. Agile is simply not suited to every part of every project or every organization. If everyone were to do everything in an agile fashion, how would anyone know what's coming down the tracks?

Agile proponents bemoan the fact that the corporate and project world insists on shoehorning agile into processes that are inherently nonagile. Others find the flip side equally frustrating—when agile purists simply don't understand the need for long-term, strategic enterprise planning. This is the reason enterprise agile frameworks have evolved.

There is a range of mechanisms and techniques that can be used for those outside development to become more agile in their thinking and planning. For example, project sponsors, subject matter experts and product managers can benefit by understanding what happens to their requirements once they have been thrown over the fence to the development team.

The great thing about requirements, if you are not a developer, is that they are all about the same size: one sentence. Stakeholders often have no idea how much effort an individual requirement is going to consume before it is delivered. By the time each requirement is delivered, the team probably survived a range of epics, dozens of user stories, and hundreds of tests. Reverse-engineering the implementation cost of requirements can help stakeholders understand the individual cost of each requirement. While this won't make the problem go away, it should make stakeholders less frivolous with their demands in the future and more open to negotiating an acceptable compromise during detailed analysis and design phases.

AGILE PROPONENTS BEMOAN THE FACT THAT THE CORPORATE AND PROJECT WORLD INSISTS ON SHOEHORNING AGILE INTO PROCESSES THAT ARE INHERENTLY NONAGILE.

However, stakeholders must be careful not to misuse this information, particularly in agile environments. Calculating the actual implementation cost of requirements delivered is not the preferred way to estimate the cost of each upcoming requirement, no matter what estimating techniques are used. In other words, past performance may not be an indicator of future results.

Why Agile Is Important to the Enterprise

Sure, there are agile methodologies such as Scrum, kanban, and DevOps that organizations can experiment with, using small, non-critical projects as test beds, in order to evaluate whether agile is suitable. However, there is a huge difference between having a go at agile to see what happens using a single team on one small project, compared to committing the future of an enterprise to something that nobody is really too sure about. This aversion to risk in the absence of evidence is not only rational but the correct and responsible attitude for enterprises to take.

As a result, there has been a growing realization among agile proponents that, unlike small software teams, large enterprises need a planning framework to incorporate agile development methodologies into their enterprise planning processes. This has led to a variety of enterprise-level frameworks being proposed and becoming more and more common. These frameworks, although not exactly as agile as eXtreme Programming (XP), help enterprises understand and adopt agile concepts and techniques. (Besides, project and corporate folk love frameworks!)

EFFECTIVE COMMUNICATION IS THE PRIMARY RESPONSIBILITY OF A BUSINESS ANALYST, PARTICULARLY IN AN AGILE ENVIRONMENT.

Perhaps the most well-known frameworks are the Scaled Agile Framework (SAFe) [2], Large-Scale Scrum (LeSS) [3], and Disciplined Agile Delivery (DAD) [4]. As with everything regarding being agile, adaptability is the key. If it works, it's right. Any given technique or task might be right for only one specific project, but that's the point of agile. SAFe, LeSS, and DAD offer neat, very accessible, and understandable medium- and long-term planning frameworks for enterprises while still retaining the essence of agile, especially with development teams.

Committing to Agile Where It Makes Sense

Because adaptability is one of the benefits of an agile way of thinking, adaptation should happen everywhere—as long as it makes sense. Adapt to whatever works, drop whatever doesn't, review, rinse, and repeat. Don't just adapt something for the sake of adapting something; that's not agile.

If that means projects become a little less predefined and prescribed than project managers or sponsors might like, then so be it. Projects are rarely delivered as planned anyway, so all that's lost is the stress and pressure of pretending otherwise. If, on the other hand, it means the development team should adapt and adopt something less than pure agile, this actually would be the agile thing to do.

Ensuring that business stakeholders are involved in and committed to the entire process is also critical to success. It is not enough to have project managers alone representing the project to the business from the day that implementation starts. The roles of business analyst, product owner, and product manager are needed as a bridge between the business and development teams. If these roles exist solely to insulate others from problems, they add expense with little value. Agile is all about early and appropriate feedback— both good and bad—to all project stakeholders.

As a business analyst, I deal with this every day and I appreciate and understand the different perspectives of business stakeholders and development teams. I have no doubt that the two sides are not only reconcilable but actually not that far apart. Honest, forthright, and timely communication of both good and bad news up and down the project and organizational hierarchy is key to successful implementation of agile on any given project. Effective communication is the primary responsibility of a business analyst, particularly in an agile environment. This is a topic I write regularly about as critical to the success of each and every project I lead.

Ideally, though, everyone critical to defining success should take an active role. They aren't expected to write code, as that would be asking a bit too much, but they should meet frequently with the team leads rather than just attending routine status meetings with the project manager where bad news tends to be avoided at all costs.

Everything Is Always Changing!

In any agile project, plans will change, requirements will change, budgets will change, and priorities will change. In my experience, requirements will be de-scoped or completely restated, and new requirements will be added.

However, this is as it always has been in any project work—especially in software development. Agile was never intended to solve those problems. By adapting the right mindset, there should never really be any major surprises to derail a project. Any surprises that do appear—whether from the development team or the business-oriented stakeholders—can be communicated, evaluated, and subsequently dealt with quickly and comparatively painlessly.

