System Integration and Build Management

Christian Schröder and Roman Antonov

May 29, 2006

Contents

1 Introduction

Extreme programming and other agile software development methods have introduced plenty of new software development methods. In this elaboration we will explain three methods that are going hand in hand with XP's principle of Continuous Iteration and thus are named Continuous Integration, Continuous Builds and Continuous Tests. On an abstract layer these three methods are mainly independent, in practice they depend and benefit from each other. For each method we will introduce tools that can be used for their realization. Each method deals with problems of mid- to large-scale software development projects. For each method it is an important precondition that every developer is working with the latest version of the source code and all sources are available from one location. For this reason the production codebase should reside in a source control system

2 Continuous Builds

For technical reasons every software development project should be build "from scratch" at least once a day. From scratch means you are starting with a clean system, the latest version of the code and all resources are checked out from a version control system. This eliminates the possibility of files not checked in to the version control system or linker failures. In mid- to large-scale software development projects a build from scratch should be done more often, because more developers working on a project increase the probability of dependencies of their work and their mistakes. Building the entire system than can be a time consuming process. Hence, dedicated build servers, running a build engine like Ant or Maven [www6] should do this work. These servers build the entire system multiple times a day, typically every hour, with given parameters and notify every developer who was involved in the last changes to the code if a build fails.

Directives

- Build the system on your local machine before checking in!
- Before leaving work, wait for a successful build message from the build server!
- If you are notified of a failed build, immediately fix the problem!

Benefits

- Reduction of problems with files not checked in, linker errors, etc.
- The dedicated build server reduces the need of builds from scratch on the developer machines and though saves time.
- When the project reaches a stable state and all parts of the project are integrated, regular builds can serve as demo-versions for the customer.

Disadvantages

- The maintenance overhead of a build server.
- The cost of a dedicated build server

Continuous Builds are not specific for agile software development projects. This technique has often been adopted in traditional software development projects and today is a common practice.

3 Continuous Tests

As described in 'Continuous Builds' regular builds can serve as demo-versions provided that the project is in a stable and integrated state. The application of Continuous Tests can lead to this required stability. Continuous Testing requires the developer to write code testing the functionality of his work. These test are called test cases. They test very specific behavior of the system as the input can only be concrete values. For organizational reasons Test Cases are often organized as groups, called Test Suites. As every change to the code can lead to unexpected side effects it is recommended to always run an entire system Test Suite after every change. This can be a process even more time consuming as a complete build from scratch. Thus the existing infrastructure for Continuous Builds is used to set up a test engine. Tools like Maven [www6] provide an easy way to integrate testing frameworks like JUnit [www7], which is a very common tool for the purpose of writing, organizing and execution of tests. This tool, originally written for Java has been ported for most programming languages. The build server executes an entire system test after every successful build of the build engine and also notifies the developers involved in the last changes to the code if a test does not pass at 100 percent.

Directives

- Write test cases for every functionality you implement!
- Test your changes locally before checking in!
- If your changes to the code do not pass the tests at 100
- If notified of a failed test by the server, immediately fix the problem!

Benefits

- Reduces the general probability of bugs.
- Especially reduces the probability of bugs caused by side effects.
- Can be easily integrated into an existing Continuous Builds infrastructure

Disadvantages

- The maintenance overhead of a test server
- High effort of writing test cases

Continuous Test are specific for agile software development projects as the use of Test Cases was developed from XP [www2]. Continuous Tests are often combined with XP's method 'Test First'.

4 Continuous Integration

In most software development projects system integration is the most time-consuming process. It is the process of bringing together the work of several teams at the end of a project iteration, merging their work and fixing bugs caused by side effects originating from the cooperation of the different sub modules. This kind of integration is often called "Big Bang"- Integration [www1]. The Extreme Programming paradigm is the source for a different kind of integration strategy, called "Continuous Integration" or "Relentless Integration" [1]. Continuous Integration tells you to integrate more often, even continuously. Continuously means to integrate several times a day or ideally after every completion of a task. If you follow XP principles your tasks are kept very small. But how can that work? The integration of trivial tasks becomes trivial itself. The integration work is broken into small pieces. When you merge your part of the software with the rest of the system after every change, you know that if a bug occurs the code that caused the bug must be the one you've just done. This disburdens you from the time consuming process of stepping through a huge amount of code and searching for the bug line after line.

Directives

- Integrate as often as possible!
- Write Test Cases for your integration code!
- Meet with members of other teams if you got problems integrating their code!

Benefits of Continuous Integration

- Integrating continuously means to have a running system at any time of the project. It means there is always a successful build available you can show to your customer and it gives the developers a better feeling for the project as a whole. In turn, this makes integration easier.
- With a running system at hand all project members (developers AND management) got a better overview of the project's status.
- Additionally the duty to integrate after every task empowers developers to keep their tasks small and simple, making the code better readable and easier to maintain.
- As a manager you got a better overview what developers are doing because of short check-in cycles.

Disadvantages

- The immediate impact of checking-in incomplete or broken code acts as a disincentive to developers to provide frequent check-ins
- The maintenance overhead

5 Conclusion

All methods mentioned in this elaboration require a severe discipline from the developer. Continuous Builds can't work if the developer doesn't know if the code she has just written causes the problem or the incorrect code was checked in earlier. This also applies for Continuous Tests. Continuous Integration and Continuous Tests always need some time to get accepted by the developer, because both methods seem to require a lot of additional work by the developer, but in total these methods lead to better code in less time. The methodologies can strongly benefit from each other because they can use the same infrastructure. They follow the same iterative approach and thus integrate easily into a agile project structure.

6 References

[**www1**] http://www.martinfowler.com/articles/continuousIntegration.html

[**www2**] http://www.extremeprogramming.org/

[**www3**] http://en.wikipedia.org/wiki/Continuous_integration

[**www4**] http://www.xprogramming.com/xpmag/whatisxp.htm

[**www5**] http://cruisecontrol.sourceforge.net/

[**www6**] http://maven.apache.org/

[**1**] A Practical Guide to eXtreme programming, D. Astels, G. Miller, M. Novak (2002)

[**2**] Agile Software Development, Alistair Cockburn (2001)