The Addison Wesley Signature Series

A Martin Fowler Signature Book

Continuous Integration

Improving Software Quality and Reducing Risk

Paul M. Duvall
with
Steve Matyas
Andrew Glover

Forewords by Martin Fowler and Paul Julius

Continuous Integration

❑ ❑ ❑ ❑ ❑ ❑ ❑ ❑ ❑

Summary

This chapter outlined the key risk areas that CI will help to mitigate, such as database integration, testing, inspection, deployment, feedback, and documentation. Table 3-1 provides an overview of the material covered in this chapter. You will find that by mitigating these risks using CI practices, you will improve software quality.

TABLE 3-1 Summary of Risks and Mitigations

Risk	*Mitigation*
Lack of deployable software	Use a CI system to build deployable software at any time. Create a repeatable build process that uses all software assets from the version control repository.
Late discovery of defects	Run builds that include developer tests with every change, so you can discover defects earlier in the software lifecycle.
Lack of project visibility	Know the health of your software at all times by running builds regularly. When effectively applying the practice of CI, the project status is no longer in question.
Low-quality software	Run tests and inspections at every change so you can discover potential defects that may be introduced into the code base by learning the complexity, duplication, design, code coverage, and other factors.

Questions

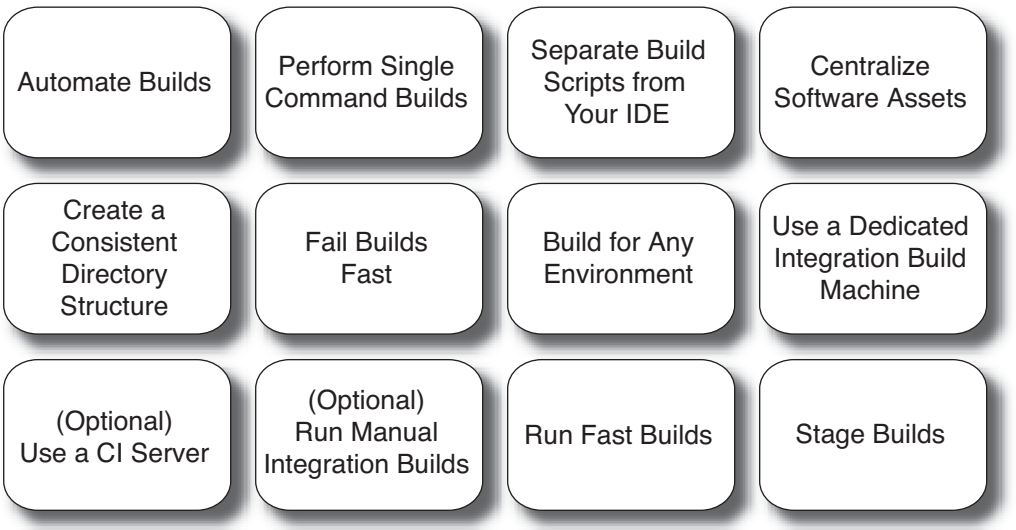
How many risks do you have on your project that CI can help mitigate? These questions should help you determine the risks on your project.

- When do you find the most defects on your project, in the beginning or in later parts of the lifecycle?
- How do you determine the quality on your software projects? Are you able to measure it?
- Which processes on your projects are manual? Have you determined which processes you can or should automate?

- Do you have all of the scripts to rebuild your database and data in your version control repository? Are you able to rebuild your database and test data during the build process?
- Are you able to perform regression testing whenever a change is made to the software? Are you able to run various types of regression tests, including functional, integration, load, and performance tests?
- Do you have the capability to determine which source code does not have a corresponding test? Are you using test coverage tools?
- What percentage of your software has duplicate code? Are you seeking to reduce this amount?
- At any point in time, how do you verify that the current source code adheres to the software architecture?
- How do you notify that the build or deployment is ready to test? Which communication mechanisms on your project are manual, and which should be automated?
- Are you able to view a current visual diagram of your software? How do you communicate the software architecture to a new developer on the project?

This page intentionally left blank

Chapter 4
Building Software at Every Change



The whole damn universe has to be taken apart, brick by brick, and reconstructed.