In short, the core philosophy of agile is to do whatever works to deliver value early. Agile is about whatever makes sense in the given scenario at the given time to realize business value. Because the definition of value differs across roles, prioritization of requirements becomes a critically important task. That's the basis of agile.

Every scenario is different, so every agile project will be different. Anyone who proposes a one-size-fits-all, silver-bullet agile methodology is either missing the point or trying to sell something. Blasphemous as it may seem to agile evangelists, "whatever works" in some cases might not actually be agile. "Adapting to whatever works" usually is. [BSM] ronhealyx@gmail.com

CLICK FOR THIS STORY'S **REFERENCES**

SOFTWARE TESTER CERTIFICATION

Professional certifications are a tangible way to set yourself apart. SQE Training offers accredited training courses for the most recognized software testing certification in the industry—ISTQB® International Software Testing Qualifications Board.

The International Software Testing Qualifications Board (ISTQB) is a non-proprietary organization that has granted more than 500,000 certifications in more than 100 countries around the globe. Certification is designed for software professionals who need to demonstrate practical knowledge of software testing—test designers, test analysts, test engineers, test consultants, test managers, user acceptance testers, developers, and more.

ISTQB Accredited Training Provider

Each of our accredited training courses go above and beyond the ISTQB syllabus, giving you practical knowledge you can apply now. All of our courses are led by instructors with an average of 15–30 years of real-world experience, meaning you can be confident that your learning experience will be second to none.

Advance your career by adding an internationally-recognized certification to your resume. Learn more about certification at **sqetraining.com/certification** or request a personal consultation with one of our dedicated Training Advocates by calling 888.268.8770.

CERTIFICATION OFFERINGS

Foundation Level Certification (CTFL)

Foundation Level Agile Extension (CTFL-AT)

ASTQB Mobile Testing Certification (CMT)

Advanced Level Test Manager (CTAL-TM)

Advanced Level Test Analyst (CTAL-TA)

Advanced Level Technical Test Analyst (CTAL-TTA)

