CONTINUOUS PRACTICES

eGuide

TECHWELL™

The widespread adoption of agile and DevOps practices have placed the software industry firmly in the age of "continuous everything." From continuous testing, continuous development, and continuous deployment to continuous integration and continuous improvement, our development cycles must embrace the need for speed to stay competitive in the marketplace.

Explore this eGuide to learn what your organization must do to keep pace in a continuous world.

In this Continuous Practices eGuide

Has Continuous Deployment Become a New Worst Practice?
Software development has been moving toward progressively smaller and faster development cycles, and continuous integration and continuous deployment are compressing delivery times even further. But is this actually good for businesses or their users? Just because you can deploy to production quickly and frequently, should you?

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

Avoiding Continuous Bugs: Speed and Quality in DevOps
Lots of DevOps initiatives focus on speed and frequency of deployment without an emphasis on quality. Bad testing practices in DevOps only deploys buggy software faster. Here are some tips to move toward a more effective testing process that supports a continuous delivery approach—without sacrificing quality.

Embedding Performance Engineering into Continuous Integration and Delivery
In the world of continuous integration and continuous delivery, the importance of ensuring good performance has increased immensely. While functional and unit testing are relatively easier to integrate into these processes, performance engineering has typically raised more challenges. Here's how you can mitigate them.

Why You Need to Be Doing Continuous Integration
It's usually easy and inexpensive to set up a continuous integration environment for either an agile or a waterfall project. Perhaps the most obvious benefit of CI is the elimination of the integration phase that existed in traditional waterfall projects, where we typically slip the worst on deadlines. But there are many other benefits to continuous integration that you may not have considered.

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

Insight from Around the Industry

Additional Resources

While continuous deployment has the potential to be a very positive trend, never underestimate people's propensity to misuse a good thing.

Has Continuous Deployment Become a New Worst Practice?

By John Tyson

From waterfall to RAD to agile, software development has been moving toward progressively smaller and faster development cycles. Continuous integration and continuous deployment are compressing delivery times even further, which is good for business and good for users—or is it?

In the '90s, we had automated nightly builds where all code that was checked in would be built and ready for deployment. Then came continuous integration, where a build was done as soon as the code was checked in and automated unit tests could be run on this new build. Now comes continuous deployment, which allows us to painlessly deploy as often as needed.

The ability to deploy to production easily and rapidly is nice to have. We've all been in a situation where we need to make an emergency fix due to a major production bug or lived through a painful release process. However, the ability to deploy quickly and frequently, while very appealing to companies, is often not to the users.

I recently heard people from one agile company boast about how they "deploy software to production one hundred to two hundred times a week." Based on a five-day workweek, this averages twenty to forty deployments per day—or, based on a ten-hour work day, two to four per hour. What could the downside of this be? From a user's perspective, here is an example.

As a web-based Microsoft Team Foundation Server user, I can recall several times where the look and feel of TFS changed without notice, which threw everyone off for a while. Making matters worse, the

"enhancements" were questionable at best, because they usually involved more clicks to perform the same function.

However, the real danger with continuous deployments is that some companies are moving away from having their software tested by professional software testers. If there's no pain or cost for deploying software, who cares if it's buggy? "We do continuous deployment, so we'll just push out more fixes," these software development managers say.

In my experience, testing has never been fully appreciated, and continuous deployment may reduce this appreciation even further. Testers are often the low man on the totem pole, and in tight economic times, they are usually the first ones to be let go—with the assurance that the developers will just have to test more, and that the rest of the testers' tasks will be replaced with test automation. Automated tests run like magic after each build, and they run quickly, too.

But before considering automated testing as a replacement for professional testers, answer these questions:

- Are your automated tests being checked for false negatives or, worse yet, false positives?
- Are you periodically injecting data that should cause your automated tests to fail and verifying that they are, indeed, failing?
- Is your automation fully exploring the product, seeking to learn and adjust as tests are run?

