Why agile isn't agile without continuous delivery

Shipping at the end of each sprint is within your reach. Here's how to get started.

BY IAN BUCHANAN

Agile practices can be viewed in two ways: recipes and sensors. People with a lot of software experience wrote down the practices that made them most productive. In addition, the practices reveal the ways in which we are unproductive by amplifying the pain they cause.

Continuous delivery is both part of the agile recipe and a great revealer of inefficiencies. Moreover, in order to reap the benefits of agile, you need to be agile through all phases of the software development lifecycle. Iterative planning and development don't count for much if your user stories and bug fixes languish in a repository for weeks on end before stakeholders and customers ever get a look at them.

You're only as #agile as your ability to ship frequently, and without drama.

The essence of agile is "inspect and adapt." Now, you probably don't need to put your build and deploy system under a microscope to help you assess whether it's hindering your team's agility or helping it. And the fact that you're even reading this is a good sign that you've already put your system in the "hindering" category. So now it's time to adapt.

Living with continuous frustration

There is a common state of software that I call "continuous frustration". It is the absence of any continuous integration or continuous delivery practices. It feels like this...

- You commit to main and anxiously wait for someone to blame at you.
- Your main code line is unstable.
- Bugs hide in a tangle of many, many code changes, and you dread the effort ahead to find the bug (let alone fix it) because so much time has passed since you worked on that area of code.
- When testers want to test a feature, they have to annoy all the developers to learn the current status and find a working build.
- Releasing requires a code freeze. Somebody becomes an artificial bottleneck to code changes so he can stabilize the release. Meanwhile, nobody really stops working. You just stop committing code changes, which means a torrent of unproven code floods your repo like so much untreated sewage as soon as the freeze is lifted.

If that story is hauntingly familiar, know that there is hope.

Think about how "inspect and adapt" improved your planning process, and extrapolate that into the development and delivery processes. Imagine you could detect problems in every commit you made. Now imagine you could detect problems on your local workstation, even before you made a commit. What you just imagined gets to the heart of why agile needs continuous delivery: developers need fast feedback.

The great thing about embarking on the journey to continuous delivery is that it starts with you. You don't have

RELATED TUTORIAL

Continuous Delivery Tutorial

Try this tutorial →

SUBSCRIBE

Sign up for more articles

Email

email@example.com

Subscribe

to sell it to anybody or convince your team that you should be more like Etsy. And nobody is using technology so unique that the basic principles, practices, and tools of continuous delivery don't apply. There are some easy things you can do right now that will make your life better.

Start with a script and a server

In past, there was only Make but many devs were bitten by whitespace bugs. Later, Ant substituted whitespace for angle brackets. Now, you can skip these previous generations. Sure, you'll still see a lot of open-source projects using Maven or even Ant. Just chalk it up to old habits. But you can resort XML-based build languages as a fallback option.

The latest generation of build languages are much easier to learn and use. In most cases, you can use a build language written in the same language you're using to build your application which makes the whole thing easier on yourself (and your team).

Language	Recommended tool
Java	Gradle
Assemby, C, C++	Make
.NET	psake
PHP	Robo
JavaScript	Grunt
Python	Paver
Ruby	Rake
Perl	Module::Build
Objective-C	Bash
GoLang	Make

The "continuous" part of continuous delivery means getting feedback on every commit, which is why build servers will automatically listen to your repository and trigger a build when something changes. Being continuous also means fixing the build when it breaks. As a developer, you benefit from isolation of change. The best way to keep bug fixes from getting complicated and unwieldy is to keep them from piling up in the first place. Build servers, like Bamboo, are easy to install. Initially, the purpose is simply to execute your build script in a clean environment. In a sense, the first thing your build server does is help you detect problems in your build script.

Start detecting problems automatically

As you can imagine, your build server can detect much more than problems with your build script. There are easy things like turning on compiler warnings (for some languages) and harder things that every agile coach will champion like automated tests. That's because manual-only testing is a bottleneck for continuous delivery. If you're just getting started with automated tests, find the layer – unit, integration, acceptance, UI – that will provide the most rapid feedback.

Let's end the controversy: automated testing is NOT a threat to manual testers.

In fact, human testers make excellent guides when it comes to deciding what (and what not) to automate. A smart automation strategy hinges on the question, "What kinds of problems would be most valuable to detect?" You need experienced humans who specialize in software quality to answer it.

The seeds of continuous delivery

Anyone can start. But if it's just you, nobody will recognize it as continuous delivery. The next step is to bring these building blocks to your team.

Draw inspiration from books and articles on continuous integration and agile testing. Later, you'll find more and more reasons that being ready to deploy is useful. Then you

can draw inspiration from continuous delivery and continuous deployment.

The starting point for everyone is pretty common but the end state will be something unique to your product and business.

SHARE THIS ARTICLE

IAN BUCHANAN

While Ian has broad and deep experience with both Java and .NET, he's best known as a champion of agile methods in large enterprises. He's currently focused on the emerging DevOps culture and the tools for enabling better continuous integration, continuous delivery, and data analysis. During his career, he's successfully managed enterprise software development tools in all phases of their lifecycle. He has driven organization-wide process improvement with results of greater productivity, higher quality, and improved customer satisfaction. He has built multi-national teams that value self-direction and self-organization. When not speaking or coding, you can find Ian indulging his passions in parsers, meta-programming, and domain-specific languages. Follow Ian at @devpartisan.

TUTORIAL

Continuous Delivery Tutorial

In this guide, we'll see how you can use Bitbucket Pipelines to adopt a

ARTICLE

Continuous Delivery Pipeline 101

In a continuous delivery pipeline, automated builds, tests and