LEARNING OPTIONS

Public P

LIVE VIRTUAL TRAINING

eLearning

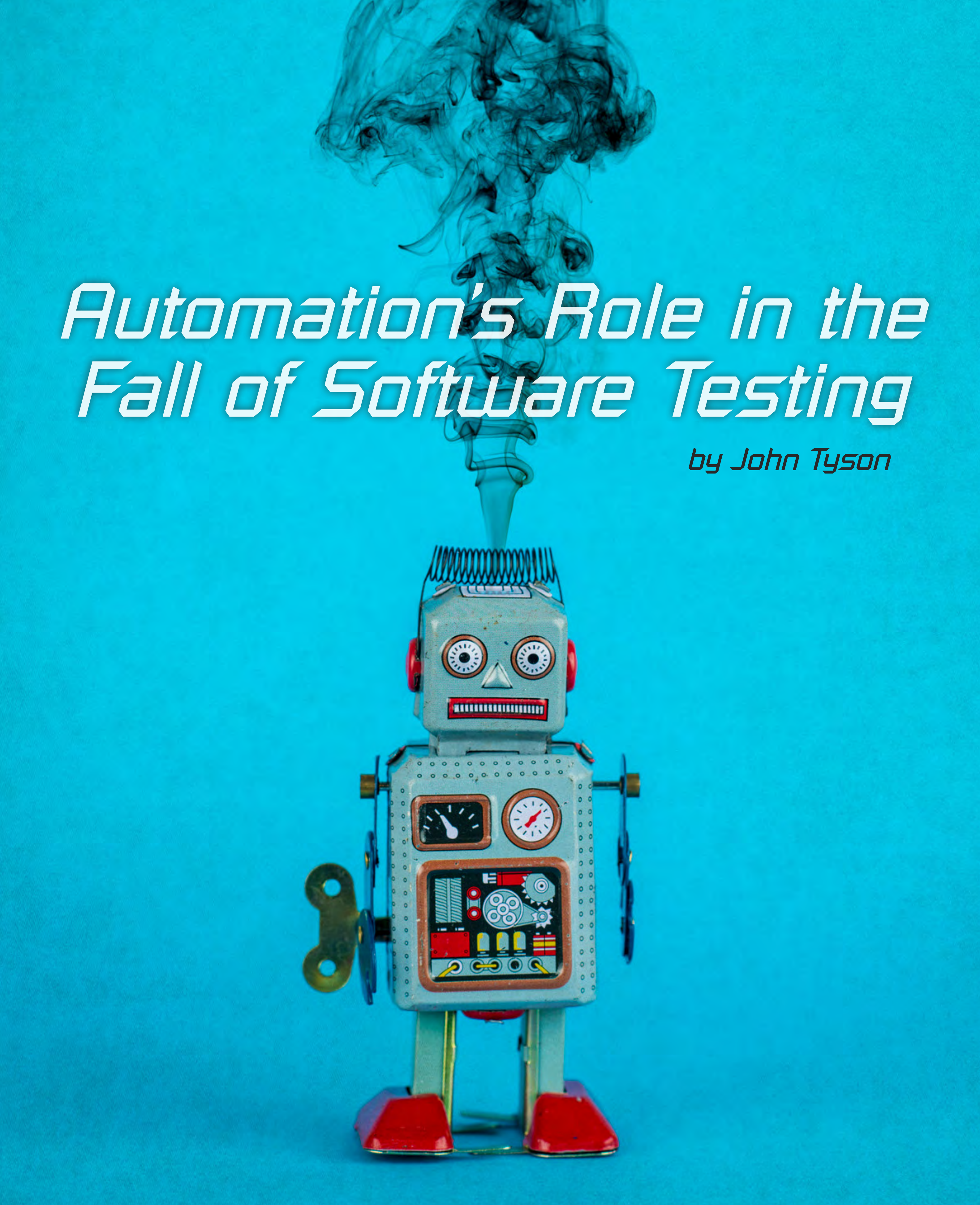
ON-SITE ADVANTAGE

SQETRAINING.COM/CERTIFICATION

SQE TRAINING
A TECHWELL COMPANY

Automation's Role in the Fall of Software Testing

by John Tyson



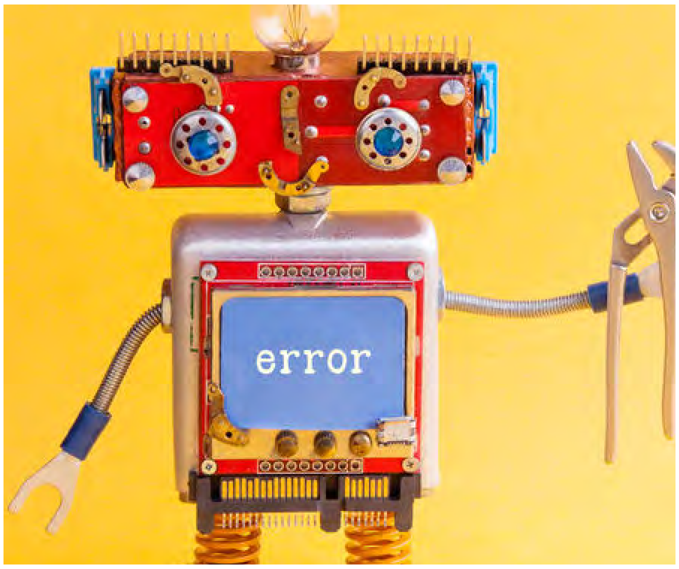
There is a trend in software testing that I don't like. Development teams prefer the speed and shallowness of automated testing over deeper, more realistic user scenario testing. Software testers used to be valued in software development for their testing skills, but the industry's obsession with test automation has now reached the point where it is believed that testers can be replaced by automated test scripts. As a result, test automation needs programmers—not testers. Professional software testers are being driven out if they cannot program test automation frameworks. This scripting leaves little time to do exploratory testing and avoids testing the user interface (UI). This trend comes at the worst possible time—just as software is becoming more complex and more critical.

By overemphasizing test automation, we risk diminishing the role of professional software testers as valued development team members. Poor quality software also impacts users, which can be in the millions or billions of people. High quality software yields a good user experience while decreasing customer support costs.

From Waterfall to RAD to Agile

When I began my software testing career in the mid-'90s, software testing had become mainstream in software development, although there were still a few problems. Testing followed the waterfall model, even on rapid application development (RAD) projects starting late into the development cycle, creating a bottleneck. Testing was often used as a dumping ground for poor programmers, perhaps indicating that testing wasn't *that* important and was more of a checkoff item.

In those days software testing was on the rise, with new tools, new techniques, and better testers. It was acknowledged that testing and developing were two very different skills not often found in the same person. Defect tracking and test automation tools started being used. Talk of testing early became common, as did discussions on testability. In addition, there was a commitment to performance and load testing, usability testing, and functionality testing.



The future was looking great for software testers. But there was a problem. All this talk of quality required investment and time. Test automation was seen as the panacea for the testing bottleneck. This came with a few warts, according to James Bach's article "Test Automation Snake Oil." [1] Software tool vendors overpromised the ease and efficacy of their record-and-playback test automation tools. Testers found their automated test scripts brittle. One small change in an application could invalidate hundreds of test scripts. People learned to mitigate these problems by taking a modular, data-driven approach to automated testing, so scripts could be reused and a change in the application would only result in changing one or two test scripts. This helped reduce the number of test scripts needed.

Record-and-playback test automation tools suffered from requiring the application to be completed before scripts could be recorded.

But the need for speed in delivering software continued to increase from waterfall to RAD and, now, to agile. Record-and-playback test automation tools suffered from requiring the application to be completed before scripts could be recorded. Worse, the scripts relied on the UI, which usually changes late and often in the development cycle, after users get to see and use their new application. To compound matters even further, the more capabilities added to test automation tools, the more expensive they became.

To remedy all of these issues, a progression of open source tools evolved, with Selenium WebDriver currently one of the most popular. Selenium eliminated the licensing cost issue, while other tools allowed developers to code tests for functions behind the UI. Even if the UI changed, test scripts would still work because underlying functions did not usually change. Now test scripts could be developed as soon as the functions became available instead of waiting for the application to become available. Ironically, moving testing earlier and making scripts more robust marked the demise of software testing as the emphasis was placed on automated script development instead of testing skills.

Should Software Testers Become Developers?

This brings us to our current situation, where test automation is still seen as the key to ensuring product quality. However, it is developers—not testers—who are needed in software testing. In job postings, the most important skill asked of testers is test automation, which translates to "Can you program?"

Testers must learn more, as it's common for testers to test an app end to end, whereas developers usually focus on one feature or a small area of an app.

