Continuous Integration: The Cornerstone of a Great Shop

Articles

Posted by:

Posted on : 2007/10/2 22:35:42

This article shows how continuous integration can help to keep projects on track with a rapid feedback on the product status.

Author: Jared Richardson, http://www.jaredrichardson.net

I did a lot of lawn mowing when I was younger. My brother and I tried to make our summer money by asking real estate agents if we could mow the lawns of their absentee clients. We'd usually land one realtor each year and they'd give us enough business to keep us busy all summer. One of the things I learned is how hard it is to cut a straight line when you're mowing a wide yard. When I was in the middle of the yard, I felt like I was cutting a straight line, but then I'd get to the end of the row and look back to discover a crooked line. It always amazed me how something could seem so right and be so off course.

Timely Feedback

A software project can be a lot like mowing a yard. Even though we try to move in a straight line, and we think we are, later we look back and are amazed at how far the project ran off course.

Whether mowing yards or building software, we need timely feedback to help keep us on track. Looking back at completed software projects, or lawns, shows you where you missed the mark, but it's usually too late for that project. We need feedback while we're still in the midst of the work. I never found a good way to get that feedback for my lawn mower, but I have found a guide for software projects. I use continuous integration systems to keep my projects on track.

Continuous Integration

Mike Clark calls this type of system a "virtual build monitor". This extra team member keeps an eye on your project and lets you know when things start getting off course. If you invest in a good automated test suite, you'll quickly catch all sorts of errors that traditionally pull good projects off course.

The more shops I get to observe, the more I'm seeing that continuous integration plays a vital role in keeping a shop on course. In fact, these days I'm telling people that I've learned one of the basic, fundamental principals to keep both you and your project on target.

Do you want to make your product a great one? Do you want to be the best developer you can be? Then make a solid continuous integration system a first-class member of your team and the cornerstone of your shop. A good CI system eliminates many of the problems that prevent you from working on the product, your career and your craft.

A continuous integration system does several things automatically.
* Monitors your source code
* Compiles after every change
* Tests your compiled code
* Notifies the developers of problems as soon as they occur

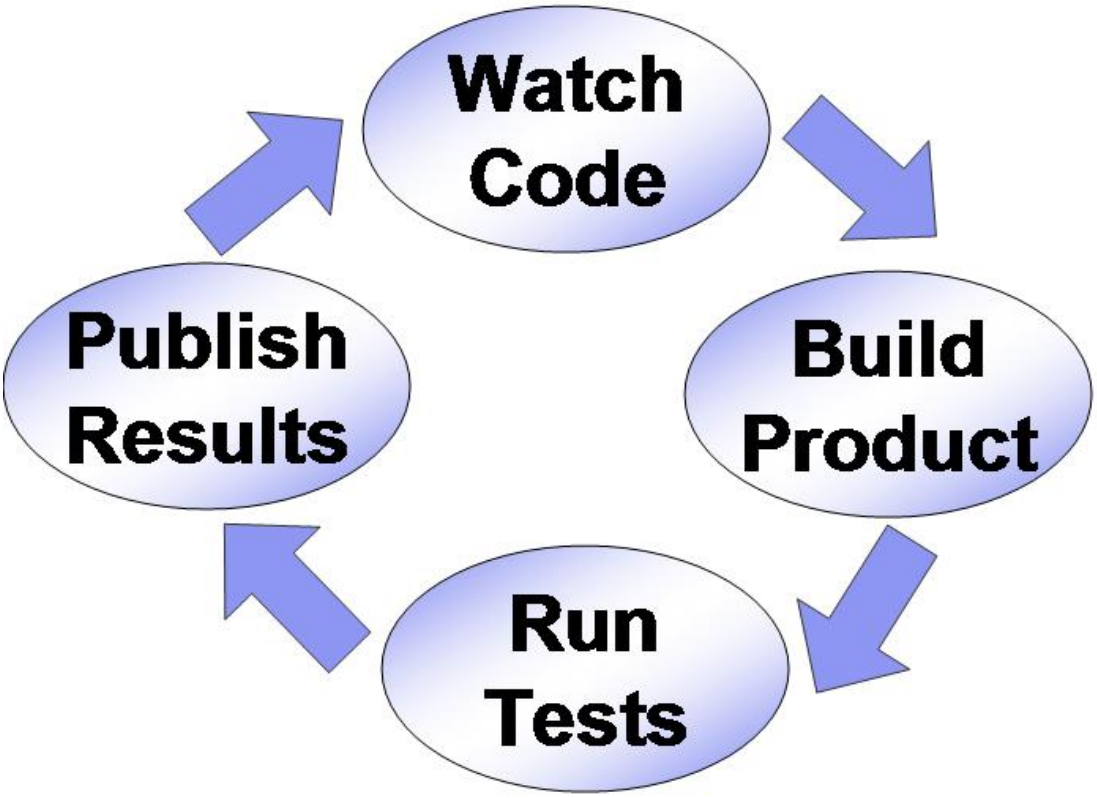

Figure 1. Continuous integration actions
As we move forward, keep an open mind and try to see where each step could’ve helped you in the last few months. Then, when we’re done, I’m going to point you to a Continuous Integration system that is trivial to install, easy to use, and open source to boot.

