SUMMER 2018

BETTER SOFTWARE™

A TECHWELL™ PUBLICATION

GREAT BIG AGILE

An OS for Agile Leaders

All Customers Are Not Created Equal

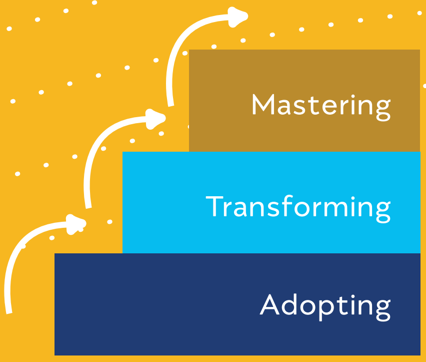
Learn how to improve your skills to improve customer satisfaction

Supporting Continuous Testing with Automation

DevOps won't be the same after you master the art of test automation

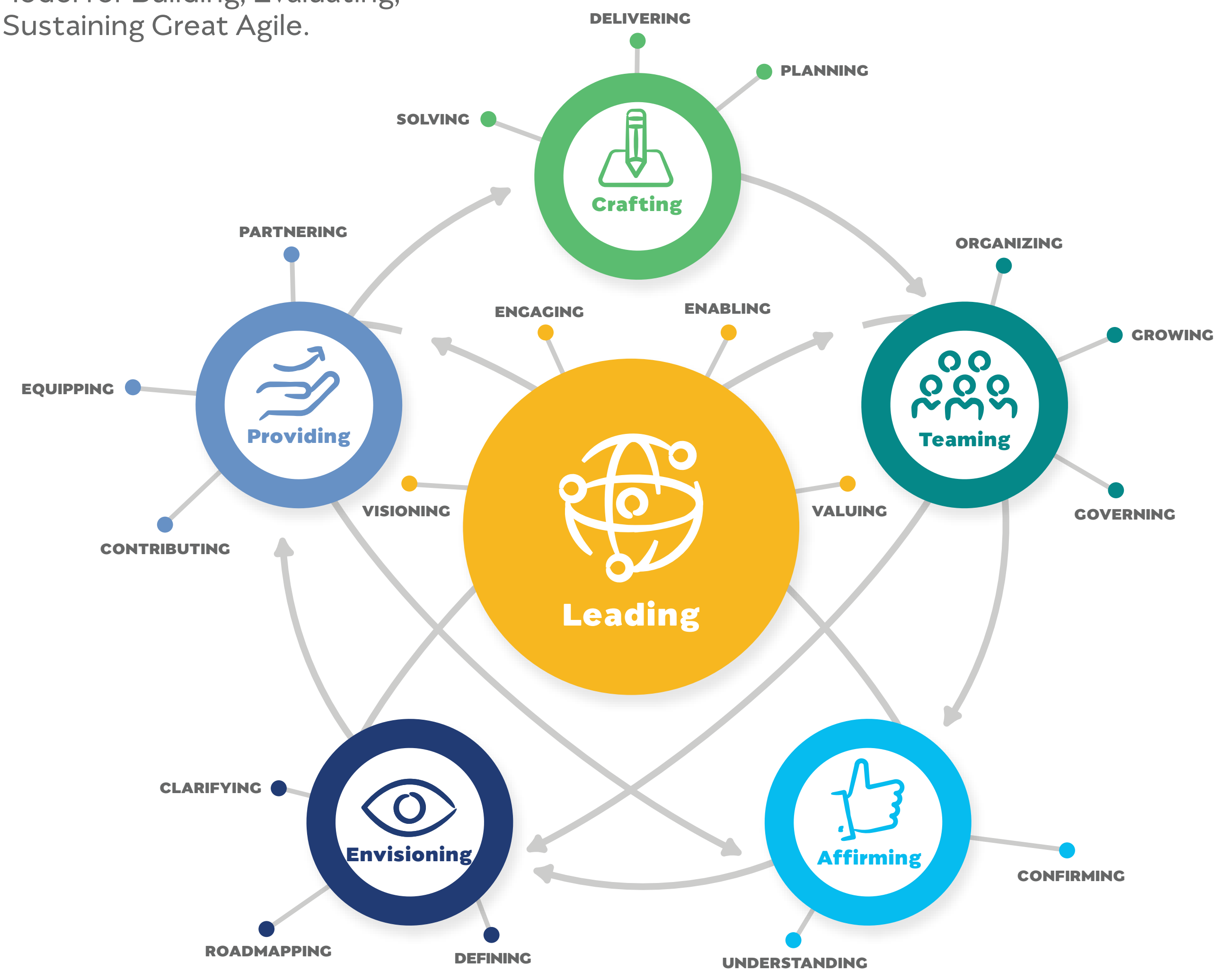
Agile Leaders Know that Agile Performance

Is Critical To Their Success.



Organizations can be certified in three levels to describe their state of agile performance

The Agile Performance Holarchy is a Model for Building, Evaluating, and Sustaining Great Agile.



Contact us at agileleader@agilecxo.org if you are:

- Interested in benchmarking and improving organizational agile performance
- A company that is interested in expanding your business by providing APH assessments, training, and coaching solutions

AgileCxO .org

GET
INSPIRED
AT THE
PREMIER EVENT
FOR
SOFTWARE TESTING
PROFESSIONALS

STAR WEST

A TECHWELL EVENT

SEPT. 30–OCT. 5, 2018
ANAHEIM, CA

CLICK TO LEARN MORE

SOFTWARE DEVELOPMENT IS EVOLVING...

SO ARE WE.

INTRODUCING

Agile + DevOps East Conference 2018

WE'VE COMBINED THE MOST VALUABLE LEARNING EXPERIENCES OF AGILE AND DEVOPS INTO ONE SPECTACULAR EVENT

- Agile and DevOps Leadership
- Agile Engineering Practices
- Agile Testing and Automation
- Building Agile and DevOps Cultures
- Agile and DevOps Certification
- DevSecOps
- Continuous Integration Practices
- Continuous Delivery/Deployment Automation
- Scaling Agile and DevOps Capabilities
- Digital Transformation

WHO SHOULD ATTEND:

- CIO, CISO, and IT directors
- Product managers
- Project managers / Scrum Masters
- Software engineers
- Test & QA managers
- Development managers
- Release managers
- Software security professionals
- Operations staff
- Release engineers
- Requirements & business analysts
- Software architects
- Measurement & process improvement staff

Agile + DevOps EAST

A TECHWELL EVENT

NOVEMBER 4–9, 2018
ORLANDO, FLORIDA
RENAISSANCE ORLANDO AT SEA WORLD

agiledevopseast.techwell.com

BSM SUBSCRIBERS SAVE UP TO $200 OFF* WITH PROMO CODE BSME18

*OFFER ONLY VALID ON NEW REGISTRATIONS OF PACKAGES OVER $800

INSIDE

Volume 20, Issue 3
SUMMER 2018

On the Cover

Features

Columns

Departments

BETTER SOFTWARE™ A TECHWELL PUBLICATION

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

August 20–22, 2018
Jacksonville, FL

August 21–23, 2018
Boston, MA

August 21–23, 2018
Phoenix, AZ

September 17–19, 2018
Raleigh-Durham, NC

September 18–20, 2018
Houston, TX

September 24–26, 2018
Washington, DC

DevOps Week

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

September 17–21, 2018
Chicago, IL

Software Testing Training Week

https://www.sqetraining.com/training-events/training-week

September 17–21, 2018
Raleigh-Durham, NC

September 24–28, 2018
Washington, DC

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 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 + DevOps EAST
A TECHWELL EVENT
Nov. 4–9, 2018
Orlando, FL
LEARN MORE

STAR EAST
A TECHWELL EVENT
Apr. 28–May 3, 2019
Orlando, FL

Agile + DevOps WEST
A TECHWELL EVENT
June 2–7, 2019
Las Vegas, NV

EDITOR'S NOTE

Stepping Back into History

Sometimes it can be fun to look back at history. *Better Software* magazine has provided innovative and pragmatic information for the software development and testing community for more than twenty years. TechWell has focused on finding the best articles that can benefit its readers over the long term. By avoiding deep dives into specific technologies that could easily become outdated with new innovation, we've concentrated on principles that will serve you well beyond the latest fad.

Better Software articles will help you create—no surprise— *better* software! The magazine started out as *Software QA* magazine in the mid-1990s with a focus on testing. The magazine was renamed to *Testing & Quality*, then to *Software Test & Quality Engineering* before finally evolving into *Better Software*. The design of the magazine has definitely changed along the way:

Whether you are a tester, developer, project manager, or another seasoned software professional, *Better Software* strives to help you improve the delivery of quality software products and services.

In this issue's cover article, "Great Big Agile: An OS for Agile Leaders," Jeff Dalton gives us something to think about in terms of how an agile approach shapes a company's culture. Regina Evans has an urgent call to arms on how to engage with customers in "All Customers Are Not Created Equal." For those of you who test for a living, you'll enjoy Amanda Perkins's article, "QA Is More Than Being a Tester." Over the past few years, we've focused on automation and continuous processes. But just because you use automated tests, it doesn't mean that product quality improves. To round out our feature articles, you'll learn three ways to avoid major test failures in Bas Dijkstra's "Supporting Continuous Testing with Automation."

In our regular "Technically Speaking" column, Mike Sowers offers some creative ways to bake quality into your product in "Using Agile and DevOps to Achieve Quality by Design." If you are looking for some career advice, Bonnie Bailey states the importance of expanding your knowledge in "Be Indispensable: Cross-Train like a Testing Athlete."

Thanks for helping us celebrate over twenty years of *Better Software* magazine. Enjoy this issue, folks.

Ken Whitaker
kwhitaker@techwell.com
Twitter: @Software_Maniac

BETTER SOFTWARE
A TECHWELL PUBLICATION

FOLLOW US

PUBLISHER
TechWell Corporation

PRESIDENT
Alison Wade

DIRECTOR OF PUBLISHING
Heather Shanholtzer

Editorial

***BETTER SOFTWARE* EDITOR**
Ken Whitaker

ONLINE EDITOR
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 MANAGER
Cristy Bird

MARKETING ASSISTANT
Allison Scholz

CONTACT US

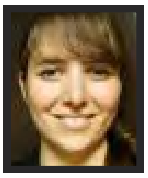
EDITORS:
editors@bettersoftware.com

SUBSCRIBER SERVICES:
info@bettersoftware.com
Phone: 904.278.0524,
888.268.8770
Fax: 904.278.4380

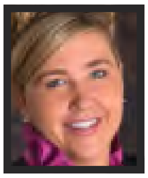
ADDRESS:
Better Software magazine
TechWell Corporation
350 Corporate Way, Ste. 400
Orange Park, FL 32073

TECHWELL

CONTRIBUTORS

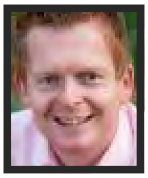


Bonnie Bailey Bonnie Bailey is a software developer for an emergency services information technology company. Bonnie is an avid reader of science fiction and books on investing, the future of humanity, and software engineering. Bonnie can be reached at bonnie.bailey@motorolasolutions.com.