Either the actual skill and mindset of testing is not considered important, or it's assumed that anyone can test.

This will have profoundly bad implications for the future of the software development industry as things like autonomous vehicles and the internet of things become commonplace. Replacing testers with developers may not have started out as an intentional act, but it has definitely become the trend. The people writing automated tests, especially using a behind-the-UI approach, usually have development skills, so programmers are the natural choice.

Some might say that programming tests with code or scripts is just another new skill that testers need to learn. This skill can be a positive thing, allowing testers to move out of testing and into software development, where far more jobs are available and pay—and respect—is higher. But there are many downsides to requiring testers to become developers. Let's look at the problems.

Automating tests usually implies shallow testing. Test automation has been criticized for falling short of expectations. Not all tests can be automated—some due to technical limitations (like timing issues), others due to economic considerations. It's just not feasible to automate some tests. This means your automated test suite will be a subset of what can be tested. Chances are good it will be a shallow, happy-path regression test suite that does not cover complex scenarios, especially if the tester has to complete the test within a single sprint.

Test automation usually takes priority over deeper exploratory testing. This is especially true if the automated tests are included in the definition of done criteria.

Testing is conflated with checking. Much has been written about the difference between testing, which requires a highly cognitive skill to perform, and checking, a static set of steps that machines execute. [2] James Bach and Michael Bolton quote philosopher Marshall McLuhan, writing, "We shape our tools, and thereafter our tools shape us." [3] They also present an analogy: "We may witness how industrialization changes cabinet craftsmen into cabinet factories, and that may tempt us to speak of the changing role of the cabinet maker, but the cabinet factory worker is certainly not a mutated cabinet craftsman." Here, the factory worker acts as a checker, operating a machine, while a craftsman acts as a tester, choosing not to use damaged or poor-quality wood, investigating why the cabinet wasn't manufactured correctly.

Checking, instead of testing, also may suffer from the pesticide paradox. Just as insects eventually build up resistance to a pesticide, repeatedly testing using the same data and the same steps will most likely miss defects that different data or different steps would uncover. [4]

There is never enough time. Testers are perpetually in crunch mode. Part of this is due to the nature of testing's infinite workload versus development's finite workload. Developer workload decreases as coding is completed, while tester workload increases as there are more features to be tested as the deadline approaches. Testers must learn more, as it's common for testers to test an app end to end, whereas developers usually focus on one feature or a small area of an app.

Another factor affecting the time available for testing is the number of developers whom testers support. In my experience in agile development projects, I've typically supported four to six developers. For testers to understand how developed features work, there is a need for documentation and knowledge transfer from developers. Because agile deemphasizes the need for documentation, it is often easier to wander off and seek clarification from the product owner consuming more of the tester's time on non-testing tasks.

The reality is that user stories or task descriptions are rarely updated. If the tester isn't informed, time can be wasted testing something that doesn't need to be tested. If the tester supports several developers, they may have to work at an unsustainable pace, violating a key agile principle.

Determining whether development is complete isn't easy. Does your team have a development freeze deadline? I've attended sprints where developers deliver code on the last day of the sprint, severely limiting the testing that can be done. Even worse is when the manager wants testers to do end-of-sprint demos.

Testers are always a scarce resource. Is it easier to find and hire a tester or a developer? Due to the lack of formal testing curricula, there is a wide disparity in software testers. Some have programming backgrounds, while others have no actual testing experience at all. This doesn't mean one is better than the other, but it makes hiring a good tester difficult. Then there are the testing specialties, where a tester may be great at usability testing and very poor at functional testing. Good testers should be able to tell you what they're good at testing and, just as importantly, what they're not good at testing. In addition to good testing instincts, they need to be honest, have integrity, possess a strong work ethic, and be good communicators. This is difficult to evaluate in an interview.

Developers are easier to find, qualify, and hire. Why would you want to throw away a talented software tester with special, hard-to-find skills by requiring them to become a programmer?

UI testing gets neglected. If test automation is replacing human testers and your application is used by humans, who is testing the UI? Who is getting feedback on the user experience? If the first-time users get to see the app is during user acceptance testing, you will have a disaster on your hands. Remember how UI changes are made often and late in development?

If professional testers aren't looking at UI and usability issues early, there will be problems. Testers are often viewed as end-user advocates within the development team and the link between users and the development team.

With UI testing being bypassed, would automated test developers fill this role?

Software development teams need both professional software testers and test automators.

There is a loss of testers. Besides not developing additional testing skills and application knowledge, this introduces a new problem—the perpetual loss of testers. [5] Testing will be seen as an entry-level, temporary position. No testing expertise will be built up, so apps are likely to be poorly tested.

The New Composition of a Software Development Team

At a time when more thorough testing is needed, we are getting more frequent, shallow testing using test automation. Talk of new, more comprehensive testing techniques has all but disappeared. Quality-tested software products are an absolute requirement in the software application industry. We need to recognize the strengths and weaknesses of automated and exploratory testing, using the strengths of each while avoiding the weaknesses. We need to acknowledge testing as a special skill required for all development projects.

Test automation is needed. Manual regression testing is terribly boring, slow, and error-prone. As new or changed features are delivered, automated regression tests are valuable in confirming that old functionality still works correctly. It also allows for functions to be tested early, providing feedback sooner and reducing the number of bugs found later. Software development teams need both professional software testers and test automators. This needs to become the de facto standard for software development.