Let's look at what a continuous integration system is and why it helps so much.

The Steps of CI

Continuous integration systems all have a few common steps.

First, CI systems monitor your source code. The system usually watches your source code management system (CVS, Subversion, Perforce, ClearCase, Visual Source Safe, etc) but most systems can also monitor other resources, like file systems. This is how the software knows it's time for a build. Every time your code changes, the CI system checks out the latest version of your code.

Second, the software compiles your project. The system runs your existing build scripts by wrapping them in an Ant script. In this step, your CI software is requiring you to have a scripted build. If you builds are not robust or repeatable, your CI tool will expose this flaw. It will force you to have a clean

build system.

Third, CI systems tests yours new build. The tests are created (or wrapped) in an Xunit framework (Junit, Nunit, HtmlUnit, jsUnit, etc), which means you have access to dozens of test frameworks that range from unit testing to browser click through testing. When you set up a system to run tests, people are more likely to write the tests. They'll also contribute the tests they've been hiding on their own machines.

Lastly, your CI system will notify everyone of the results. The developers or testers who just changed the code will get email telling them how long the build and test took, how many tests passed, how many failed, etc. Your system will also archive the results to a web page.

However, the publishing step is very configurable. You can publish in a variety of interesting ways beyond a standard web page. You can publish to a custom web page, XML log, email, an instant messaging client, or even a Lava Lamp. The publish step is an extremely flexible way of sharing your build results.

What's the big deal?

There are several key practices that continuous integration either requires or encourages. They are source code management, scripted builds and test automation. Much of the benefit that comes from using a CI system actually comes from the foundational practices that a CI system requires.

Don’t get me wrong. CI adds plenty of benefit as well. It’s just that many day-to-day problems go away when you use these other practices first.

Code Management

One of the first things your CI system will do for you is make sure you've got your source code organized and (hopefully) into a source code management system. After all, your CI software can't watch a code tree you can't identify. The first practice CI encourages is good source code management.

This benefit will seem very elementary to many people, but I've seen shops that still use network drives and zip files. Quite a few developers still haven't discovered source code management.

Proper source code management doesn't take much time at all once you've learned how to use it. Like any good tool, you’ll save much more time than you’ll spend learning to be effective with the tool.

You’ll save the time you normally spend reconciling code differences by hand, not to mention rewriting the work that careless coworkers overwrite from time to time. Code collisions and lost work are common issues but a good source code system also merges your changes for you, maintains a history for each file, and more.

If you're not using a proper source code management system, I urge you to rethink you

position. It's a huge time saver.

A Scripted Build

The second thing your CI system will require is a scripted build. Moving to this step requires a level of build automation. Fortunately, this is easy to add. There are many tools available, both commercial and open source, that solve this problem for you. You still have to understand how to build your product, but these tools will keep from learning the different command line options for javac or jar on different operating systems. Look at tools like Ant, Maven, and Rake.

Like good source code management, a scripted build provides many benefits.

For starters, your teammates aren't all busy building their own version of the build script. Everyone needs to build and developers, being clever, will all find a slightly different way to solve the same problem. When you have a single build script, everyone's building the same way. It's okay is someone still wants to build differently (an IDE maybe?), but they need to have the ability to build the same way that everyone else does.

Don’t ignore the maintenance savings either. You'll eventually improve the build script, find a bug in it, or decide to make it faster. With a single script, you do the work once time. When everyone has his or her own build method, everyone solves the same problem repeatedly. What a waste of time!

When you build your code the same way, everyone gets the same version of the product. This means that the testers report problems in the same version of the program the developers run.

Without a canonical build script, you don't always get everyone on the same page. In fact, the customers, testers and developers often run very different versions of the same product and then wonder why they can’t reproduce the same issues. If you’ve had trouble reproducing your customer’s bugs, then start here. Is everyone running the same version of the product?

Test Automation

Another practice that a CI system encourages is test automation. Writing and running tests is a huge milestone for many shops and is one of the hallmarks of a great shop. I think test automation is the core of why a CI system adds such benefit. People who recognize the benefit of automating common tasks tend to be more strategic thinkers. They automate everything possible, including building and testing, and it frees them up for more interesting work. (Of course, this doesn’t eliminate manual testing, but that’s another topic.)

