Scaling Continuous Integration

R. Owen Rogers
ThoughtWorks, Inc.
Peek House, 20 Eastcheap
London, United Kingdom
orogers@thoughtworks.com
http://www.thoughtworks.com

Abstract. Of all the Extreme Programming practices, continuous integration is one of the least controversial – the benefits of an integrated, streamlined build process is something that software developers immediately recognise. However, as a project scales up in size and complexity, continuous integration can become increasingly hard to practice successfully. By focussing on the problems associated with a growing project, this paper describes a variety of strategies for successfully scaling continuous integration.

1 Continuous Integration

The practice of continuous integration represents a fundamental shift in the process of building software. It takes integration, commonly an infrequent and painful exercise, and makes it a simple, core part of a developer’s daily activities. Integrating continuously makes integration a part of the natural rhythm of coding, an integral part of the test-code-refactor cycle. Continuous integration is about progressing steadily forward by taking small steps.

1.1 Integrating Continuously

Integration should happen continuously, and continuously is more often than you might think. The frequency of integration will vary from project to project, from developer to developer, and from modification to modification. However, as a goal and a good rule of thumb, developers should integrate their changes once every few hours and at least once per day.

Learning how to integrate so frequently requires practice and discipline. Fundamentally, an integration can occur at any point when the code compiles and all the unit tests are passing. The challenge is learning how to write software so that you never stray too far from this point. If you are testing at the right level of granularity and are refactoring regularly, then you should never be more than a few minutes away from this point. This means that you are almost always in a position where you can launch a new integration.

Deciding when to integrate is all about controlling risk. When making modifications in a high traffic area of the code base or when conducting broad refactorings like class renaming or package reorganisation, there is an elevated risk...
of impacting other developers or of having merge conflicts when committing. The longer that developers go without integrating, the greater the likelihood of conflicts and the larger the effort required to resolve those conflicts. As the effort of integration increases exponentially in proportion to the time between integrations, best practices dictate that when making high-risk changes a developer should start from a clean workspace, focus only on required modifications, proceed with the smallest logical steps, and then commit at the earliest opportunity.

A successful integration is a measure of progress. It provides feedback that the new code runs correctly in the integration environment and successfully interoperates with the rest of the code base. Code sitting unintegrated in a developer’s workspace simply does not exist. It is not part of the code base, it cannot be accessed by other developers or tested by the customer. Only when it has been successfully integrated is the benefit of the new code realised.

1.2 Continuous Integration Tools

In order to integrate as frequently as possible, the integration process must be easy to launch. If the integration process requires multiple manual steps then it is easy to forget steps or make mistakes. Ideally, the integration should be initiated by invoking a single, simple command. Build tools such as make, Ant, or NAnt are excellent candidates for scripting the integration process to achieve this.

Automated integration servers, such as CruiseControl (Java), CruiseControl.NET (C#) or DamageControl (Ruby), automate the integration process by monitoring the team’s source control repository directly. Every time a developer commits a new set of modifications, the server will automatically launch an integration build to validate the changes. When the build is complete, the server notifies the developer whether the changes that they committed integrated successfully or not. Effectively, integration becomes as easy as checking in code. Using an automated integration server not only makes integration easy, it also guarantees that an integration build will happen. There is no danger of developers forgetting to validate their changes after checking in.

1.3 Practicing Continuous Integration

Tools play an essential role in practicing continuous integration. Having the right set of tools is what changes integration from a painful and time-consuming task into an integral part of the development process. However, with the power that they bring, it is easy to focus on the tools and lose sight of the fact that continuous integration is a practice – it is about what people do, not about what tools they use.

As a project starts to scale, it is easy to be deceived into thinking that the team is practicing continuous integration just because all of the tools are set up and running. If developers do not have the discipline to integrate their changes on a regular basis or to maintain the integration environment in good working