Professional software testers will perform exploratory testing and dig deep. They will test things that automated tests can't catch, like timing issues, UI issues, complex scenarios, and things that aren't economically feasible to automate. Software development teams of tomorrow need both test automation specialists and software testing specialists. **[BSM]** jmt_research@yahoo.com

CLICK FOR THIS STORY'S **REFERENCES**

Building Autonomous DevOps Capability in Delivery Teams

by **Miiro Juuso**

Embarking on the DevOps journey is difficult, and there are lots of opportunities to get it wrong along the way. But like most things in life, taking risks can yield great rewards.

Recent studies show that enterprise software organizations with established DevOps ways of working consistently deliver better software. A software team's ability to reliably deliver value rapidly translates directly to the success of the business. DevOps capability has become a differentiator that sets the most successful technology companies apart from the rest.

There are as many ways to implement DevOps as there are teams implementing it. While the DevOps movement is still relatively new in the world of software engineering, we are starting to recognize which approaches work well—and which do not.

In the past few years, organizations eager to jump on the DevOps bandwagon have invested in dedicated DevOps teams. These teams are generally given ownership of delivery pipelines, setting up monitoring, and manning the on-call shifts. They sit in the twilight zone between developers and system administrators, applying a mixed set of skills to fix problems that do not unequivocally belong to either development or operations. These DevOps teams are given the mandate to implement continuous delivery and, in some cases, to ensure that changes are released multiple times a day. This is a distinct difference from the big and scary releases in the past. Sometimes DevOps can be so fluid that the team may not even notice frequent releases.

Nothing Comes for Free

Just as some things are too good to be true, there are a number of problems with this approach. While a DevOps team is often brought in to bridge the gap between software development teams and infrastructure operations teams, it nearly always ends up being an island. Instead of pushing releases over to system administrators for deployment, the organization now relies on the nearly magical DevOps team to make sure code reaches production.

Instead of breaking down silos, the DevOps team ends up creating one of their own.

One of the consequences of this is lack of accountability. When teams are asked what failed in a release, it's either application problems, build pipeline problems, or environment problems. The challenge is that issues in modern software delivery often span two or three of these areas, and all of these areas have their respective owners. When a problem doesn't have a single owner, fixing it becomes much more difficult, regardless of shared responsibility models that might have been implemented.

Another problem is that a silo of people almost always results in a silo of knowledge. The traditional DevOps problem remains unsolved when all this knowledge is concentrated within the DevOps team. Software developers may not know anything about the infrastructure their application runs on, while system administrators are none the wiser about the applications they are hosting. Instead of improving communication and propagation of knowledge across the organization, an isolated DevOps team can inadvertently hinder both by simply being another link in the chain.

Where DevOps Approaches Fall Apart

There is no established best practice for a framework of how an isolated DevOps team should work. In my experience, approaches differ wildly. Some teams use kanban while others work in sprints. All of them struggle to balance planned and unplanned work (commonly called "firefighting"). Coupled with the fact that DevOps teams rarely contribute to or even attend delivery team planning sessions, the amount of unplanned work can be quite large.

Similarly, when software development functions scale, the number of delivery pipelines grows—and so does the amount of firefighting. The ensuing reactivity instead of proactivity is challenging to manage from a leadership perspective and easily leads to a state where whoever shouts the loudest gets their request fulfilled.

A dedicated DevOps team like this can often be better described as an automation, pipeline, or infrastructure engineering team—and it suffers from the same dysfunctional, reactive workflows as any external team. Apart from implementing automation, these teams do very little to advance the key benefits of a DevOps culture: reduced external dependencies, improved delivery velocity, and improved communication. Overall, these are factors that improve the organization's maturity in continuous delivery—the ability to ship small increments of change rapidly and consistently to production. There are three concepts that are critical to realizing the true potential of DevOps: delivery team autonomy, consistency through empowerment, and the DevOps teacher model.

Delivery Teams Must Be Autonomous

Instead of building isolated DevOps teams, improving autonomy of delivery teams is a much better approach. However, it does come with its own complications: Upskilling software engineers takes investment, and the initial transformation will adversely impact delivery velocity. But these downsides are easily offset by reduced external dependencies, added confidence in continuous delivery, and improved delivery velocity. The more changes that can be released to production without relying on other teams, the better.

Ultimately, DevOps capability should be seen as a *feature* of every delivery team instead of the *function* of a dedicated team. The best way to improve the autonomy of a delivery team is to enable them to own their continuous delivery pipeline—from development through production. This means that the delivery team will need to learn new tricks. Instead of offloading the responsibility to another team, the DevOps team must take ownership of some operational aspects themselves.

Increased autonomy of a delivery team does not mean they should maintain their own kernel patches or reinvent the wheel every time they need to build a blue/green deployment model. There is still a role for system administrators and infrastructure engineering teams. They should concentrate on building frameworks and automation so that the delivery teams can concentrate on releasing changes to production. Defining a deployment framework and a managed platform as a service is a great place to start. The good news is that managed services on modern cloud platforms can make this relatively trivial with tools like Elastic Beanstalk on Amazon Web Services. Using a platform abstracts away low-level tasks and enables its users to focus on delivering application features.