Remember, not all tests can be automated. Tests based on timing can be difficult to automate, if not impossible.

Some business people and development managers may think, "Okay, our QA person found some bugs that automation did not, but would a 'real user' find them? Would they care? Let's just deploy it and see what happens." They may see users as unpaid beta testers or crowdsourced testers.

But how many users will take the time to report a bug? How well will their bug reports be written? How easy will the company make it to report bugs, and how responsive will communications between the company and bug reporter be? Also, there are certain kinds of critical bugs that should never happen in production. Are you confident that your test automation is sufficient to find them?

Test automation focuses on testing functionality but has no clue about usability. Who is using your program—another program or a person? If it's a person, then you will want a professional tester exercising that application with a keen eye on usability. An alternative to having a professional tester do this is having a group of users participate in a formal usability test. This is a good idea, but in all my years of testing, not a single application I've tested has ever gone through a formal usability test. I wish more companies would perform usability testing, but it is faster and cheaper to use a knowledgeable tester.

The need for speed in software development and continuous deployment has caused some traditional things, like software versions visible to the user and release notes, to fall by the wayside. For companies that are deploying ten updates a day to an application, how will users know that changes have been made without release notes or versioning? In my experience, testers were the ones who put together release notes. If a user wanted to report a problem, testers are usually the ones who try to reproduce the bug. With no software version being reported, how will users or testers know if the bug has been fixed? Agile is based on feedback, and cutting out versions and release notes hinders feedback. It will also limit your analysis if you're collecting metrics.

For the company that boasted about deploying to production up to two hundred times a week, is this good, bad, or ugly? I don't know the number of applications involved, so I can't say. If they have ten apps and we assume half are new or improved features and half are bug fixes, we have five new features and five bug fixes per app per week at the low end and ten each at the high end. If this is a new application, then adding five new features a week may be reasonable, but after six months, this would be excessive. Might this be an indication of feature creep or gold-plating?

The same figures apply to bug fixes. Five bug fixes a week may be very reasonable in the beginning, but if this figure remains steady, then it may be a good indication of inadequate testing.

We'll have to wait and see if continuous deployment has a positive or negative impact on software quality in the long run. If companies feel they can replace professional software testers completely with automated tests, I think we know the answer. While continuous deployment has the potential to be a very positive trend, never underestimate people's propensity to misuse a good thing. {end}

Continuous Testing, Continuous Variation

By Hans Buwalda

With the arrival of continuous integration/continuous delivery (CI/CD) the notion of continuous testing (CT) is taking center stage. Knowing that comprehensive tests are running smoothly can be of great benefit for the CI/CD pipeline. But running tests can be both time and resource consuming, not to mention that tests can become boring and rigid. Using the repetitive character of CI/CD for testing can be a way to address this.

Reducing the Number of Test Runs

In a well-run CI/CD process, rebuild times will depend on how many components are affected by changes and need to be rebuilt, including execution of their unit tests. However, the amount of functional testing needed when there is a rebuild is less straightforward. Functional tests, in particular business level tests, can have a wider range than a single component and may need to run in multiple configurations and/or environments.

The need for many test executions can be addressed with technical means, like more hardware, headless browsers, API testing, etc. However, an additional interesting approach can be to leverage the fact that tests are executed so often. Instead of running the same test over and over again, bring in some variation. I like to call this "continuous variation," as a next step up from continuous testing.

The first way to embark upon continuous variation is in the choice of which tests to run. Obviously tests that address the components that were changed would always be executed, but to gain more coverage, pick additional end-to-end tests, either randomly or in a round-robin fashion. That way, all tests get their chance to run and find possible side-effects of changes.

For configurations and environments, a similar principle can be considered. Use a list of configurations, select or provision one or more configurations and/or environments from that list, each time a test run is due. The list could be compiled or generated using pair-wise testing. For more information on pair-wise testing, see www.pairwise.org.