Jennifer Bonine is the VP of global delivery and solutions for tap|QA Inc., a global company that specializes in strategic solutions for businesses. Jennifer began her career in consulting and implementing large ERP solutions. She has held executive-level positions leading development, quality assurance and testing, organizational development, and process improvement for Fortune 500 organizations. Contact her at jbonine@tapqa.com.

Jeff Dalton is chief evangelist at AgileCxO, a research and development organization that studies organizational agile performance. He is an author, conference speaker, blogger, coach, and technology leader with more than thirty years in the software development industry. Jeff is author of "A Guide to Scrum and CMMI: Improving Agile Performance with CMMI." In his spare time he plays jazz bass in live bands and on recordings. Jeff can be reached at jdalton@agilecxo.org.



Bas Dijkstra is an independent professional helping teams and organizations improve their testing efforts through smart application of tools. He is also a trainer on various subjects related to testing and automation. Living in the Netherlands, Bas likes running, reading a good book, and listening to good (mostly classical) music. Contact Bas at bas@ontestautomation.com.

As an engineering program manager at NetApp, **Regina Evans** has been working in IT for more than fifteen years. Most of her experience has been working with customers in software development and test engineering. In addition to being a customer evangelist, Regina enjoys the beautiful state of North Carolina with her husband and two children. You can reach her at Regina.Evans@netapp.com.

Amanda Perkins has six years of experience as a quality analyst. With experience in customer service, business analysis, product ownership, and automated testing, she brings a unique mindset to test approaches. A constant student, Amanda is learning all there is to know about cyber security testing approaches. Amanda can be reached at agiese79@gmail.com.

Mike Sowers has more than twenty-five years of practical experience as a quality and test leader of internationally distributed test teams across multiple industries. He is a senior consultant who works with large and small organizations to improve their software development, testing, and delivery approaches. He has worked with companies including Fidelity Investments, PepsiCo, FedEx, Southwest Airlines, Wells Fargo, and Lockheed to improve software quality, reduce time to market, and decrease costs. Reach Mike at msowers@techwell.com.

Using Agile and DevOps to Achieve Quality by Design

INSTEAD OF TESTING FOR QUALITY ON A FINISHED PRODUCT, HERE ARE THREE APPROACHES TO BUILD IN QUALITY AT EVERY STEP OF THE SOFTWARE LIFECYCLE.

by Mike Sowers | *msowers@techwell.com*

Anyone who has attempted to architect, design, develop, test, deliver, and deploy business value with software knows that there are no silver bullets. Humans are fallible, and as hard as we try, we make unintentional mistakes due to the fragility of our methods, tools, techniques, and skill sets. Mistakes are amplified by the stresses of our working environment.

Whether the approach taken is traditional or agile, these fragilities and stresses always exist. This is especially true with projects that rely on continuous integration or a DevOps framework. However, our opportunity to better mitigate these risks significantly improves as we adopt more modern software engineering practices. Key principles such as rapid design and refactoring, delivering small increments of customer value more frequently, engaging customers throughout the process, failing fast and learning, collaboration across development and operations, simplicity, and automation are accelerating our ability to significantly ratchet up our engineering excellence.

Shift from a traditional mental model toward a continuous, iterative series of automated verification steps.

The software development industry now has the opportunity to realize the goal we've always had to build in quality rather than attempt to test quality in after the fact.

Over the years, I have learned a number of techniques and approaches to achieve the goal of designing for quality at every step of the software engineering value stream. This works for both agile and DevOps project environments.

Invest in a Plan

Upfront planning approaches and whole-team engagement and accountability help clarify the value increments to be delivered.

Collective ownership engages the entire team in quality delivery. The customer, product owner, analyst, architect, developer, tester, and operations roles form a cohesive team, laser-focused on delivering customer value.

Design planning helps teams collectively envision and design features and functionality from the beginning of the project.

Prototyping offers the team techniques to assist with the definition of functional and nonfunctional requirements. This includes wireframes, mockups, personas, and in-sprint experiments.

Peer reviews enlist the expertise of others and help ensure that the multiple perspectives are represented throughout a project.

This includes pairing between the developer and tester, business analysts and product owner, and operations team members.

Grooming is vital to keep the team aligned on the most important value to be delivered. This can be accomplished through backlog grooming, prioritization, and story refinement and definition.

Test-first development assists the team in defining the required customer value prior to implementing code.

Techniques such as specification by example help specify the requirements and "test first" approaches help to ensure code is right the first time.

Implement in Small Increments

Implementation approaches quickly verify small increments of change and new functionality.

Guidelines and checklists for architecture, design, coding, testing, and operations, as well as other team norms, promote preventive practices and help avoid mistakes and rework.

Static and dynamic analysis tools help the team understand code structure, complexity, coverage, and security vulnerabilities, and detect anomalies that need to be corrected quickly.

Unit testing and refactoring reduce the risk that code changes cause additional problems, promote early detection of defects at the structural level, and allow the code to be refined with refactoring.

Mocking helps the team have higher confidence in progressive integrations, abstract out dependencies, and verify interactions between dependent classes early. This is accomplished by faking and stubbing at the interface level.

Continuous integration for each code commit and performing integration testing at multiple levels effectively exposes interface defects at the unit and each succeeding level as code is further integrated into the application or system.

Automation of the infrastructure that moves the code and that assists with the testing of the code eliminates errors so that code deployment and delivery yields consistent and repeatable results. Applying the concept of "infrastructure as code" and then defining and automating the value stream (the pipeline) codifies the creation of environments and reduces the variability between development, testing, staging, and preproduction.

Defining "done" clearly specifies the criteria necessary to move an increment of value forward in the pipeline, eliminating disagreements and setting expectations up front for what work completion means.

Think Continuous

After planning and incremental implementation, continuous processing techniques can lead to rapid product delivery.

Fast feedback informs the team immediately when something is not performing or is drifting off course by using in-process dashboards or postprocess production and user experience monitoring.

Iterative risk assessments based on fast feedback enable the team to adjust their verification focus and strategies with immediacy.

Functional and nonfunctional testing increase defect yields earlier in the development lifecycle, as does performing story tests, exploratory tests, and user acceptance tests. Using heuristics helps the team design better tests.

Regression testing for each level validates changes and ensures that a recent change did not impact another area of the application or system.

Combining microservices with container deployment, thus decomposing the application into smaller services, results in improved modularity, makes the application easier to test, reduces resource consumption, and speeds deployment. Together, these practices reduce fragility and aid continuous delivery and deployment.

Continuous delivery and deployment provide quick feedback, employ pipeline automation, and keep the code production line running smoothly and efficiently with a defined set of quality assurance steps and gates that are automated.

Operational tests that validate security, user provisioning, backup, and failover are shifted left and incorporated into earlier testing stages. Operational requirements should be specified and agreed upon by the collective DevOps team early in the project.

Feature and application delivery using feature toggles helps reduce risks when a change is deployed, as the change verification rollout and rollback can be better controlled. Change automation adds an additional level of production control and mitigates defect exposure and risks.

Monitoring of both preproduction and production environments provides the team with a deeper understanding about the quality and usage of the customer value being developed and delivered. In addition to standard environment monitoring, the team also can use application monitoring and user experience monitoring and analysis to understand usage patterns more quickly.

All of This Results in Higher Quality

I've always thought of software development, delivery, and deployment as a series of imperfect translations. Defects are often introduced during the translation process from one handoff to the next.

The above approaches enable the team to — from a traditional mental model toward a continuous, iterative series of automated verification steps.

This investment drives software assurance left to earlier in the lifecycle, which helps us achieve the quality-by-design goal and reduces translation errors. [BSM]

CONNECT. SHARE. LEAD.

Join our global community of change enablers, innovators, and design thinkers.

Individuals, just like you, who are working together to create better business outcomes.

International Institute of Business Analysis™ (IIBA®) is the non-profit professional association serving the growing field of business analysis and related business disciplines. IIBA connects business analysis professionals with global and local networks, professional development opportunities, resources and knowledge. As the voice of the business analysis community, IIBA supports the recognition of the profession, and works to maintain standards for the practice and certification.

IIBA® International Institute of Business Analysis™

IIBA.ORG/MEMBERSHIP

"Once you change the culture, then you can start changing the processes to make things fit in. Once you have the processes, then it's about making sure you have the right tools in place and the right solutions to make it happen."

"To make the digital transformation happen, the first thing that has to happen is you have to have a culture change. Right? It's getting people bought into it. Once people start seeing success, then they start buying in. Then they're like, 'Wow. Why didn't we do this earlier?'"

"To have an automated test case, I have to have the right environments and the right data, so that when I send something down, my test case knows what to evaluate against. So I'm always evaluating against known data."

Use Service Virtualization to Increase Test Coverage and Improve Quality

Kenneth Merkel

Years in Industry:	**22**
Company:	**CA Technologies**
Interviewed by:	**Jennifer Bonine**
Email:	**jbonine@tapqa.com**

"Look at what you're doing today and then look at the things that are blocking you from where you want to go. We call it current state versus future state. Once I know why I can't get there [to future state] and what the value is if I remove the obstacle, then we can talk about the solutions to make that happen."

"If I can make development more efficient and ship higher quality code to QA, QA becomes more efficient and they can get to 100 percent test coverage sooner, hitting our quota in production. The whole thing dominos and slices down."

"We do a lot of value stream analysis with our customers to talk about "big rocks" (i.e., release delays). What does it really mean from a cost perspective? If you change it, does the company care? Does it move the needle?"

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

CLICK HERE FOR THE FULL INTERVIEW 》

LIVE VIRTUAL TRAINING

LEARN ANYWHERE! LIVE, INSTRUCTOR-LED PROFESSIONAL TRAINING COURSES

Live Virtual Courses:

» Agile Tester Certification
» Software Tester Certification—Foundation Level
» Fundamentals of Agile Certification—ICAgile
» 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

Great Big Agile

An OS for Agile Leaders

By Jeff Dalton

As the prolific and popular SPaMCAST podcaster Tom Cagley proclaimed during his keynote at the 2017 Agile Leadership Camp, "Values aren't really what matters; behavior matters." Cagley, who has interviewed more than five hundred technology leaders on his podcast series, hit the nail squarely on the head. Culture is usually derived from organizational values.