Consistency through Empowerment

One of the underlying causes of excessive external dependencies is a control mindset, commonly introduced as an attempt to bring order to chaos. Typical symptoms of this mindset are system administrators not giving access to application servers or QA requiring manual testing of each and every change.

There are better ways to approach this need for consistency. I have yet to meet a software engineer who deliberately makes bad decisions, instead of making decisions due to lack of knowledge or overall context.

Empowering teams to make the right choices yields better results than imposing strict rules. This can be accomplished by continuously upskilling the teams, providing tools that encourage best practices, and making sure the team knows how the thing they're building fits in the bigger picture.

From a DevOps perspective, tooling is a natural place to start. The teams can, for example, be provided a boilerplate for a build and deployment pipeline that encourages automated testing and deployment to an environment running on a modern cloud platform.

To continue on the concept of delivery team autonomy, a practical example of empowering delivery teams to work autonomously and consistently is building self-service automation for completing tasks that were previously manual and time-consuming. By automating the management of testing environments, the teams can create environments whenever they need one.

Sometimes tools will need to bend for the rules. The regulatory environment might, for example, impose boundaries on how software can be delivered. It is important to identify the hard limits and design the tooling and processes to accommodate them. If a delivery team needs business signoff before making every feature live, implementing feature switches in their workflow could still enable fully automated continuous deployment.

By automating the management of testing environments, the teams can create environments whenever they need one.

The DevOps Teacher Model Works

Upskilling teams requires planning and resources, and pending major breakthroughs in AI, self-service interfaces do not build themselves. There is a clear requirement for dedicated DevOps enablement teams who don't own delivery pipelines. Instead, delivery teams should be empowered to own them. DevOps teachers should sit within the delivery teams and work within their backlogs as a member of the delivery team.

First and foremost, a DevOps teacher's objective is to enable the team to own their delivery pipeline by upskilling and coaching team members. When a DevOps teacher does technical hands-on delivery work, they make sure another team member is equipped to accomplish the same task in the future. Further, the complete delivery team must be in a position to support any systems built.

As full-fledged delivery team members, DevOps teachers should attend agile ceremonies along with the rest of their team. With active attendance in the planning stages of new work, DevOps teachers can work proactively and are in a unique position to promote best practices in testing, deployment, and monitoring. Similarly, when actively participating in retrospective meetings, the teachers are able to drive continuous improvement of the delivery pipeline.

DevOps teachers are a great vehicle for cross-pollinating knowledge across a wider digital delivery function. Teachers should meet regularly to discuss blockers and dependencies within their individual teams.

I've found that a stand-up once or twice a week works well, depending on the maturity of the teacher role. Secondarily, the teachers should be rotated every three to six months. This enables the delivery teams to learn from each other and ensures that teachers are able to build their knowledge across all delivery teams' products.

The natural question is how to transform a DevOps team into a team of DevOps teachers. When a DevOps team is accustomed to owning delivery pipelines, the change in paradigm can be challenging. The shift from "doing things" to "teaching others how to do things" is always difficult. The reality is that some people are not able to make the leap. This reminds me of an adage: "If you can't change the people, change the people."

An established, isolated DevOps team might find a better role as an infrastructure engineering team, with a new embedded DevOps teacher team working toward the shift of pipeline ownership.

The Proper Role for DevOps

When we start looking at DevOps as an enablement function and an instigator of change instead of a team that owns delivery pipelines, we can realize the true potential of the DevOps movement. There is a place for dedicated DevOps professionals in modern software delivery functions, but it is not a pure engineering role that takes sole ownership of delivery pipelines.

With thousands of organizations worldwide looking to hire DevOps engineers, we should consider the DevOps engineer a natural phase in the evolution of a software delivery organization.

Treating DevOps as a way of working promotes a culture of autonomous delivery teams that have full responsibility for the success of their digital products. Ultimately, this translates to happy customers—and who doesn't want that? **[BSM]** miiro.juuso@gmail.com

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!

3 Major Continuous Delivery Hurdles Teams Need to Overcome

By Josiah Renaudin

Teams that leverage continuous delivery and continuous integration are playing an entirely different game than software teams of the past—instead of mapping out this major, ridged timeline, data is being both gathered and used more frequently (and effectively) than before.

Read More

Using Feature Flags to Boost Testing and Deployment

By James Espie

A feature flag is a configuration setting that lets you turn a given feature on or off. There is no need for a feature to be complete before you can start testing—as soon as the first piece of code is merged, you can turn the flag on in your test environment and begin. This also reduces risk.

Read More

The Need for Well-Formed, Creative Minds in Software Testing

By Rajini Padmanaban

The need for creativity and innovation is felt in the world of software testing more than ever before given how dynamic and fast-paced it has become. With so many changing technologies and a multitude of people to interact with, a tester's job calls for newer and better ways of accomplishing tasks.

Read More

What We Talk about When We Talk about Test Automation

By Justin Rohrman

Testers talking about test automation often mean browser automation. Developers are probably talking about unit testing or something at the service layer. And operations people are most likely thinking of monitoring and the guts that control continuous integration. But the practices are more important than terminology.

Read More

6 Major Challenges of Cloud Computing

By Ray Parker

Companies of all sizes depend on cloud computing to store important data. However, significant factors such as cost, reliability, and security must not be overlooked. Here are six common challenges you should consider—and develop plans to mitigate—before implementing cloud computing technology.

Read More

Insider Threats: What's the Biggest IT Security Risk in Your Organization?

