# Continuous integration

Build your team's agility with faster feedback. Because you only move as fast as your tests.

BY DAN RADIGAN



Nothing builds–or destroys–agility like a team's commitment to continuous integration (CI). That might sound ominous (especially if your team has yet to embrace CI), but there's good news. Regardless of the technologies a team uses, chances are there's a continuous integration and automated test framework that will work for their codebase.



## What is continuous integration?

Continuous integration is an agile and DevOps best practice of routinely integrating code changes into the main branch of a repository, and testing the changes, as early and often as possible. Ideally, developers will integrate their code daily, if not multiple times a day.

## Benefits of continuous integration

Investing in CI results in fast feedback on code changes. Fast as in "within minutes" fast. A team that relies primarily on manual testing *may* get feedback in a couple hours, but in reality, comprehensive test feedback comes a day–or *several days*–after the code gets changed. And by that time more changes have occurred, making bug-fixing an archeological expedition with developers digging through several layers of code to get at the root of the problem.

That is decidedly *not* fast.

> Agile and DevOps teams deliver quality software fast, without death marches or heroics. CI makes this possible.

## Protect quality with continuous builds and test automation

How many of us have downloaded the latest source code and found it didn't compile or had a significant bug? What a productivity killer!

Two practices keep us out of that situation:

**Continuous builds:** Building the project as soon as a change is made. Ideally, the delta between each build is a single change-set.

**Test automation:** Programatic validation of the software to ensure quality. Tests can initiate actions in the software from the UI (more on that in a moment), or from within the backend services layer.

Think of these two practices like peanut butter and jelly: taste good separately, taste great together! Continuous integration pairs continuous builds with test automation to ensure that each build also assesses the quality of the code base.

And remember: to fully realize the benefits, a team must also have the discipline to pause development and address breakages *right away*. The energy a team invests (and make no mistake: it's an investment) in writing tests and configuring the automation is all for naught if builds are allowed to languish in a broken state. Protecting the investment in CI and protecting the quality of the code base are one and the same thing.

## Testing in CI: Unit, API, and functional tests

CI runs have two major phases. Step one makes sure the code compiles. (Or, in the case of interpreted languages, simply pulls all the pieces together.) Step two ensures the code works as designed. The surest way to do this is with a series of automated tests that validate all levels of the product.

### Unit Tests

Unit tests run very close to core components in the code. They are the first line of defense in ensuring quality.

**Benefits:** Easy to write, run fast, closely model the architecture of the code base.

**Drawbacks:** Unit tests only validate core components of software; they don't reflect user workflows which often involve several components working together.

Since a unit test explains how the code should work, developers can review unit tests to get current on that area of the code.

### API tests

Good software is modular, which allows for clearer separation of work across several applications. APIs are the end points where different modules communicate with one another, and API tests validate them by making calls from one module to another.

**Benefits:** Generally easy to write, run fast, and can easily model how applications will interact with one another.

**Drawbacks:** In simple areas of the code, API tests can mimic some unit tests.

Since APIs are the interfaces between parts of the application, they are especially useful when preparing for a release. Once a release candidate build passes all it's API tests, the team can be much more confident shipping it to customers.

### Functional tests

Functional tests work over larger areas of the code base and model user workflows. In web applications, for example, HTTPUnit and Selenium directly interact with the user interface to test the product.

**Benefits:** More likely to find bugs because they mimic user actions and test the interoperability of multiple components.

**Drawbacks:** Slower than unit tests, and sometimes report false negatives because of network latency or a momentary outage somewhere in the technology stack.

Teams often find that as they get closer to the actual user workflow, the speed at which automated tests run decreases. HTTPUnit is quicker because it's not a full-fledged web browser. Selenium can only run as fast as the web browser, but has the advantage to run across multiple web browsers in parallel. Despite these caveats, functional tests are enormously valuable and provide feedback much faster than human testers ever could.

Speaking of which...

Some testers view automated tests as an existential threat. This thinking is short-sighted, and couldn't be further from the truth. Freed from the drudgery of repetitive testing tasks, testers can spend time on risk analysis, test planning, and building other skills–like learning to code!

## Make your continuous integration fast

At Atlassian, we strive to keep developers innovating and our code bases healthy. We place a big emphasis on tightening the developer's "inner feedback loop"–the time required to build changes and get test results.

Running automated tests can quickly add up and draw out build duration. One strategy is to parallelize automated tests across several servers, or "build agents," so the CI

server is actually running 2, 20 or even 200 tests simultaneously. With cloud technologies, CPU can easily scale to meet the needs of your development team as your test suites grow. But CPU is not unlimited. Test each area of the code completely, but not redundantly. Redundant tests bloat build duration (and waste CPU). The faster engineers get the green light, the faster they can move on to the next item in the backlog.

## Branching and CI: a match made in Heaven!

Many teams avoid branching because of painful merges. With newer technologies in version control like Git, both branching and merging become easy. To ensure that the primary code line ("main" in Git parlance) remains healthy, run the same level of continuous integration on all development and stable version branches as well. When the build passes on a branch, the team has the confidence to merge that code upstream.

With branching, continuous integration, and test automation, teams can be productive and innovative while still protecting code quality. If you're ready to take the next steps, check out our step-by-step guide to getting started with CI.

This is agile development at its best: delivering working software regularly, with minimal technical debt and without compromising ingenuity.



DAN RADIGAN

Agile has had a huge impact on me both professionally and personally as I've learned the best experiences are agile, both in code and in life. You'll often find me at the intersection of technology, photography, and motorcycling.