The behavior of team members, business stakeholders, partners, and leadership is all that matters, as it demonstrates real, as opposed to stated, culture. Too many companies say the words while demonstrating antipatterns that proliferate throughout the organization. Instead, companies should project and promote behaviors that build scalable and sustainable self-organization at all levels.

While many leaders are asking about scaling agility these days, they're asking the wrong question. They should be asking how to scale self-organization using a healthy dose of agile values, frameworks, and techniques.

From time to time, people have made attempts to transition from the "rules of men" to the "rules of nature," a system that more closely mimics the natural world. This is where the scales are inverted, roles and accountabilities are dispersed throughout the organization, and people go about the sometimes messy process of organizing themselves without having to ask permission from any business leader.

In more than one hundred organizations we've assessed, tech leaders tell us that they want to push decision-making down and give their teams greater autonomy, but their behaviors are in conflict with agile values, creating an organizational type mismatch. In other words, the API is broken, and the architecture needs to change.

Introducing Great Big Agile

The concept of "great big agile" requires leadership at all levels, just not the kind we are used to. Simply working with an agile coach to implement well-known ceremonies is not enough. Metaphorically, the operating system needs an upgrade.

In today's corporate hierarchies where command-and-control structures, low trust, long-term planning, and risk management reign supreme, the skills required to thrive and survive are anything but agile. This leaves agile teams to push the culture uphill, leading to unpredictable results once business operations expand beyond the boundaries of the core agile team. This creates chaos because information technology, operations, marketing, infrastructure, business development, sales, and end-users are not on the same page.

Agile without self-organization isn't agile at all. There is nothing wrong with adopting ceremonies and techniques that are most commonly identified as being agile, and many companies have found some success with that, but the power of agile values and their associated frameworks grows exponentially once self-organization is perfected.

How Agile Is Your Organization?

I have witnessed only a few examples of large organizations that have been successful with true agility. Far more insist that they are agile by merely adopting a couple of techniques or ceremonies within an otherwise command-and-control, low-trust, and traditional operating model.

Even with impediments to self-organization and agility, companies and government agencies are increasingly turning to agile frameworks because they sense, correctly, that by improving their methods and tools, they may increase customer satisfaction, speed delivery of value, and raise the quality of software, systems and services. The problem is, they often think that it's *only* about changing their methods and tools, and they give short shrift to the power of culture.

Once the domain of mid-size software companies, "agile-like," a term that describes an organization that adopts some agile ceremonies without the accompanying organizational change, has become mainstream in the IT shops of Fortune 100 companies and government agencies.

The concept of "great big agile" requires leadership at all levels, just not the kind we are used to.

Why Agile Matters

Without exception, all of the organizations I work with have expressed an interest in "going agile," if they have not already done so. This is a strategic decision that has deep-rooted cultural implications and should not be taken lightly. Many leaders do not realize the extent to which they have to change the way *they* behave.

There are several reasons why an organization should transition to a model that is agile and self-organizing:

Agile frameworks reduce the cost of failure. It is conventional wisdom in the technology industry that failure is inevitable, with many companies seeing failure rates as high as 70 percent. [1] Research conducted by organizations such as the Project Management Institute and the Software Engineering Institute has consistently confirmed high failure rates, so it makes sense to seek solutions that assume failure, not success, and to simply reduce its cost.

Failure is not just an option; it should be expected. A foundational premise of agile is to acknowledge that failure is normal, and we should plan to fail fast and learn as much as we can.

This reduces a project's cost while allowing teams to redirect efforts toward a more successful approach through the use of experimentation, retrospectives, and short, timeboxed iterations. Quality professionals will recognize this as an application of W. Edwards Deming's "plan-do-check-act" framework of continuous improvement applied in short iterations. [2]

Agile methods deliver business value to end-users more quickly. Value is delivered more quickly with an iterative and incremental delivery approach due to low-value features being de-prioritized or discarded, freeing up valuable resources to focus on the high-priority needs of the customer.

Self-organization pushes decision-making downward, freeing leaders to focus on strategy. For decades, the technology industry has explored ways to push decisions downward. Agile frameworks finally provide a model that can make that a reality, if only leaders are willing to accept their role as enablers rather than task managers. A successful agile team requires minimal oversight, makes day-to-day operational decisions, collaborates with business customers, and delivers business value without the need for continuous management intervention.

Agile complements important IT industry models. If CMMI, ISO 9001, and the *PMBOK® Guide* are models we *use*, agile is something we *are*. For example, CMMI has a perspective of defining what needs to occur for a product or service to be successfully delivered, while agile values describe why we take those actions. If adopted in this way, CMMI makes agile stronger. [3]

Many leaders do not realize the extent to which they have to change the way they behave.

All Is Not Well with Agile

While the popularity of agile frameworks like Scrum, Extreme Programming, and Scaled Agile Framework cannot be understated, in some ways, they have been a victim of their own success.

Large companies eager to replicate small company successes; satisfy younger, more self-organizing employees; and to just simply "go agile" have jumped on the agile bandwagon. Unfortunately, they often give inadequate attention to the changes in governance, infrastructure, measurement, and training required to succeed. The results have been chaotic, with large organizations adopting some elements of Scrum (e.g., daily standups and sprints) and force-fitting them with more traditional roles and techniques that are in conflict with agile values. This conflict dilutes the value of the very agile ceremonies they use and leaves the organization without the benefits they were hoping to achieve. Out of more than two hundred companies assessed by AgileCxO and its partners:

- More than 90 percent assigned project managers for task management, oversight, and control of agile teams
- More than half did not conduct regular retrospectives
- Almost half conflated story points with hours yet still considered velocity to be a reliable metric
- Most made no changes to governance, infrastructure, or training to support agile adoption

These obvious conflicts with agile values result in a scenario where leaders may desire agility but continue to apply low-trust defined process control models to run the business, when a high-trust, empirical process control model is required for successful agility. This friction, often manifesting itself as "Scrummerfall" or "ScrumBut" ("we're agile, but…"), corrupts and degrades the very performance that agile leaders are seeking to achieve.

Jim Bouchard, author of *The Sensei Leader*, sums it up: "Don't even attempt to transform your organization until you can transform yourself." [4]

The Missing Layer in the Operating System

While the Agile Manifesto excels in describing *why* we do what we do, and industry frameworks and models describe *what* we need to accomplish, there is little guidance for leaders or teams on *how* to experience consistent success with self-organization and agility.

This layer isn't a process, but a set of guide rails that help leaders and team members recognize what large-scale agile looks like and provide the ability to recognize, evaluate, and improve agile performance. As I often tell conference audiences during my talks, "It's not magic. You just need to be able to recognize it."

To succeed with "great big agile," technology leaders and teams can start by categorizing capability into three interdependent layers: *why*, *what*, and *how*.

"Why" models: The set of values and guiding principles that are traced directly to the goals and methods of the organization. With its guiding principles, the Agile Manifesto is perhaps the best example.

"What" models: The set of frameworks, methods, roles, and artifacts derived from industry-standard models or internal methodologies. These models define what needs to be done and often provide examples that help us understand what we need to do while executing the software product development process.

"How" models: A set of behaviors, actions, and outcomes that helps define and evaluate organizational success and supports the culture, goals, and objectives of the organization. "How" models trace directly to established values, guiding principles, and frameworks to ensure that the behaviors exhibited by teams reflect the values of the organization.

An Operating System for Scalable Agility

AgileCxO's Agile Performance Holarchy (APH) is an organizational operating system that encapsulates all three layers, providing leaders with an integrated view of organizational agile performance.

APH provides agile leaders and teams with a model to build, evaluate, and sustain great agile behaviors and habits. It is not an agile maturity model or a process, but an operating system for sustainable agility. Figure 1 shows how APH defines performance circles and holons.

Introduced in the 1967 book *The Ghost in the Machine* by Arthur Koestler, holons are described as self-reliant entities that "possess a degree of independence and can handle contingencies without asking higher authorities for instructions." [5]

Koestler defines a holarchy as a "hierarchy of self-regulating holons that function first as autonomous wholes in supraordination to their parts, secondly as dependent parts in a sub-ordination to controls on higher levels, and thirdly in coordination with their local environment."

A holarchy works well for describing and evaluating agile performance, where behaviors are self-organizing and empirical and the sequence of actions and outcomes is unpredictable, iterative, and recursive, rather than procedural.

The APH is composed of interdependent actions and outcomes that provide guidance for the behaviors, ceremonies, and techniques that might be performed to meet the outcomes by team members, functional groups, and leaders throughout the organization. Sequence, rigor, and intensity are determined by functional and project teams, not by management. There are several key APH components.