Making Tests Less Boring and Rigid

Continuous variation can also help address another issue with automation: its tendency to make tests rigid. Once bugs found by the tests have been resolved, the automated tests can have a hard time finding any new issues.

However, knowing that tests will be run many times makes it possible to follow a data driven approach. Instead of hard-coded input values, use variables for which the values are picked from a row in a table, which then can include expected outcomes. In a traditional data-driven approach, a test will loop through all rows, but in Continuous Variation it will only take one row each time the test is executed. No looping takes place, since that is already taken care of by the repetitive CI/CD process. This applies even more if the variation is done in multiple places in an end-to-end test, since in the course of time many combinations of values will be used.

Leveraging the continuous repetitions that are part of the CI/CD process can be one way to reduce testing times while increasing their effectiveness. If and how this can be meaningful can be very different from case to case and is best for the DevOps teams to decide upon. {end}

Avoiding Continuous Bugs: Speed and Quality in DevOps

By Jeffery Payne

There is an old saying in testing: Automating a bad test only gives you bad results faster. The same thing applies to DevOps.

Lots of DevOps initiatives focus on speed and frequency of deployment without an emphasis on quality. Bad testing practices in DevOps only deploys buggy software faster. Here are some ideas about how to avoid this situation.

Produce Working Software

Agile principles make it clear that the only measure of progress is the frequent delivery of working software. This means a user story is not complete until it has been built according to its acceptance criteria, adequately tested, and approved by the product owner.

Make sure your definition of done for both individual user stories and entire sprints includes adequate functional and nonfunctional testing against story, integration, and end-to-end testing criteria. Use continuous regression testing to make sure new code does not break old code. Also, demand that your developers do adequate unit testing so implementation bugs are caught early in the process.

Move toward Producing Releasable Software

DevOps expands on that last principle by advocating that all software also be releasable—including being adequately tested in a production (or production-like) environment.

Traditionally, access to production environments has been out of the reach of development teams, but this is changing quickly. By embracing a DevOps philosophy that integrates release management,

Your goal should be to make as many of your environments as production-like as possible to allow earlier detection of defects that have typically not been found until late in the software release process.

Leverage Test Automation

Test automation is critically important when seeking to deploy software frequently into production. When beginning an effort to increase test automation to support DevOps delivery, focus on areas that provide the biggest bang for the buck first:

- Unit testing to identify defects early and validate code changes during check-ins
- Smoke testing to make sure testing environments are properly set up and your software is ready for testing
- Frequent regression testing across your code base to detect code that has broken something that used to work

environment and production support, and application monitoring into your teams, frequent testing on production environments is now more realistic and can be done as part of each sprint.

If your production environments are too costly or complex to fully replicate, establish production-like environments that still allow some late-lifecycle testing to be shifted left and performed continuously. And you don’t need to solve all your environment access issues at once in order to make gains toward producing releasable software.

Be sure to look at the entire test process. Establishing automation to provision environments, install software and databases, and tear them down after testing is completed can save a lot of time during testing and support the goal of continuously delivering software to downstream test environments and, ultimately, into production.

Use these tips to move toward a more effective testing process that supports a continuous delivery approach—without sacrificing quality. {end}

Test automation is critically important when seeking to deploy software frequently into production.

TECHWELL™

Embedding Performance Engineering into Continuous Integration and Delivery

By Anjeneya Dubey

In the world of continuous integration and continuous delivery (CI/CD), where technology companies are rapidly deploying code and infrastructure changes so their application can gain a competitive advantage, the importance of ensuring good performance has increased tremendously. While functional and unit testing are relatively easier to integrate into these processes, performance engineering has typically raised more challenges, especially in analyzing the results and pass/fail decision-making.

My company is currently going through this transformation, and embedding performance evaluation of the services as part of CI/CD is a must. Here are some of the challenges we faced and changes we made across the board to make this happen.