By Pete Johnson

Any modern company should give the line-of-business teams the ability to provision self-service, on-demand resources, but to ensure security, you have to do so in a way that has the necessary monitoring built in via automation. One good way is to use a cloud management platform that helps you keep your app secure.

Read More

Breaking the Cycle of Bad Scrum

By Ryan Ripley

When practiced well, Scrum can empower people, teams, and organizations to solve complex problems and deliver value to their customers. But bad Scrum does the opposite. If team members or leaders don't embrace Scrum values, it can be oppressive and create tension. Here's how you can prevent bad Scrum from taking hold.

Read More

Why Frequently Delivering Working Software Is Crucial to Agile

By Jeffery Payne

While completing documentation is often an indication that some progress has been made, until software has been implemented, tested, and approved by a customer, the amount of progress cannot be measured. Here are some common reasons agile teams fail to frequently deliver working software—and how to avoid them.

Read More

Measuring Objective Continuous Improvement in DevOps

By Logan Daigle

When you're beginning your DevOps journey, it is incredibly important to know where you are starting. You will want to know later on what progress you have made, and you won't be able to figure that out unless you have benchmarks from the beginning. Here are six steps to objectively measure your continuous improvement.

Read More

Troubled Project or Disaster? Understand What You Can Manage

By Payson Hall

There is a big difference between a troubled project and a disaster, and not being clear about the distinction is hazardous to decision-making. If a project you're managing is in danger of missing deadlines, that doesn't mean it's out of control—you just need to explain to stakeholders how it can get back on track.

Read More

How You Can Help the Human Animals in Your Group Thrive

By Isabel Evans

On our teams, we deal with many individuals with diverse perspectives. It's not always easy, but we are animals, and many animals live and work—and are only able to survive—in teams. You can look to how animals interact with and react to each other to see how we, as human animals, can not just survive, but thrive.

Read More

Transforming Your QA and Test Team

By Sophie Benjamin

Testing professionals are essential to the success of technology projects. Delivering better, faster, and at a lower cost is not solely done with automation and development teams—testing professionals are here to stay and grow. But we have to fight for our place, and that means evolving with industry requirements.

Read More

FDA Pilots Program to Pre-Certify Digital Health Software

By Pamela Rentz

As healthcare undergoes a digital transformation, how can the traditional regulatory process keep pace? The FDA recently announced the initial participants in a pilot program that will pre-certify digital health tech companies that meet quality standards for software design, validation, and maintenance.

Read More

Performance Testing for Our Modern, DevOps World

By Paola Rossaro

As DevOps-based methodologies are more broadly adopted, we'll increasingly move to a continuous testing model. Containerized environments and microservices make it easier to optimize your application by validating changes to the environment or system configuration, allowing you to deliver better products faster.

Read More

Balance Technical and Social Skills for Project Success

By Marcia Buzzella

Software testing is a socio-technical undertaking, which means that effective test strategies must incorporate a balance of technical capabilities relating to processes and tools and social capabilities used for communication and problem-solving. This balance enables true project success.

Read More

Use Continuous Backlog Grooming to Refine Agile Requirements

By Susan Brockley

Continuous backlog grooming means systematically refining your user stories: breaking up larger stories, obtaining detailed requirements, writing the requirements in terms of acceptance criteria and acceptance tests, and sharing and refining these details with the team. Acceptance test-driven development can help.

Read More

Adopt an Innovative Quality Approach to Testing

ALTHOUGH IT IS IMPRACTICAL THAT EVERY TEST CONDITION CAN BE VALIDATED, INTRODUCE INNOVATIVE TEST PRACTICES ALONG WITH TESTING IN PRODUCTION.

by Rajini Padmanaban | *rajini.padmanaban@qainfotech.com*

Most testers are stuck in time with testing practices and approaches that are not aligned to today's needs. The product landscape has changed significantly in the past few years, and the use of technology is no longer for the few. This is especially true in the software world, where the impact of quality is significant. There are now approximately twelve billion users, processes, and devices connected to the internet. This is expected to reach fifty billion connections by 2020. And by 2022, a majority of people under the age of twenty-five will be using some sort of digital device or service. [1]

This makes an innovative quality approach more important than ever to ensure that your software product is truly ready for the marketplace. Innovative quality does not necessarily mean doing something drastically different. It means:

- Creating a quality strategy where the status quo is constantly questioned and evaluated for ongoing continuous improvement in testing, even after an app goes live
- Aligning the quality efforts with the needs of the day, including user-centric quality and equal focus on non-functional test attributes
- Dialing up (or down) an exploratory test effort and comparing outcomes to the overall quality goals
- Demonstrating a willingness to learn from users, competitors, and the industry at large while contributing what you've learned back to the quality community

Although it may not always be true, it is my experience that every team works within time, cost, and resource constraints. This is especially true for testers, and it often comes down to the question, "Just how much testing is enough?"

Not every scenario that needs to be tested can be accommodated ideally before release. Combine that with outdated testing practices, and the gap between what end-users expect and the actual product is only going to widen.

A better approach must bridge this gap by testing in production and by introducing innovative test practices.

The Importance of Testing in Production

Regardless of how well an application is tested before it goes live, a team cannot ignore validating under a production scenario. Traditionally we have known production testing to include monitoring app availability and performance by injecting automated tests, simulating a live end-user or programmatic usage. These automated tests are created by the test team and largely used by the support teams. This is changing with the use of analytics tools that make monitoring easy, reliable, and quick.