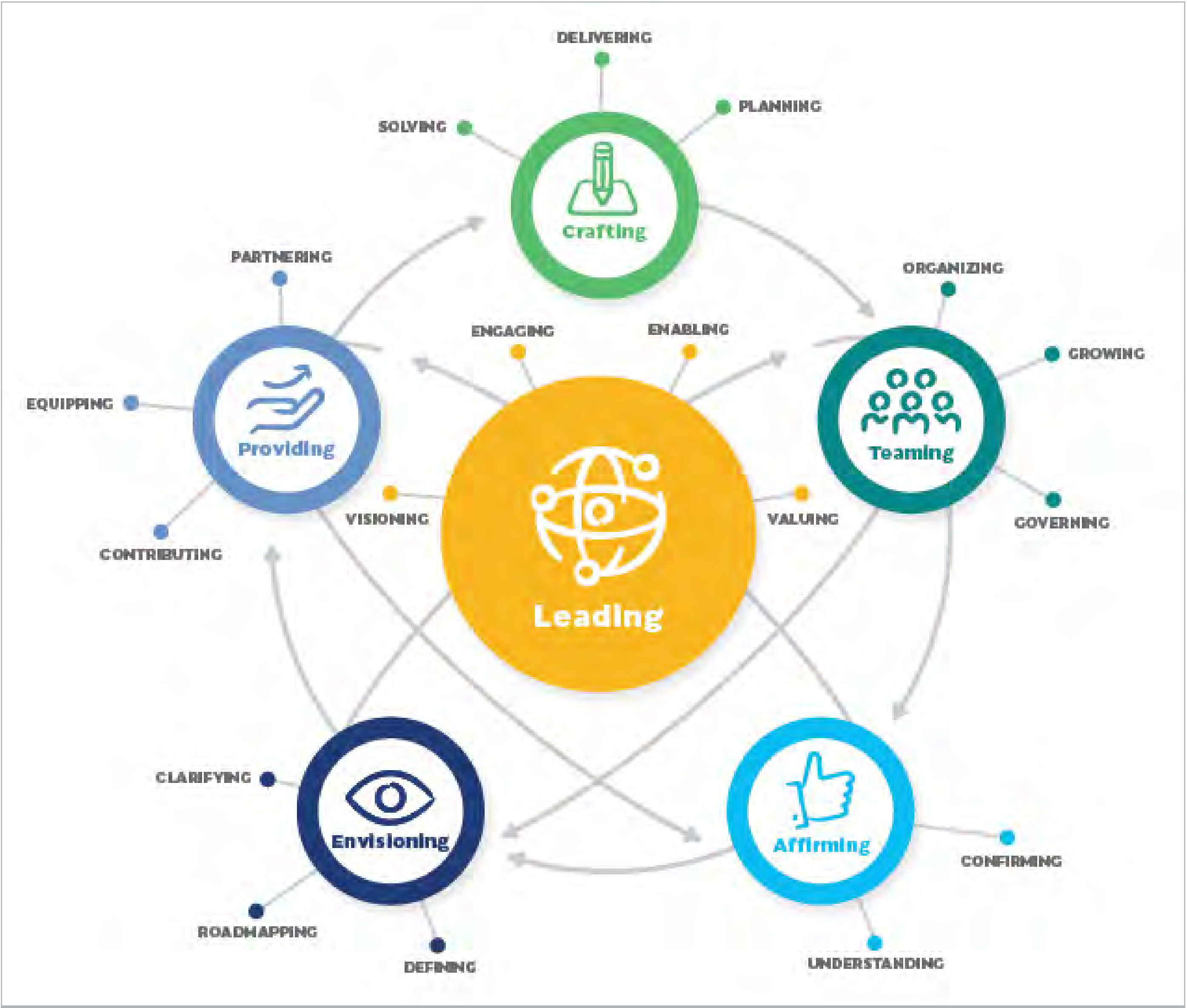


Figure 1: The agile performance holarchy

PERFORMANCE CIRCLES

Performance circles encapsulate a discrete set of behaviors with a set of actions and outcomes that are essential to successfully step through the process of adopting, transforming, and mastering large-scale agility.

Organizations wishing to benchmark performance against the APH may evaluate performance circles to determine how they are adopting, transforming, or mastering the behaviors of that circle.

There are six performance circles, each with a specific objective for leadership, depicted in the classic user story format of role, mission, and business value. These are described in table 1 in terms of goals and benefit.

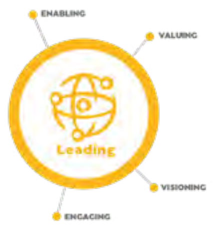
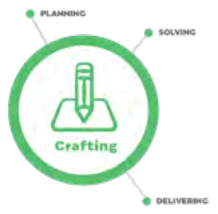
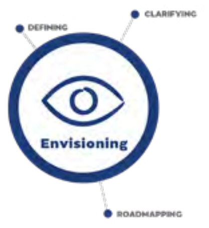
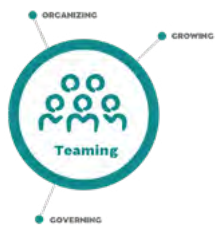
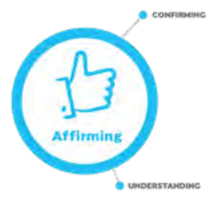
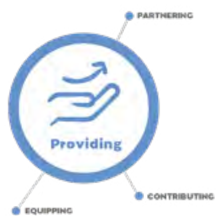
Performance Circle	Objective User Story
	**As an** agile leader, **I want** to project agile values, provide the environment, and establish a vision **so that** my teams can be agile and successful in everything they do.
	**As an** agile leader, **I want** agile team members engaged in the planning and building of high quality products **so that** we deliver the solution as expected.
	**As a** product owner, **I want** to establish a roadmap, release plan, and backlog **so that** the overall vision of the product or service can be realized.
	**As an** agile leader, **I want** teams and functional areas to learn and master self-organization and agile ceremonies and techniques **so that** the entire organization can benefit fully from agile adoption.
	**As an** agile leader, **I want** to confirm that teams are demonstrating agile values, methods, and techniques as expected **so that** I can understand what is working well and what needs improvement.
	**As an** agile leader, **I want** to foster a continuous improvement environment and engage with agile partners **so that** agile teams can grow their capabilities.

Table 1: Performance circles and objectives

HOLONS

Several holons are encapsulated within each performance circle, and they represent a set of actions and outcomes that can effectively stand alone but are also an integral part of a greater whole. All the actions and outcomes should be implemented in order to realize the value of each holon.

There are eighteen independent holons within the APH, as shown in table 2.

As an example, the engaging holon encourages the use of *gemba walks*, a process where leaders walk around to observe their teams and "understand by seeing" in order to improve engagement and enable quality.

ACTIONS

An action is the specific behavior that is applied to meet a holon's objective. All behaviors in the holon should be demonstrated in order to meet the intent of the objective, and each must always be aligned with agile values. The APH recommends agile ceremonies and techniques that, when executed successfully, will meet the intent of the actions. Not being a process, the APH does not require any ceremony or technique, although it does provide a list of potential options for those who wish to benefit from their guidance.

CEREMONIES AND TECHNIQUES

Each action provides a recommended set of ceremonies and techniques derived from agile and lean frameworks that can be adopted to demonstrate the desired behavior and meet the intent of the action.

The sixty-eight ceremonies and techniques described in the APH include all the typical visual information indicators, roles, expected behaviors, and actions, allowing leaders and all levels to effectively play their roles as servant leaders and recognize, evaluate, and enable improved organizational performance where needed.

Large-scale agility requires large-scale self-organization, but it isn't magic. Contrary to the claims of many agile-like practitioners that "agile doesn't use process," agile uses a lot of process; however, it may not be the kind you're used to. The freedom to successfully self-organize results in freedom through mastery.

Leading	Crafting	Envisioning	Teaming	Affirming	Providing
Valuing	Planning	Road-mapping	Organizing	Confirming	Contributing
Enabling	Solving	Defining	Growing	Understanding	Partnering
Envisioning	Delivering	Clarifying	Governing		Equipping
Engaging					

Table 2: Holons within each performance circle define expected behaviors

OBJECTIVES AND OUTCOMES

Each holon describes objectives in a user story format that should be met in order to instantiate the value of the holon. An objective can be met by taking the defined actions in a manner and behavior that is consistent with agile values.

Holons contain a set of outcomes, like the performance circle they are surrounded by, that can be used to evaluate, improve, and sustain organizational agile performance within that context. The outcomes at the holon level are categorized into three levels, adopting, transforming, and mastering, and are used to help leaders evaluate and improve their organization's agile maturity.

For example, the objective and outcomes of the delivering holon, part of the crafting performance circle, are depicted in table 3.

Adopting Level Outcomes	Transforming Level Outcomes	Mastering Level Outcomes
* Current state of organizational performance is defined.	* SWOT is complete and published.	* Organizational performance sprints are executed.
* Future state is identified and displayed.	* Backlog for future state exists in visual format.	* Progress is visually displayed using VIM.
	* Culture transformation release plan exists.	* Impediments to organizational performance are regularly identified and removed.

Table 3: Leaders can assess each holon's objectives and determine which outcomes have been met

Self-organizing teams must also learn their craft, practice their forms, and progress through the stages of adopting, transforming, and mastering agility in order to be self-reliant.

Put In the Work and Reap the Rewards

According to VersionOne's "11th Annual State of Agile Report," 98 percent of respondents who believed they were agile reported success with agile projects. [6] That's a stunning statistic, but it isn't a coincidence. When agile works, the results are spectacular.

However, the same survey said that many organizations report a conflict between corporate policies and agile vales, that leadership lacked the skills to enable agile teams, and that more than half were still maturing years after adoption.

Successful agile organizations are not successful because they adopt popular ceremonies or frameworks. They are successful because they are committed to open, collaborative, and transparent servant leadership at all levels, and they have cultures where failure and risk are not punished, but celebrated as a way to learn and improve. Strong agile organizations are learning organizations that can demonstrate a mastery of self-organization.

For those companies, the behaviors they exhibit are the natural outcome of organizational culture change, and they produce better results when they align with the rules of nature, which consist of, among other things, iteration, continuous learning, incremental wins (and failures), transparency, and team collaboration. They succeed because they rely on trust, collaboration, and deep respect for team members, and they recognize that while teaming is highly valued, personal commitment to behavioral excellence is the prime directive. In other words, an agile culture aligns with the rules of nature, and successful agile teams are those that have mastered both self-organization and personal self-reliance.

Conversely, if an organization is autocratic, with a high degree of secrecy, distrust, and negativity; blames people for failures; and is generally low in trust, then they will struggle with agile adoption. Without an "operating system" upgrade, they can never realize the benefits of organizational agility. For those companies, adopting Scrum or Extreme Programming, both excellent frameworks, can be misleading. They feel agile, but, in practice, they are anything but. This will come back to bite them in the form of missed deadlines, high turnover, unhappy customers, low quality, and high cost. This eventually leads management to declare, "We tried agile, and it didn't work for us."

To summarize, the agile performance holarchy provides leaders and teams with objectives, outcomes, and behavioral guide rails to succeed. Along with an assessment method, training, and certifications you can chart a course to a high-performing agile future. **[BSM]** jdalton@agilecxo.org

CLICK FOR THIS STORY'S **REFERENCES**

TRY IT FREE

QMETRY
TEST MANAGEMENT

Deliver High-Quality Software
Faster and Intelligently

sales@qmetry.com www.qmetry.com +1 408 727 1101

ALL CUSTOMERS ARE NOT CREATED EQUAL

By
REGINA EVANS

If you have ever worked with a customer, it shouldn't be news to you that all customers are not created equal. It is even more difficult to know customer needs on software development projects, because end-users have different requirements, or sometimes no idea what they want at all.

I have had the privilege of working as a customer relationship manager in various industries, and it is vital to learn how to successfully develop software that meets end-user needs.