Cultural challenges: If you want to evaluate the performance of every line of code that is checked in, culture is the most important yet difficult change required. There were several areas where we struggled. Performance was an afterthought; teams were focused on functionality more than performance. There was also a lack of understanding of performance and scalability needs of the product, and performance engineers were not part of the agile teams and weren't included in the agile ceremonies.

To help alleviate these challenges, we needed to shift the performance lifecycle to the left. We empowered our developers to test, allowing good performance to be baked into the code.

Process changes: We also had to make significant changes in our Scrum process to create awareness of performance. We made nonfunctional and performance requirements part of the functional requirements, which required us to have scalability needs, API contracts, and service-level agreements clearly defined at the story level. This enabled us to learn what "performance-ready" services mean to our customers.

We mandated performance as part of the definition of "done" for sprints and included performance results during sprint demos with all stakeholders.

Tools and accelerators: At this point we had to make sure we had the right tools and accelerators to create, execute, and analyze performance tests at the sprint level, ready to be integrated into the CI/CD pipeline.

We built a performance engineering platform that manages the testing cycle of our services, leveraged functional test scripts to collect browser-side response time, created self-contained tests that create test data before each test and destroy it after each test, developed a central repository for all performance metrics, created an algorithm for dynamic thresholds for pass/fail of each build and individual REST APIs for test status and decision-making, and automated defect creations and notifications through our performance engineering platform.

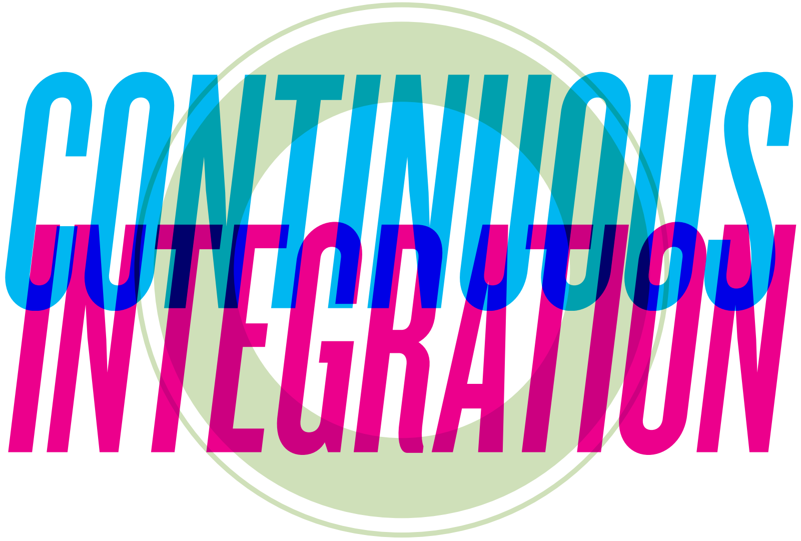
All our services are in the cloud, making it easier for the performance environment to expand and contract as needed and adding flexibility to build and kill new environments. Such benefits can go unnoticed, but they are a huge driver to implement our test framework that can scale to address business needs. {end}

Why You Need to Be Doing Continuous Integration

By David Bernstein

Perhaps the most valuable of all the software development practices today is continuous integration (CI). It's usually easy to set up a CI environment for either an agile project or a traditional waterfall one, and the benefits are huge.

Continuous integration essentially means merging all developer work to a shared mainline several times a day. It's more than just setting up a build server, and it's also more than version control. Continuous integration is an automated system, and when new source files are created or existing files are modified, it automatically kicks off the build to compile the system and run the entire suite of automated regression tests.



I won't be going into how to set up a continuous integration system, or even some of the best practices around it. There are several articles and other sources on how to do this. In fact, many CI systems are open source or otherwise free, so they can be downloaded and installed at essentially no cost. Even the ones that are commercial products are typically pretty inexpensive, especially considering the value they provide. Instead, I'll detail some of the many benefits of continuous integration that you may not have considered.

