Continuous Integration: An Agile Necessity

Articles

Posted by:

Posted on : 2010/10/25 3:01:59

Agile development is the future of modern software engineering. Companies that have implemented it successfully have seen great improvements in their software—both in cost, stability, and in the utility of the software itself. However, some companies have struggled, finding it difficult to adapt their processes and culture to work in an Agile environment. One of the areas that best highlights the cultural shifts required to become an Agile organization is Continuous Integration (CI).

Author: Micah Hainline, Asynchrony Solutions, http://www.asolutions.com/

Continuous Integration is made up of several important components, including a source control repository, an automated build system, tests, and a CI server. As source code is changed in the repository, the CI server automatically detects the changes, runs all of the tests, and builds the final product, with a goal of detecting problems early in the process, and keeping all of the pieces of the code integrated with one another at all times. There are excellent open-source tools available for all the components of a CI system, and they usually beat the commercial products hands down. Hudson is an excellent choice for the CI server, although there are many other good options available.

Continuous Integration is about people more than tools. In order to be effective, it requires that developers check their code changes in to source control on a frequent basis, keep the tests complete and up-to-date, and move quickly to address test failures when they occur. Each of these activities can require a major change in mindset from a more traditional model in which developers each have their own area of expertise, where testing is something done by a Test Engineer after the project is finished, and where “Integration” is typically a six-month line item on the Waterfall process Gantt chart.

Agile is about flexibility, stability, and responsiveness to change. Test Driven Development (TDD) is at the core of that but Continuous Integration is what makes TDD work in the real world. When a team writes code test-first and test-driven, the end result is a lot of small tests covering all of the functionality and that are ready to let the team know when any piece of code breaks. Unfortunately, they do no good at all if the team doesn't run them, and run them often. Tests are about stability, but they're also about flexibility. Having a strong set of tests that are run after every change gives the developers the confidence they need to make changes—changes that are necessary to keep the code base healthy and development focused on the goals of the project, which are refined daily. Because test failures on the CI server are highly visible and because feedback is immediate, the CI server acts as an extra team member at the daily stand-up meetings, pointing out any issues that have yet to be addressed, and keeping the team focused on quality.

Most large projects are broken up into several libraries, usually each with their own set of tests. Because of the time it takes, and the fact that many development environments don't have a

simple way to run all of the tests at the same time, developers will rarely go to the trouble of running every test. This is especially true if they don't think their changes will affect a particular area of the code. A CI server is a much more reliable and cost effective way to ensure the tests don't get skipped. Integration tests—tests that exercise the application as a whole—are particularly onerous for the developers to run as they take additional time and often take over the workstation while they are executing. They are also very important, and have the capability to catch problems that otherwise could be missed by a unit test. In fact, they can take the place of the manual test scripts most projects rely on during the Regression Testing phase of a release, which is often weeks long. Integration tests are particularly useful for ensuring that problems never resurface after they have been fixed the first time, and are well worth the investment both in time to run and time to write.

Integration tests are necessary. The only question is whether they will be automated and run on a regular basis, or whether they will be run manually before each release. Often when deciding between these two choices only the cost is considered—the cost of the test engineer's time versus the cost of creating and maintaining executable tests. An important factor that is often left out of the equation is the time it takes to turn out each release. If the project team has to spend two weeks performing final testing every time it needs to release another version of the product, it isn't really very Agile. When the team knows that every test scenario is being run automatically, it greatly reduces the amount of time taken for the simple mechanics of a product release, which greatly streamlines the efficiency of the team as a whole. This is not to say that there is no place for regular QA work, but a CI server makes a big difference in the time is takes to get a stable release out the door, and it keeps the team closer to that elusive goal: clean, bug-free code ready to ship on a moment's notice. A CI server also can keep track of older builds, making it as easy to ship a previous release as it is to ship the current one. It can even be used to serve up the latest installation directly to the client, assuming downloadable media is preferable to a DVD and FedEx Overnight.

Given the benefit, why is it that most projects in the industry don't even have an automated build of any kind? The answer is that it requires a cultural commitment at all levels of the company. Without that commitment, Continuous Integration ends up on the back burner. How, then, does an organization go about building a culture that supports Continuous Integration? One of the keys is to build on early successes, and this means that the first attempts cannot be halfhearted. Agile doesn't work when it isn't pervasive. The benefits of a well-tested system under Continuous Integration allow any developer to make changes with little fear of making a mistake. If only one or two of the developers are writing tests, there will be holes—gaps in which errors will be made, time will be lost, and sentiment will start to build up that “this just doesn't work.” In my personal experience working with developers from many backgrounds, the person least likely to get behind an Agile initiative is the person who thinks they've already done Agile before. The key to building a culture that will support Continuous Integration is to make sure it works the first time. By all means, pick a small project or project without a lot of baggage, but make sure it happens right the first time. Out of your early successes you will create people in the organization who believe in the process because they have seen the benefits for themselves.

Continuous Integration takes some work but if you are committed to Agile it is not a luxury—it is a necessity. The effort put into it will reflect in code quality, responsiveness of the team to change, and in the confidence of a job well done.