The Necessary Skills to Work with Customers

Whether your customer sits on the other side of the computer or in some other city or country, there are four basic skills that anyone working with customers must understand and master:

- Understand the customer's situation
- Put yourself in the customer's shoes
- Remember that we are all human
- Do not take negative communication personally

One of the first things that we must do is be willing to hear another person's point of view. In fact, *hearing* is probably not strong enough while truly *listening* is more important. Often, we have so many conversations going on in our own head that we are not paying attention to the conversation that is taking place with customers. When dealing with customers, you must give them the full attention that you would want someone to give you. In doing so, we can minimize frustration and any misunderstanding.

Understanding a Customer's Needs When Gathering Requirements

Before development, it is important to understand what the end-user needs and how they will use the software. One way to ensure clarity about customer needs is to confirm accuracy of the requirements and prioritize them so that most, if not all, of the critical functionality is built into the finished product. Take time to create an outline of what the finished product should look like and perform some level of peer review or user testing to avoid developing something that easily breaks or is difficult to use.

When developing software, it is important to understand that the end-user wants to be able to access an application easily, securely, and with a full understanding of user requirements. When an application is difficult to install, it frustrates the end-user and can result in their not wanting to use the app.

Any fear of internet safety or lack of privacy protection will also keep them from wanting to access your application. Application speed is important to end-users, so make sure that the application does not have built-in flaws that cause long lag times or create a situation where it is difficult for the user to stop the application.

To avoid these possible situations, it is helpful to make your application easily accessible, secure, and transparent regarding, any licensing requirements. Gaining an end-user's trust is critical. It is always best to create a long-lasting customer relationship that can lead to positive referrals based on great end-user experiences.

You're a Customer, Too!

Developers need customers to understand what is needed for the creation of a good software application. Sometimes, it is difficult for customers to understand the software development process, so it is important to help them understand the development side of things. One way to do that is to have an open dialogue about what is being developed and educate the customer up front about what to expect, from initiation through product delivery.

When end-users have some idea about what it takes to develop an application, they are less likely to get frustrated and impatient when they have questions or when timelines require adjustment.

Developers must be able to put themselves in the shoes of the customer.

I have witnessed several situations where customers were completely mishandled in customer support organizations. One event that stands out is a customer calling in claiming that their phone would not forward calls. They were frustrated and yelling because they viewed it as a defect. After walking the customer through the steps to forward a call, the anger dissipated.

DEVELOPERS MUST BE ABLE TO PUT THEMSELVES IN THE SHOES OF THE CUSTOMER.

Try to handle these types of situations with empathy, because when something is important to a customer, it must also be important to us. As people, we should care about each other, and if it is urgent to the end-users, we should not disregard their feelings, but try to understand the situation and see what we can do to help. Support staff must be prepared to be assertive when representing customer needs, and software developers must be open to performing fixes and improvements in a timely manner.

How a product or service provider attends to their customers becomes a true competitive advantage. As technology advances at a fast rate, customers have plenty of choices based on intense competition. To ensure customer loyalty, be empathetic to the end-user and provide a level of customer service that treats everyone as if they are important. A good example of attention to the customer is how quickly a software development organization responds to software issues.

Even if you are not able to fix the issue right away, it is good customer service to state a realistic timeframe for when you will have an issue addressed and to keep the customer updated on progress. And, if a prioritized fix will be delayed, inform the customer. Always treat others as you'd like to be treated.

We're All in This Together

A developer who exhibits integrity can go a long way in building your organization's relationship with your customer. It is important to remember that we are all human. All people—especially your customers—expect a level of respect. If someone is experiencing negative emotions, it does not hurt to respond with empathy and courtesy. The use of words like "please," "thanks," and "you're welcome" should be a common part of our vocabulary. People appreciate the use of positive words, and it usually causes people to reciprocate.

Let's switch hats for a moment. Some of my own customer service experiences have been in situations where I was upset and angry and the customer service representative helping me was able to sincerely express concern and courtesy.

In these types of scenarios, it is best to calm the person down, get to the root of the problem, and together come up with a solution. This is not something that just happens—it has to be practiced. Don't get me wrong, the solution may not always be exactly what the customer wants, but everyone benefits if the situation can be diffused and both parties can work through the problem together.

In the world of software development, where technology can be pretty dry, the end-user will appreciate a personal touch.

For software products where quality means everything, developers should invest time and effort to take an active role in quality validation as well as end-user acceptance testing. When code updates are needed, be sure to test your changes before implementation, communicate the changes to the end-user, and welcome feedback so that you can make continuous improvements. If someone suggests a noteworthy idea, take the time to say thank you.

Develop a Thick Skin

Last, but not least, you cannot take anything negative that the customer says personally. Most of us have experienced these kinds of situations:

- You say hello, and you cannot get another word in. The person wants to speak to a manager before you even have a chance to find out the details of the problem, and you are left not understanding what just happened.
- A customer types a negative comment regarding your work, and before you think it through, you have responded, and a shouting match ensues. The conversation is out there for the world to see.

In our lives, we will encounter all kinds of customers: business partners, buyers, parents, kids, spouses, colleagues, managers, vendors, purchasers, strangers, and so on. Regardless of what product or service you are selling or who your customer is, we must all be willing to develop the skills and desire to successfully help others. Be intentional about showing respect toward customers and you will become successful at customer relationship management.

Negative responses never solve customer problems. This goes for those in customer support roles as well as software development roles. Both career paths thrive on your ability to solve problems, so look beyond the surface of the reactions to determine what problems you can solve.

Negative feedback can be difficult to take, but don't take it personally. Instead, take it as an opportunity to improve your work. When developing, think about what an end-user may see as negative when using the application. For example, if you build in an animation or advertisement, you want to make sure it provides the needed content but does not run too long and frustrate the user. Forward thinking provides a way to address these types of challenges and minimize negative feedback.

REGARDLESS OF WHAT PRODUCT OR SERVICE YOU ARE SELLING OR WHO YOUR CUSTOMER IS, WE MUST ALL BE WILLING TO DEVELOP THE SKILLS AND DESIRE TO SUCCESSFULLY HELP OTHERS.

Developing the skill to consider these types of reactions up front will assist you in creating great work. Having peer reviews are a great way to get feedback that will help enhance your work, too. In addition, you should consider having someone who knows nothing about software development be a part of your user testing. These individuals will give you insights that you may not have considered.

There's More to Customer Service Than Lip Service

Successfully supporting customers is a tough job. All customers are not created equal, but they all expect to be treated with respect. If you foster an atmosphere of open communication, ensure security they can trust, and provide a quality product or service that meets their needs, you can earn their respect and loyalty in return.

As you build customer loyalty, you will also build a reputation that stretches far beyond one customer. [BSM]

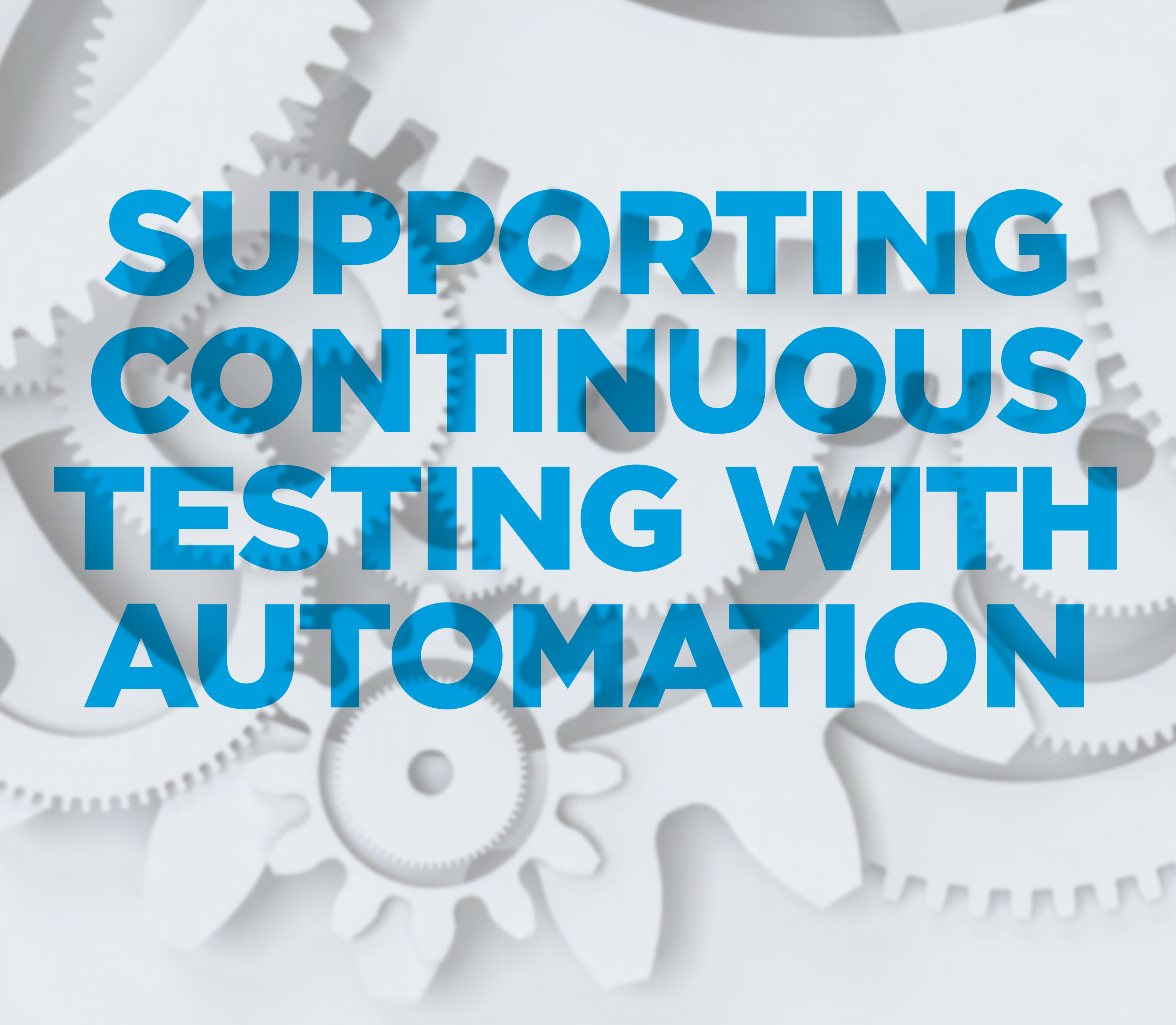
Regina.Evans@netapp.com

STARCANADA 2018

FULL PROGRAM AVAILABLE

Plus you can enjoy a multi-day training class to advance your career, experience the Test Lab, network with other software testing professionals, meet with the speakers, enjoy a bonus day with Women Who Test, and more.

https://well.tc/sc18



SUPPORTING CONTINUOUS TESTING WITH AUTOMATION

BY **BAS DIJKSTRA**

We're living in a world that requires software development organizations to continuously deliver value to their customers. As a result, software development teams need rapid feedback on software quality and its business value. In order to keep up with this demand, many organizations have abandoned their traditional testing efforts in favor of a more flexible, risk-reducing approach, including adopting test automation.