What is an automated test?
* Binary
* Automatic
* Repeatable
* Binary

A test with a binary result passes or fails unambiguously. There’s no question about whether

the test succeeded. Sometimes a test will return a result that requires a judgment call from a tester. The odds are good that you don’t need this.

Work hard to make your tests clean and binary. Write them so they evaluate the result and tell you if it passed or failed.

Automatic

If the test isn't automatic then someone has to set up an environment, start the test, click a button, or look at the results. When this happens, the test becomes interactive again. Much of the benefit of test automation is lost.

You've created a hybrid test somewhere between an interactive test and an automatic test. Instead of letting a small number of testers baby-sit a large number of tests and continually adding more tests, you'll have a large number of testers looking at log files all day long. Half-automated tests are certainly better than pure interactive testing but they fall far short of where you can be. Work hard to make your tests completely automatic, including the determination of the pass or fail status.

Repeatable

An automated test also needs to be repeatable. A good test doesn't give you different results for three out of five test runs. If your tests aren't repeatable, break the tests down into smaller tests. Eventually you'll isolate the problem area and as a bonus, you’ll have new tests created for your test suite.

Don't forget about external dependencies either. You can rebuild and restock database tables cleanly before each test run with tools like Ant's SQL task or dbUnit (http://dbunit.sourceforge.net). A dirty database table can introduce all sorts of variation into a test run. (You may want to create a small but representative data set to load for your testing runs.)

Leverage Yourself

An automated test is a great way to leverage your experience and expertise. As an expert on your product, you probably know how to test parts of it in a way that few other people can. You can encode that knowledge in a reusable format by creating an automated test. This makes your experience available without diverting your attention. Sometimes a co-worker will run the tests, other times you will. In other cases, a program will run them.

Let these bits of your expertise exercise the product while you do other things, like go home on time, or stay late and solve problems that are more interesting. These tests might run while you are coding or at home sleeping, but you are doing something else. Your tests are working in the background.

Getting Started

Sometimes people won’t install a CI system because they don't have tests ready to run in the system. There are enough benefits from fast compile cycles to justify using Continuous Integration, so don’t wait. You don’t wait to see a doctor until you’re not sick

anymore, right? Having the CI system keep your compiles clean will free up some of the time needed to start writing tests as well.

I've also found that people are much more likely to write automated tests if they’re sure the tests will be used. By providing a CI system, you have a place to house your tests and run them immediately. This is the best way I know to encourage test creation. People want to create things are used and this assures them the tests they create will run regularly.

The best way to get started with Continuous Integration is to start using an existing software package. I'm going to point you to CruiseControl on Source Forge (http://cruisecontrol.sf.net/). Since version 2.3 Cruise Control comes with an embedded servlet engine (Jetty- http://jetty.mortbay.com) and a sample project. You can download the project and see CC running in less than five minutes. Then, to add your own project, just copy the bundled example. It’s very easy to get started.

The CC team has a great write-up on how to run the binary release of CruiseControl. Visit http://cruisecontrol.sf.net/ and click the "Getting Started" link on the left.

In Conclusion

The teams I know running smoothly and cleanly always have continuous integration in place. It's a practice I respect more everyday.

I’m seeing this practice overshadow all others. Teams that run smoothly use continuous integration. They respect the system instead of tolerating it, and the developers treat the notifications seriously. When the system says something is broken, these teams address the problem quickly. These teams insist on CI coverage from the first day of a new product.

Other shops, even those who are using a CI system but ignoring it, are very different. They live in turmoil. Heroic efforts are not the exception but the rule. In fact, these teams always seem to be running behind. They always have a crisis issue to resolve or deadline to meet.

They work and live in a perpetual twilight of stress and problems. They’ve lived there so long that they think it’s the only way to write software. Sadly, these teams tend to burn people out. I’ve been there and it’s no fun. Creating software can be a great joy -and is- when done right. CI can’t solve every problem, but it can remove several categories of problems that would otherwise clutter your day and slow you down.

If you’re not using a Continuous Integration system, try one out this week. Get a system installed and leave it running for one month. At the end of that month, turn it off if you don’t see the benefit.

Don’t be surprised if you find yourself missing the system the first day it’s gone. You might just become one of the developers who insists on continuous integration coverage on your new projects.

Resources

Martin Fowler and Matthew Foemmel on Continuous Integration
http://martinfowler.com/articles/continuousIntegration.html

CruiseControl page http://cruisecontrol.sourceforge.net/
CI product page: http://www.jaredrichardson.net/ci.html
Scripted build links: http://www.jaredrichardson.net/buildscripts.html
Mike Clark’s automation blog: http://www.pragmaticautomation.com
Jetty http://jetty.mortbay.com/ Originally published in the Spring 2006 issue of Methods & Tools