Some Surprising Benefits of Continuous Integration

Perhaps the most obvious benefit of continuous integration is the elimination of the integration phase that existed in traditional waterfall projects. This phase is where we typically slip the worst on deadlines.

During many projects I've worked on, we'd be in good shape during design, development, and even unit testing, but when it came time to integrate all the code together, we found integration bugs that we never realized existed. What should have been a simple process of stitching all our modules together turned out to be integration hell, taking far longer than we ever anticipated. It was not uncommon to recognize in the eleventh hour, during integration, that we made some serious mistakes in the design and implementation of modules that made our code incompatible with other pieces. We would then have no choice but to go back to the drawing board and redesign and recode big parts of a system weeks before we were supposed to deliver it. As a result, I've seen project delivery delayed by weeks or even months.

This does not happen when continuous integration is used on a project because we get continuous feedback about whether the code we recently wrote integrates into the system. Rather than a single integration point at the end of a project, with continuous integration there are hundreds or even thousands of tiny little integration points along the way, which makes for a far more stable product that has far more predictable results.

> *Feedback is like shining a light into that dark room. Suddenly we see what's there—and not there.*

Developers on a team using CI should integrate their code at least once per day. But if the team is told that they must integrate any code they've written before they leave at the end of the day, something interesting happens. The last person to integrate usually finds some of the worst problems, so in order to avoid that, many developers integrate more frequently. It's not uncommon for a developer to start integrating their code as they build it, perhaps several times a day or even multiple times each hour.

Doing this gives instant feedback so we can tell where a problem exists in our code that affects other code in the system. As we integrate and get feedback throughout the day, we begin to think about how we build our software a little bit differently and start to focus on practices that will make our code more robust and less brittle.

Working on a large system can often feel like fumbling around in the dark. Feedback is like shining a light into that dark room. Suddenly we see what's there—and not there.

People naturally adjust to feedback when it's given to them on a timely basis, especially impartial feedback like the kind we get during CI. When we see that our code is constantly breaking other code, we naturally think of ways to make that not happen. Even "cowboy coders" have to take a step back and see how their work integrates with the rest of the team or they end up spending late nights in front of the computer rather than at home. I've found that once a team adopts continuous integration, it automatically leads to building higher quality software because higher quality software is what passes the integration server faster.

How to Get Started with Continuous Integration

Continuous integration is easy to set up. All you need is a build server and some free software. It's really a matter of just sitting down and setting it up, but there is one subtle aspect to continuous integration: the quality of the automated tests you have.

Automating unit testing means you don't have to do manual regression testing, which can take an enormous amount of time and resources on traditional projects. Unit tests are only effective if we're testing the right things, so there it is a bit of an art to building the right tests.

I believe the best way to build unit tests is with test-first development because it also informs our design and assures us that we have a full suite of tests for all of our implementation. Doing test-first development, however, doesn't replace a quality assurance effort, so you may also want to add additional acceptance, integration, and system tests to assure your code is behaving as you expect.

Test-driven development is a discipline in and of itself, but unfortunately, there aren't a lot of good resources available to learn how to do TDD properly. Tests have to be complete but not redundant, and they have to test the right things so they run fast. There are many schemes for staging builds when projects become very large. We want our builds to run quickly, or else we'll tend not to use continuous integration. A minute or two to run is fine, but when we start

Team members work together better because they realize what they do impacts each other's code.

to take ten, twenty, forty minutes or more to build the system, then developers tend to shy away from doing it—and that's not at all what we want.

The correct way to do continuous integration is to do a complete build on our local machine before we build on the integration server to assure that everything works correctly and that the first layer of unit tests all pass. Then we promote our code to the mainline and run the build again on the integration server. This can be a more extensive build and test environment. Depending on the size and complexity of our system, we may do nightly builds so as to run a full suite of tests that we can't run during the day. Staging builds in this way lets us work with very large systems that require a great deal of processing power to determine if they're running correctly. Even very large projects can be handled using staged builds.