However, teams currently implementing test automation fail to support the constant assessment of quality and business value—a process known as continuous testing.

Let's take a closer look at why automation efforts fall short and what can be done to improve the situation and make test automation the key ingredient for continuous testing.

A Primer on Continuous Testing

Continuous testing can be defined as subjecting every build of an application to a specific set of tests that assess and report on the quality and the business risks for that build, in every environment that build passes through, often including production (figure 1).

Putting the term continuous aside for the moment, these tests should contribute to rapid feedback about the value that the product provides to someone who matters. This is consistent to the definition of quality coined by Jerry Weinberg and adapted by James Bach and Michael Bolton. [1]

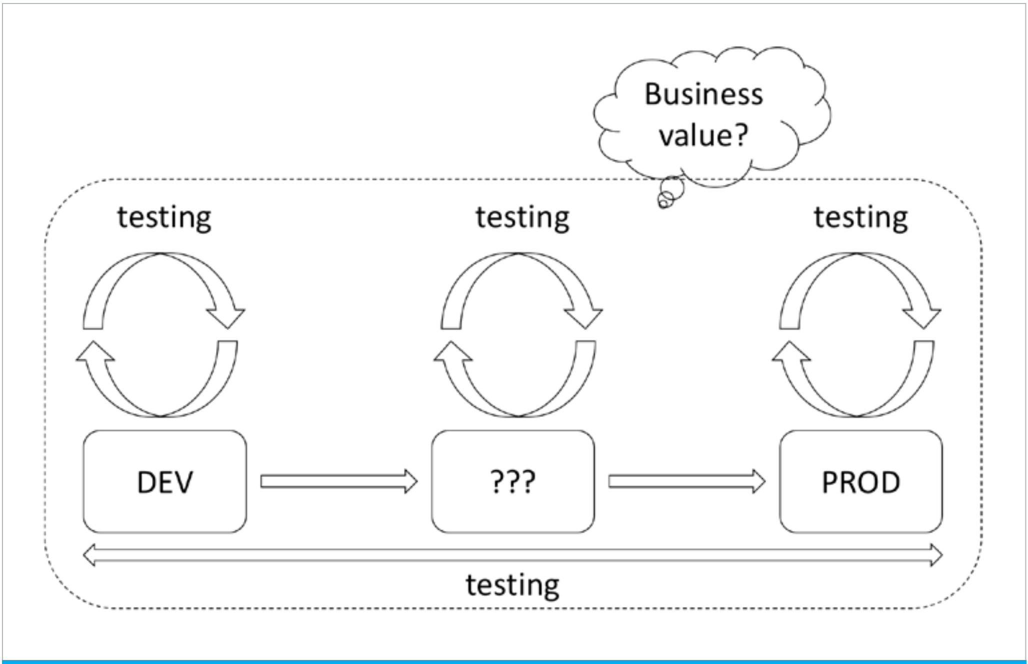


Figure 1: Depiction of continuous testing

Introducing test automation as part of the overall testing approach seems like a straightforward choice, and, indeed, automation can potentially provide considerable value to teams implementing continuous testing. Too often, though, it falls short of promises made and expectations set. Test automation takes too long to run, is hard to maintain, and fails to provide the required insight into application quality. It can mislead its users and stakeholders if the technology results in time and money wasted, applications being released into production without confidence, or bugs going undetected. This can cause automation efforts to be shelved.

In my opinion, there are three reasons test automation fails to deliver in support of continuous testing. Let's look at each of them in detail.

Reason 1: Tests Are Unfit for Continuous Testing

If your automated tests are meant to support continuous testing, they should be able to run continuously or on demand. Whether you're running them once a day or once a minute, your tests should be ready to go and provide the required feedback whenever you need it. Unfortunately, this is not always the case.

To help you make sure that your automated tests are better fit for continuous testing, I use the acronym FITR as a reminder of the qualities to look for when assessing automated tests: Your tests should be focused, informative, trustworthy, and repeatable.

Focused: Having focused tests means that your tests should be tied as closely as possible to the piece of business logic or functionality being tested.

One prime example where tests fail to meet this requirement is when teams write user interface tests. Using a tool like Selenium WebDriver can verify the workings of business logic that's exposed to the user interface through an API, but this can lead to unnecessarily long feedback loops, as UI-driven tests tend to be comparatively slow in execution and can give "shallow" feedback. The further away a test is from the logic it is verifying, the harder it is to pinpoint what, exactly, is wrong when a failure is encountered. The same principle applies to using checks to verify logic that operates at the API level that can be more effectively tested with unit tests.

In short, always make sure that your tests operate at the right level and with the right scope. You should dive deeper into your application under test to find the most efficient way to test anything and everything.

Informative: In order to quickly gauge the effect that a change has on the business value your application provides, your tests should communicate its intent and its result in a clear and unambiguous manner. This minimizes the time needed to investigate failures or to update tests that reflect changes in the application under test.

To improve the way your tests communicate their intent, practice good programming principles like DRY ("don't repeat yourself") and YAGNI ("you ain't gonna need it"), as well as useful naming conventions to make your code readable. Keep your code as close to self-documenting as possible. If it makes sense, consider adding a behavior-driven development library, such as Cucumber or SpecFlow, to document the expected behavior that your tests are verifying.

To improve the way your tests communicate their result, make sure that the target audience is correctly identified for the reports and logging created by your tests. You must also ensure that information requirements are fulfilled and that the audience can make the appropriate decisions based on these reports. Remember that your target audience can be a person (yourself, another tester, or a product owner) and a machine (such as a build server or deployment pipeline).

Trustworthy: Especially for teams that are practicing continuous delivery or even continuous deployment, automated tests are often the only means used to gauge software quality before it is put into production. This means software development teams need to be able to rely on the outcome of these tests in order to be confident that their product retains a certain degree of quality at all times.

Therefore, a word of caution is in place. Automated tests can trick you with false positives when tests fail but there's no actual defect in the application. Automated tests can also incorrectly report false negatives when tests pass and allow a defect that should have been caught before deployment into production.

To help prevent deceptive automation, review your tests periodically. Fix tests that cause problems, and don't hesitate to delete tests. Experiment with techniques such as mutation testing to help assess the quality of unit tests.

Repeatable: Continuous testing requires that you're able to run your tests on demand. However, there are a number of factors that can prevent you from running your tests over and over again with the click of a button. The two main culprits are test data and dependencies in test environments.

When you're running integration or end-to-end tests, it's highly likely that your test triggers an interaction between different components of your system or between your system and third-party dependencies. This often requires test data to be synchronized between these components. For example, if you're trying to create an insurance policy for a car and part of the application process is a license plate validity check, you'll need to make sure that a license plate number exists in the service each time it is entered. When you're running tests frequently, tests like this can require a lot of test data, and your test data strategy should be able to support that.

A lack of synchronized test data is just one of the ways in which test environments can be a bottleneck when it comes to implementing continuous testing. Critical dependencies required for integration and end-to-end testing can be difficult to access for other reasons, too. Real-world dependencies must be considered, including requiring access fees for third-party services and shared access to mainframes or web services.

One possible approach to dealing with these test environment issues is the implementation of stubbing, mocking, or service virtualization. These are techniques that can be used to simulate the behavior of these critical yet hard-to-access dependencies, thereby removing bottlenecks that are in the way of creating repeatable, on-demand test automation.

Reason 2: Too Many or Too Few Tests

Continuous testing is about continuously gaining insight into the quality and business value of an application using the shortest possible feedback loop. This means there is a trade-off between adding more tests, test cycles, and test types. Increasing test coverage can conflict with the need to get the information required to make decisions (to deploy or not to deploy) as fast as possible.

For every test they are about to include in their continuous testing efforts, software development teams should ask themselves, "Do we really need the information that this test provides every time we build our software?" From a software testing perspective, it is easy to "add another test, just to be sure," but teams should remember that every test added has a negative impact on the length of the feedback loop. From a business value perspective, it might be tempting to cut testing efforts relentlessly in order to minimize the time to production, but this comes with a cost of its own in terms of lower test coverage. It's imperative that for every test that's not added to the continuous testing cycle, teams ask themselves, "Are we OK with not having the information provided by this test for every build?" Performing certain types of tests periodically may be a useful strategy to reach a good trade-off between risk, quality, and speed, but perhaps not on every build.

Too much testing results in unnecessarily long feedback loops and a loss in time to market. Too little testing results in poor insight into application quality before it is put into production, as well as a higher risk of defects finding their way to the customer. As with so many things in life, there is no one right answer or approach that works in all cases. Using the above considerations, find the trade-off that works best for your situation, learn from the outcomes, and continuously monitor and improve your testing efforts.

Reason 3: Not Everything Can Be Automated

Contrary to popular belief, not all testing activities can be automated. I'd like more people to view automation as something that supports testing, rather than something that replaces the things testers do when they test. [2]

Even though automation is likely to be an essential part of most continuous testing efforts for its ability to give rapid feedback, it should be considered a bad idea to rely solely on automation. It cannot paint a complete picture about product quality. On the other hand, there are likely more opportunities for automation to support your continuous testing efforts than you might think. Consider using continuous testing in test data generation, monitoring, log analysis, continuous performance testing, or service virtualization—all forms of automation that use tools for testing. Adopting more innovative test automation coverage can help you achieve your continuous testing goals.