—Henry Miller, American writer and painter (1891–1980)

In the early 1900s, workers on the Ford assembly line manufactured cars by hand. A Model T took several days to create. Today, cars are infinitely more complex than the Model T, yet they now take a fraction of the time to assemble. Why is this? The answer is simple: *automation.* The automobile manufacturing industry has removed humans from repetitive tasks and replaced them with robots. So, too, can time-

consuming tasks within a software process be mechanized using automated builds. In fact, in both industries, the volume of demand has necessitated this advancement. If a worker's effort in her eight-hour day is tied up in eight hours of manual tasks, there is absolutely no time left for monitoring the process and the product, planning improvements, and so on.

Sometimes developers are like the cobbler who provides all his customers with shoes, but forgets shoes for his children: We create applications to automate processes for users, yet we don't automate our own processes for *developing* software. A study[1] conducted in 2003 indicated that approximately 27% of development teams run daily builds. As an industry, you could say that we are still using the old, manual automobile assembly line model.

People sometimes refer to the complex nature of software as an excuse for not automating portions of development. Yes, developing software *is* often complex, but there are many repetitive, error-prone activities that we can automate. The development of software may be complex, but the *delivery* of software must be a push-button affair.

The Integrate button (as seen in Figure 4-1) contains an "automated assembly line" that embodies many practices that compose the high-level practice of CI. An **automated build** represents the modern-day automated assembly line that uses "robots" to integrate software.

In this chapter, we discuss the benefits of using a CI server to perform an integration build whenever a change occurs. Not all builds are built the same, so we cover the types of builds you will typically execute and how to stage your builds. We also cover the aspects of choosing and using a separate integration build machine for CI. Automated CI is not the only viable approach to running an integration build; we also cover a technique to run manual integrations using a queued approach. Since obtaining build feedback quickly is so important, we finish the chapter with the bane of CI, long-running builds, along with common questions about CI we've heard over the years.

---

1. Cited in "Software Development Worldwide: The State of the Practice" (with Alan MacCormack, Chris Kemerer, and Bill Crandall), *IEEE Software,* November–December 2003, vol. 20, no. 6, pp. 28–34 (Invited). www.pitt.edu/~ckemerer/CK%20research%20papers/SwDevelopmentWorldwide_CusumanoMacCormackKemerer03.pdf.

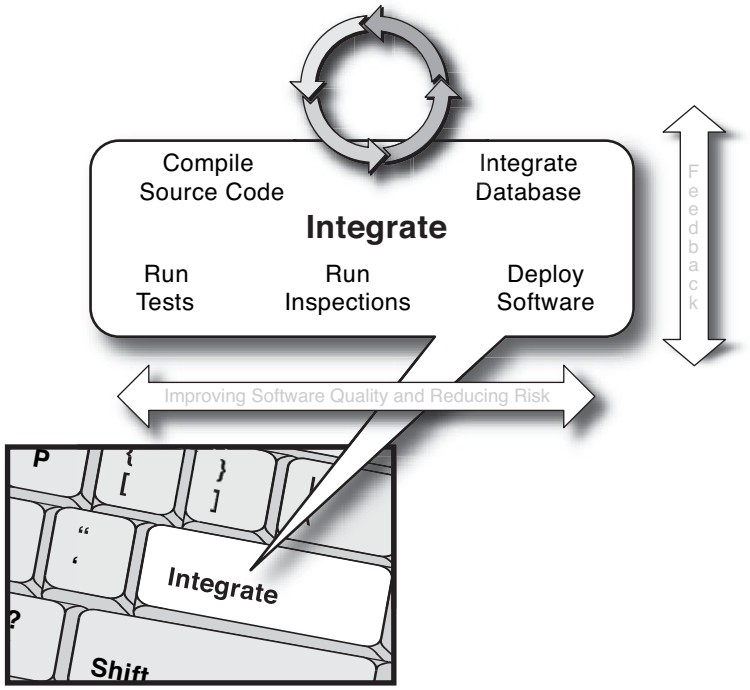


FIGURE 4-1 Building your software to improve software quality and reduce risk

Automate Builds

By writing automated build scripts, you reduce the number of manual, repetitive, and error-prone processes performed on a software project.

What is a **software build?** Is a build just compiling software components? Is a build compiling components *and* running automated tests? Is it a build only if inspections are included? A build can be any of these, yet the processes that you include in a build can more effectively reduce risks; however, the more processes added to a build, the slower the feedback. Therefore, you must determine which processes to include in an automated build. For example, in Chapter 2 we described the practice of running a *private build,* which consists of integrating changes from the team, and running a *full* build (which may include compile, test, inspections, etc.) on your workstation prior to committing code to the version control repository to prevent broken