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