Automation can be an invaluable asset toward a successful adoption of continuous testing, but we should definitely see it in a more realistic light. By adopting the advice in this article, you should be able to get the most out of your automation and your continuous testing efforts. [BSM] bas@ontestautomation.com

CLICK FOR THIS STORY'S REFERENCES

TRAIN YOUR TEAM ON YOUR TURF

ON-SITE ADVANTAGE

BRING THE TRAINING TO YOU

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

80+ ON-SITE COURSES

25+ AGILE & DEVOPS COURSES

40+ TESTING COURSES

8 REQUIREMENTS & BUSINESS ANALYSIS COURSES

8 DEV & TESTING TOOLS COURSES

8 PROJECT MANAGEMENT COURSES

7 TEST AUTOMATION COURSES

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

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

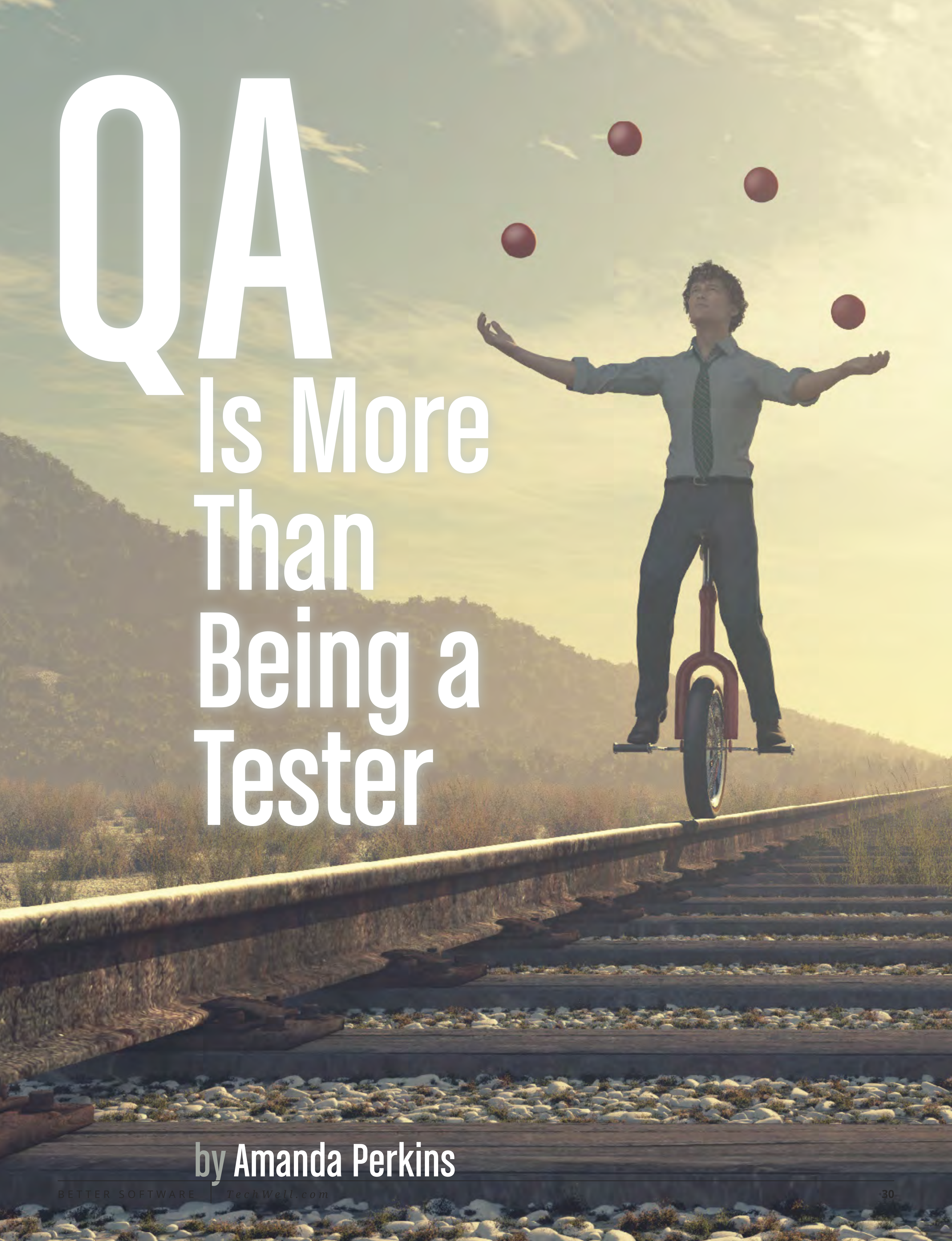
IF YOU HAVE 6 OR MORE TO TRAIN, CONSIDER ON-SITE TRAINING

SQETRAINING.COM/ON-SITE

SQE TRAINING
A TECHWELL COMPANY

QA Is More Than Being a Tester

by Amanda Perkins



Some say quality assurance (QA) work in an information technology environment is easy. All you have to do is make sure code works. For those not in the computer field, the perception of what QA is centers around pushing buttons. But that's not even close to being true in the life of a QA professional. QA is so much more than just being a tester.

Examining the QA Role

I believe there are three fundamentals to how we verify and how we test.

1. We test limits: A key part of QA involves stressing limits, and that includes load testing in a system, testing endurance when new testing methods are introduced, and testing ourselves when we become key contributors in a development or project team.

2. We test patience: QA work provides an objective yardstick for the quality of code produced by developers by breaking their hard work and then telling them all about it. We test our own patience when test environments do not work or when the code released to us is pronounced "done" but it isn't even close.

3. We test software: Finally, this brings us to testing software. We test what the requirements tell us needs to work a certain way, and we test what is not in the requirements. We test the system in ways the end-user understands, and we test things we thought about while testing something else. Software testing is only the start.

QA Takes On the Role of Business Analyst

QA individuals often act as business analysts. Sometimes user stories are laid out neatly for testing, and while we are reviewing the story, we realize there is something else to test.

We ask questions. We use our knowledge to dispute some of the requirements set in front of us. We think like the user, the business, and a tester, then we make suggestions based on what we find. We seek answers from any source and debate the way the task or ticket was written. QA practitioners think outside the box and imagine that some future user is going to push that silly button and find something no one expected. Based on my experience, the best folks in QA are always curious and question everything.

Roles often change hands, and that is part of belonging to a team. For example, sometimes we must write the tickets, create user stories, write reproduction steps, and make ourselves available when our teammates have questions. We answer questions but consult with our project manager or business analyst to verify what they want the system to do. We work the system and write out tests on what requirements we understand.

This isn't all we do.

QA Must Be an Effective Communicator

Besides being curious, a truly great QA professional is an effective communicator. We talk to developers to find out what they did (or didn't do). And when developers are anxious to start coding and avoid further detailed discussions, QA must talk to the business analyst (BA) to find out what requirements actually mean.

We occasionally get the chance to talk to end-users. And not all discussions are about product quality. There are times when QA gets the opportunity to ask an end-user about how they use the system and what works best for them. Sometimes QA gets to introduce the end-user to an improvement to their current workflow and discuss how best to implement the changes. At other times, we get to talk to them and learn what the business analyst and developers think what would be great improvements but instead could actually slow them down.

It's these interactions where we gain the knowledge QA needs to become advocates for the end-user.

If you know the basics of how your development team writes code, you should have a good idea of how things should (and should not) work.

Talking to our teams, we get to know each other and learn how we work best together. We geek out about movies, board games, video games, kids, hobbies, and pets. We question things we see and things we know. We are not afraid to raise questions. We read articles on testing and tell our teams the things we've learned. We want everyone to get excited about testing topics. We talk to managers to find out what the plan is, what their expectations are, and where QA can best be of service. We talk to ourselves, especially when we hit a snag and just can't figure out where we went wrong.

Sometimes We Develop, Too!

So, you don't know how to program? Learn a programming language. If you know the basics of how your development team writes code, you should have a good idea of how things should (and should not) work. Automated testers know they have to develop code or scripts, so they have at least some idea of how programming works. QA personnel look at the user interface and data tables and employ manual test cases to help automated tester counterparts write tests in any number of frameworks and any number of programming languages. The best way to help the automated tester is to assist them in writing some tests.

Not only do we speak about what we are testing, we also talk through the issues with developers: QA is often used as a sounding board. There will be times where you find yourself on a team and suddenly you and a developer will start talking through an issue

they are having. Having general knowledge of the programming language they are using gives QA a working idea of what the developer is saying, or maybe you have just enough knowledge that the SQL query the developer asked you about makes sense. Taking a testing point of view, you can help solve the problem by offering a better query.

But this is still not all we do.

QA Has User Experience Opinions

The end-user can take many forms, including client, customer, student, or the beta tester who is using the software that the team has been working on for the past sprint. QA needs to put themselves in the shoes of those who are going to use the system in unexpected and complicated ways. Surprisingly, testing for a wide variety of end-user personas often validates how many of the requirements are correct. Pretending to test the software as the end-user brings out the extra paths that need to be tested, because the "happy path" should not be the only way to achieve a quality result.

Great QA will drive the user experience (UX) team, BAs, and developers crazy, because we're the ones who point out that that image is three pixels too far to the left and that button is the wrong color blue. It is what gives you permission to tell the team that red is probably not a good color because there are color combinations to avoid for those with color blindness, like red next to green.

This is your chance to use the system and bang on the software in ways the actual end-user will. You get to make up stories in your head about the user and what they are doing, and go about inventing new ways of interacting with the product that no one else considered when they wrote the stories.

QA gets to play in the UX space, too. QA typically reacts to tickets specifying what needs to be tested. You get a ticket about a button in a certain shade of blue, asking you to right-click and inspect that button. You'll examine the CSS to make sure that blue is exactly what the BA put on the ticket. You click through the workflow and make suggestions about why it takes nine clicks to perform something that should take no more than four clicks.

While you're at it, you suggest that the header text is not bold enough for the user to understand what is being called out. Do you stop there? Of course not. This workflow is attached closely to another one that you know is a little less heavy, so you make another suggestion that, because they are so similar, maybe they should match. You even find yourself defending your suggestion and advocating for the user.

Although subjective, QA can offer a different perspective about the experience of using the application. Does it look good? Does it feel good? Does it do what I need it to do without having to wait forever for it to respond? These are all valid questions we ask ourselves almost every time we test something.

We learn that our users aren't actually using the system in the way we thought. We study the tickets as we would exams. We take online courses about UX and sit down in classrooms to learn about a new programming language. We discover that we aren't the only ones out there with this situation; our internet searches come up with the same questions we have.

And—no surprise—there is more we do.

QA Can Be a User Advocate

This list can go on and on; it really all depends on what we can do in a day. We test and we always will. But we have to be much more than just testers.

We have to gather requirements and question what we see. QA has to effectively communicate with our team, our businesses, our users, our bosses, and each other. We have to learn about the code so we can be better teammates and perform better quality assurance. We come up with new and exciting ways to test like an end-user and make suggestions for improving the application. We are creative when we look at what we are testing, and we pick out colors, images, and buttons to determine which workflows are most beneficial for the end-users. We are students with never-ending curiosity. We are sponges soaking up any information we can get on what will make us better at what we do.

Whenever somebody says, "Well, you're just a tester" or "So, that is all you do?" tell them that being in QA is so much more than being a tester. We're advocates for the end-user just like them.