The test team needs to play a very active role in monitoring feedback—both proactively and reactively—in the live environment.

However, there is more to testing in production. The test team needs to play a very active role in monitoring feedback—both proactively and reactively—in the live environment.

A formal approach to testing in production should include passive monitoring with real data, active monitoring with synthesized transactions, experimentation with real users, and a workload to simulate live system stress. If production testing isn't done correctly, the result can be a double-edged sword— wasting resources and the team's effort in a direction that doesn't yield productive results. [2]

Testing in production is often mapped to merely look at user feedback and identify what hot fixes or product recommendations can be taken up in future releases.

But how many projects allow the time and effort for the team (and testers) to play in the live environment is a big question. This kind of active monitoring in live scenarios, with synthesized transactions simulating end-users, is equally important.

The product's test strategy should try to accommodate any huge mishaps, allowing the tester to immediately switch to a nonlive environment to debug further. Similarly, an ongoing team of beta or crowd testers (including actual end-users) is valuable to keep the testing in production taking place on a formal basis.

Live testing can be a rough zone, especially when dealing with real end-user data, carrying out tests in the space of performance, and security that may render the system vulnerable to attacks and downtimes. However, live production testing has tremendous benefits that outweigh any risks, if the effort is bifurcated enough to allow specific tests to be run in each environment.

As software products vary greatly, every organization should create a custom strategy that works best for them, based on their product, its maturity, user base, and competition—with a specific focus on what testing should be performed in production.

Rethink and Innovate Your Testing

Innovation often implies rethinking past approaches, but if not clearly defined, an innovatve approach can be too broad, vague, and difficult to gain team buy-in. Limiting innovation to such a high-level view often falls short of a clear implementation strategy, causing the team to understand the need to innovate but not knowing where to start.

Periodic meetings, such as "think week" programs, to brainstorm with the quality team can help focus on specific areas of innovation and gain team support.

Based on my experience, there are a few ideas that will further promote innovative testing by aligning with user needs while bringing in deep technical focus.

Adopt device performance testing. When the testing focus is on performance, application-level testing may not be enough. Sometimes, poor performance may be due to the app's interaction with the device or other apps on the device.

In this case, use tools that track device performance and use data alongside core performance test parameters.

Security testing needs to be more robust. Most of us assume security testing to be synonymous with the Open Web Application Security Project (OWASP) top ten vulnerabilities testing. While this is a good start, there is a clear need to dive deeper into network and web services layers. The good news is that OWASP has guidelines to achieve this, and a lot of open source tools (like FuzzAPI) are available.

Make your app accessible by everyone. For many industries that need to conform to the US Government's Standard 508, accessibility test efforts' outcomes are still mapped using a voluntary product accessibility template (VPAT). The VPAT is a very reliable checklist, but does it give you a good gauge on your accessibility baseline? Do your executives fully understand the VPAT and have time to review it to determine what's next in the space of accessibility? A better approach would include an objective indicator that shows the accessibility compliance score. Using a scorecard can not only track progress but also help plan for future release.

Think globally. Designing a robust localization test effort can be challenging. It usually requires manual testers, local language experts, and language translation tools. To complicate matters further, apps may have to be localized to handle complex locale dialects. As more applications reach out to larger markets around the world, innovation around localization testing and optimization becomes critical.

Testers should think like engineers. While there is a lot of emphasis on test automation today, becoming technical is not just about automation. There are several other things a tester can and should do in his path to retaining his independence and at the same time becoming more technical and valuable to the organization. A manual tester can first take the path to becoming more technical with F12 developer tools and plug-ins. These include products like PageSpeed, Axe, CSSViewer, and Instant Translate that enable many functional and non-functional quality checks. Whatever tools are selected, they should relieve the need for endless cycles of monotonous manual testing.

What to Do Next

As part of the continuing journey of improving software testing, my company has explored several ideas to adopt production testing and to innovate. We recently had an internal company conference where several of these topics were covered in detail. [3] As an active test evangelist myself, I can confidently say that these sessions helped my company improve how we test. Hopefully, you'll be inspired to have similar meetings in your company. [BSM]

CLICK FOR THIS STORY'S REFERENCES

CAN'T ATTEND A TECHWELL CONFERENCE?

WE'VE GOT YOU COVERED!

Check out the TechWell Happenings YouTube Playlists.

Hundreds of interviews, lightning talks, and STAR*EAST*, STAR*WEST*, and *Better Software* conference presentations are grouped by topic, so it's simple to take control of your learning experience.

Covering software testing and development topics ranging from mobile testing to enterprise-level agile development and pretty much everything in between, TechWell Happenings Playlists deliver expert-level knowledge directly to you, for free, whenever you want it.

Visit **well.tc/TWHapps** to subscribe to the **TechWell Happenings Channel** so you won't miss out on the newest interviews and TechWell conference presentations.

LINK TO OUR ADVERTISERS

Agile Dev, Better Software & DevOps West	2
Agile Testing Days USA	4
Ranorex	3
SQE Training—Live Virtual	17
SQE Training—Software Certification	22
STAREAST	12

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. Entire contents © 2018 by TechWell Corporation, 350 Corporate Way, Suite 400, Orange Park, FL 32073 USA 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.