Continuous Integration: Better Collaboration, Better Software

By doing continuous integration, our code is always ready to ship. This means that at any point in time we can push a button and produce shippable code. Now, we may not want to do this very often, but it's good to know we can at any point in time. Suddenly development is decoupled from marketing or other services that are involved in the delivery of software.

Continuous integration means that there is no integration phase like there is in a traditional waterfall project, so we're not deluding ourselves in situations where we might end up having more work than we thought due to integration bugs. Using continuous integration, we get a true measure of our progress, and the worst bugs are found sooner.

Another huge advantage of CI is that it promotes "fearless refactoring" because the risk of breaking something else while changing code and not detecting it quickly is dramatically reduced. This gives developers confidence that they can continually improve code maintainability and quality without breaking other things in the process.

But I think one of the most important benefits of doing continuous integration is how it impacts the culture of the development organization. Team members work together better because they realize what they do impacts each other's code. It's very easy to get siloed on traditional waterfall projects and end up building software in a virtual vacuum, but continuous integration gives us a touchstone for the rest of the team and helps us build software that's more compatible with other team members' code.

So, continuous integration has many benefits and very low cost. Why not try it? **{end}**

TECHWELL™

Measuring Objective Continuous Improvement in DevOps

By Logan Daigle

DevOps has become the perennial buzzword on the technology scene, but it means a lot of different things to different people and organizations. Some definitions are heavy in regards to automation, and others are heavy in regards to culture, but the true definition is somewhere in between.

My favorite definition of DevOps is the one from Gene Kim in *The DevOps Handbook*: "the emerging professional movement that advocates a collaborative working relationship between Development and IT Operations, resulting in the fast flow of planned work (i.e., high deploy rates), while simultaneously increasing the reliability, stability, resilience and security of the production environment."

Much more important than how we define the work is the fact that the technology industry is leaping at the opportunity to "implement DevOps."

This is a great thing, but in order to bring about the implementation successfully, there are a few things that an organization should consider when starting and attempting to sustain DevOps. Here are six steps to follow on the road to "DevOps awesomeness":

1. Figure out the long-term goals of using DevOps to facilitate change in your organization
2. Determine the short-term goals that support your long-term goals—things that can be accomplished soon so that you know you are on the right path
3. Create a value stream map of your organization, product, application, system, or team
4. Decide on metrics to capture that allow you to make objective, data-driven decisions
5. Choose a release period in which you can capture metrics as a baseline
6. Use the baseline to figure out where your bottlenecks and constraints are

These six steps will help you get off to a good start on your journey because it is incredibly important to know where you are starting. Once you are three, six, or even twelve months into your DevOps adoption, you will want to know what progress you have made, and you won't be able to figure that out unless you have benchmarks from the beginning.

It's important to remember that research shows that implementing DevOps gets hard before it makes things better for your organization. This is where the culture of DevOps should kick in. While things are difficult to implement and figure out, as the teams dig out of the hard part, they will begin to build a culture of trust, courage, and determination that will help them be the type of organization that is incredibly competitive and valuable in its market.

Also remember that DevOps is a journey that should never end. The law of entropy teaches that things constantly want to be in a state of disorder. This also applies to technology organizations, and teams must work to constantly and continuously improve themselves in order to fight entropy.