[BSM] agiese79@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!

Artificial Intelligence and Health Care: Predicting Patient Deterioration

By Pamela Rentz

As part of a medical research partnership with the US Department of Veterans Affairs, the team of scientists and engineers at DeepMind, the artificial intelligence group at Alphabet (Google's parent company), will work on the global issue of patient deterioration during hospital care.

Read More

What's Our Job When the Machines Do Testing?

By Geoff Meyer

It's a safe bet that testing jobs won't be taken over by machines anytime soon. However, those of us in the test industry would be wise to heed cross-industry applications of analytics and machine learning and begin staking out the proper role of the machine in our testing domain. What could AI mean for testing?

Read More

Is There a Bias against Manual Testers?

By Josiah Renaudin

Manual testing might not be as all-important as it once was, but it's still needed if you have any hope of delivering software at a quality you can be proud of. How we create software is going to continue to change, but the burden of that change needs to be handled by more than one group within the industry.

Read More

Unit Testing? Consider Taking a Rain Check

By Hans Buwalda

Unit testing is a great way to verify software at an early stage and to ensure that modified functions are still working as specified. However, unit tests are not a magic wand. Rather than making such a big testing effort, consider taking "rain checks" for certain tests.

Read More

Does Testing Truly Improve Software?

By Ingo Philipp

Without revealing problems, there is no problem-solving, since we can't solve something we aren't aware of. Each solved problem is one fewer problem in the software—and the software is improved each time a problem is removed. But it's not testing alone that improves software. So when does that happen?

Read More

The Future of Testing: VR and AR in Mobile Apps

By BJ Aberle

With the ability to experience virtual and augmented reality using mobile devices on the horizon, the potential for these new technologies to go mainstream is huge. New test environments, configurations, and interfaces will require testers to change their methods, so get out of your comfort zone—and your office chairs!

Read More

Dealing with a Change-Resistant Manager

By Naomi Karten

With almost any change, whether a trivial adjustment in procedures or a large-scale organizational change, people will vary in their receptiveness to it. But if you and your teammates have some good ideas to improve processes and your manager keeps shutting them down, you may be dealing with a change-resistant manager.

Read More

How Mobile Developers Can Take Advantage of Android Instant Apps

By Nicholas Roberts

Google's Instant Apps offer users a way to easily engage with software without having to download the app. Any Android developer can now develop Instant Apps—or adapt their existing apps to support this feature—streamlining the way users interact with their content, store, or game. Will you modify your native app?

Read More

Accelerate Your DevOps Transformation by Focusing on Culture

By Adam Auerbach

The toughest part of a DevOps transformation is the cultural changes required to make it successful, so to accelerate your transformation, figure out what they are as soon as possible. Explore your company's attitude toward innovation and the tools you have and how you use them, and it will make the change easier.

Read More

The Role of the Test Manager in Agile

By Jeffery Payne

In traditional software processes, test managers are responsible for all management aspects of their team. Agile, however, is self-directed, so teams handle all the usual duties. Still, there is a role for test managers in agile, and it's much more strategic than it was before. Here are the opportunities for the role.

Read More

What the Future Holds for Cloud Computing

By Anabel Cooper

Thanks to the development of cloud architecture away from a server-client pattern, those in the software industry will be able to embrace heterogenous cloud services that can only run when needed. Consequently, we are due for a cloud renaissance. Here are some predictions for what the future holds for cloud computing.

Read More

Stress and Project Management: 5 Ways to Relieve Project Pressure

By Krystle Dickerson

Project managers have many sources of stress. They are responsible for the performance of their team members, and often for whether a project is successful or fails. Keep the pressure off with these five tips for reducing your stress and ensure a smoothly functioning team.

Read More

6 Ways to Protect Your Organization from DDoS Attacks

By Pradeep Parthiban

During a DDoS attack, no one can use your application, which will result in loss of business. Brand reputation also tumbles if customers can't access your site or become casualties of the data breaches. However, there are some proven practices for preventing DDoS attacks—and for what to do if you fall victim to one.

Read More

A Definition of Done for DevSecOps

By Gene Gotimer

In DevOps, we have a software delivery pipeline that checks, deploys, and tests every build. The goal is to produce a viable candidate for production, so we have to look at many different aspects of quality, including security. To be sure we hit all the crucial marks, we should have a definition of done for DevSecOps.

Read More

A Primer for IT Project Sponsors: 10 Steps for Success

By Payson Hall

Much time has been spent examining the project manager's role in IT project success, but the role and duties of project sponsors are often overlooked—even though sponsors are an essential element of success (and failure). Here are ten rights and obligations a project sponsor should perform to improve project success.

Read More

Testing Your DevOps Is Just as Important as Testing Your Software

By Alan Crouch

Many DevOps engineers fail to test their automation code in the same way they test the software they deploy. It's crucial for software to have tests, and this should apply to infrastructure-as-code software too, if we plan to change and improve this code with no worries about breaking automation in our DevOps pipeline.

Read More

Be Indispensable: Cross-Train like a Testing Athlete

TESTERS ARE INCREASINGLY LOOKING FOR WAYS TO TAKE THEIR SKILLS FROM HELPFUL TO INDISPENSABLE. PRO ATHLETES' APPROACH TO CROSS-TRAINING MAY BE THE "SECRET SAUCE."

by Bonnie Bailey | *bonnie.bailey@motorolasolutions.com*

In 2016, with his seventh win, Jimmie Johnson tied Richard Petty and Dale Earnhardt for the most NASCAR Cup Series championships of all time. How has Johnson stayed on top in a physically and mentally draining sport for so long?

These days, winning in racing at the highest level requires more than skilled driving, a fast car, and interminable patience—all the top drivers have those things in mostly equal measure. Also, cars don't break down like they used to, which means far less is left to luck and chance. Adding one or two more hours driving the car isn't likely to give Johnson that much of a boost. Similarly, if an elite marathoner added five more miles to his already arduous daily run, any benefit would be negligible. The marathoner must look outside his training runs to become a better runner.

To wit, Johnson employs a tactic commonly used by athletes to get better at their particular sports: cross-training.

Johnson is famous for his triathlete-style workout routines, which he believes confer enormous benefits for both the mental and physical requirements of racing. When other drivers are sleeping or watching TV, Johnson is doing long-distance runs, bikes, and swims. The combination of these activities—the driving-specific training and the triathlon training—produce a synergistic interaction. His cross-training is like a superhero's sidekick, helping the main strength to show up more often, better, and stronger.

The path to improving a weakness is theoretically simple: Learn and practice basic fundamentals and you will become better at anything, whether that's sports, leadership, communication, or technical skills. But weakness isn't what Jimmie Johnson needs to improve. Like all top athletes, Johnson wants to get better at one thing that earns him income, fame, and success.

It's not much different for those of us in software development. Companies don't want software people—whether that's engineers, testers, project managers, or engineering managers—who are merely good at what they do. They want tech folks with distinctive strengths. Weaknesses, as long as they aren't fatal flaws, don't have to be deal-breakers.

Becoming indispensable to a software organization doesn't require a software technologist to become a jack of all trades. In fact, trying to become a jack of all trades is a great way to stay stagnant as other technologists wield one brilliantly valuable and unique skill to climb right on past you. The trick of success is not to go from terrible to passable at something, but to go from good to great. That results in a competency that matters both to organizations who offer to pay for it and to you, the practitioner.

As easy as it is to improve on a weakness, improving on a strength is much more difficult.

As easy as it is to improve on a weakness, improving on a strength is much more difficult. Like Johnson would attest, doing more of what you already do well will only yield small improvements. To enhance what is already a strength, one must work on complementary competencies.

Take a mid-level software tester who genuinely enjoys the art of testing and knows she does well because she stays employed, she finds bugs, and she is told by others that her bug reports are well-written. But this tester is not satisfied with being good enough and wants to step up her game. She has a nagging feeling that she isn't as good as she could be, but she's not sure how to get stronger at what she does every day. Doing the same thing day in and day out isn't increasing her skills that much. One possibility she might consider is to learn test automation, even though her passion is really investigative testing.

The application of complementary competencies would prescribe that this tester not try to learn test automation because she isn't really interested in it and she would be starting from scratch. Because her work experience lies in testing, her focus needs to be on test cross-training.

I have found some unconventional yet effective ways to cross-train yourself to advance in your profession. Making a list of these core skills and fundamentals can serve as a starting point. Let's pick one and develop it as an example.

According to James Bach, a core skill of software testing is the ability to "reflect upon, describe, explain, and defend your work." [1] This makes sense because a tester's job requires thoughtfulness and effective communication. Sloppy skills in either area can doom a tester to the hall of mediocre. This tester is highly regarded for writing good bug reports, but what will make her excellent in this area is her ability to think, reason, and communicate clearly about her test strategy.

If strategy means the overall plan that leads to tactical execution, what can the tester do to cross-train to become a better test strategist? An approach I'd recommend is to develop the ability to discover information and make connections. Another competency to develop would focus on learning how to represent test strategy to others.

Cross-training in information discovery and connection making could include studying systems thinking, modeling, and logical reasoning (or fallacies). Study in these areas will help the tester learn to test her own assumptions about what she knows.

Some suggestions for cross-training include playing strategy games like Risk, Axis and Allies, or chess. Listen to the popular "You Are Not So Smart" podcast, and read and digest "Harry Potter and the Methods of Rationality" at hpmor.com. All these sources can help develop critical thinking that will benefit your work.

Cross-training in representing the strategy means intentionally practicing verbal and written communication skills. An obvious choice here would be for the tester to join Toastmasters International, where she will learn from safe outsiders where she is communicating well and where she needs to improve. The tester could find someone to mentor them at work, as teaching others is a great way to learn how to distill information for particular audiences. Spending more time writing generates critical feedback, whether that is on a blog or participating in a fiction-writing club.

Developing written skills can help the tester learn to eliminate unnecessary words and phrases in her communication and focus on the important stuff.

This concept of building a strength by engaging in complementary behaviors can be applied to any software discipline. In a world where so many feel compelled to jump on board the next big thing, thus inhibiting mastery of any one, a focus on becoming an expert in what one is already good at is the best way to become indispensable. [BSM]

CLICK FOR THIS STORY'S REFERENCES

LINK TO OUR ADVERTISERS

Agile + DevOps East	4
AgileCxO	2
IIBA	11
QMetry	21
SQE Training—Live Virtual	13
SQE Training—On-Site	29
STARCANADA	25
STARWEST	3

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.