"DevOps awesomeness" means that organizations figure out how to build culture and automation that inspires continuous improvement, continuous learning, and continuous feedback in order to make it the best it can be. **{end}**

Insight from Around the Industry

"I've been getting a lot of questions this year—much more than years past, all of a sudden—about continuous integration, then DevOps this and DevOps that. And there's been a sort of an underthread of anxiety from testers about, well, what does that mean for me? Because people look at the DevOps process and they say, 'Where does test fit?'"
» Melissa Benua

"I've met with various leaders who think automation is the magic pill, but it's not. How you automate and how much you automate is the key. Skills and the team effectiveness is also a large ingredient in this calculation."
» Jennifer Scandariato

"There's certainly the companies that you read about all the time that are doing continuous delivery and continuous deployment, and have been doing it for a long time, but they're on the leading edge. There's a lot of people that are just getting going, particularly those that have industrial-strength legacy types of applications."
» Jeffery Payne

"Artificial intelligence technology that can mimic the human eye and brain has opened the door to automated visual testing at scale, and behind that first door is a whole house full of applications and use cases that will be addressed with visual AI. This is exactly why we believe a new category is emerging called *application visual management.*"
» Gil Sever

"So, everyone is fully automating their builds and deployments and testing, and they start releasing really, really fast. The release is becoming smaller and smaller, but then the first thing happening is that quality starts dropping, and that repeated itself project after project."
» Alon Eizenman

"The most important thing for getting test automation up and running is to actually run test cases! If you're not running your tests, you're not receiving any value from them, no matter how well-designed your test suite is and how well it covers your top business risks."
» Maximilian Bauer

"While many teams now focus on implementing continuous integration and continuous delivery practices, they fail to embrace continuous planning and other engineering practices such as continuous design, proper branching and merging strategies, continuous data management, and continuous testing during the release management process."
» Uday Varma

"What does not go away with a serverless infrastructure is the need for proper automated testing, continuous integration (CI), continuous delivery (CD), and a DevOps mindset. Giving a developer unfettered access to production code almost never ends well. While they may fix whatever immediate issue is at hand, that easy-access cloud editor that can change production code will also allow defects to creep in. But there are tools that can help eradicate defects during delivery by gating code with automated static analysis, unit tests, and even functional UI tests."
» Glenn Buckholz

The most important thing for getting test automation up and running is to actually run test cases!

TECHWELL™

Additional Resources

MORE INFORMATION FOR SOFTWARE PROFESSIONALS

STICKYMINDS™ A TECHWELL COMMUNITY

StickyMinds is home to thousands of software testing resources, including articles, *Better Software* magazine articles, conference presentations, and interviews with industry notables.

CLICK HERE

NARROW YOUR SEARCH TO A SPECIFIC TYPE OF CONTENT:

StickyMinds Articles

StickyMinds articles cover a wide range of software testing topics, such as continuous testing, test automation, test management, test design techniques, agile testing, test process improvement, test tools, and much more. **Click here** to read continuous testing articles on StickyMinds.

Better Software Magazine Articles

Better Software magazine is a digital quarterly filled with expert analysis, how-to articles, and real-world case studies covering all aspects of software development. **Click here** to join StickyMinds and access *Better Software* magazine articles about continuous delivery and testing.

TechWell Conference Presentations

Couldn't make it to a TechWell conference to sharpen your continuous testing and delivery skills and knowledge? TechWell conference presentations are available to StickyMinds members soon after a conference ends. **Click here** to join StickyMinds and access conference presentations related to continuous testing and delivery.

Interviews

Each year, TechWell interviews dozens of software professionals, including well-known thought leaders, seasoned practitioners, and respected conference speakers. **Click here** to read, listen to, and watch interviews with continuous testing and delivery experts.

TechWell Conferences

STAR CONFERENCES TECHWELL EVENTS

Agile + DevOps CONFERENCES TECHWELL EVENTS

AGILE TESTING DAYS USA

TechWell conferences include STAR conferences, Agile + DevOps series, and Agile Testing Days USA. These popular events are full of keynotes, tutorials, and classes covering everything from agile testing to UX testing. Learn from experts in the field and network with your peers to get the most immersive conference experiences possible. Learn More.

SQE TRAINING A TECHWELL COMPANY

SQE Training offers both foundational and specialized agile, DevOps, and software testing courses to help your team deliver better software. Learn more about their continuous testing, test automation, and test planning courses at sqetraining.com/testing

TECHWELL™