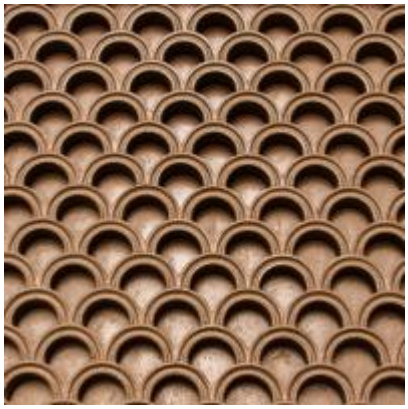
Achieve Repeatable Builds with Continuous Integration

By Steve Berczuk - April 9, 2020



Successful agile delivery requires a good execution process. Continuous integration is essential to provide the feedback needed to keep a team's code agile.

There are many parts to a successful CI process, and one crucial aspect is a repeatable build. There are two parts to maintaining a repeatable build: the idioms and practices to define it, and the feedback cycle to maintain it.

A repeatable build enables anyone (or any process, like the CI build) to build the code at any time and get the same result for a given version of the code. When you change the code, you want to be sure the changes in behavior are entirely because of changes you made, and not related to something unknown.

With repeatability, a developer can run a build locally (a "private system build," in software configuration management patterns language) before pushing a change to the repository, and if their build worked, so will the integration build that runs after the commit.

The first part of a repeatable build is having a configuration that specifies all required dependencies and their version, such as in a build tool like Gradle or Maven. Some tools even allow you to download the correct version of itself using something like the Gradle Wrapper (or its Maven equivalent). Once you have this set up, the private system build and the integration build should each work reliably to provide the feedback loop you need to detect issues.

Even when you fully specify artifact dependencies, external dependencies are unavoidable. In Maven and Gradle, you specify a list of repositories to get dependencies from, and these may be out of your control. Using reliable repositories such as Maven Central helps with stability, but even with that, changes can still break your build.

For example, in January, Sonotype required central repositories to use https. If you didn't prepare for this, builds may have failed because of your repository configurations. Also, from time to time dependencies move. The Maven Restlet Framework moved out of the Maven Central repository, and afterward, builds suddenly failed because of unresolved dependencies.

Using a repository manager or proxy repository can help address these repeatability issues by insulating you from changes to external repositories and simplifying the repository management section of your build configuration. Once a build works, the artifacts it needs will be cached by the repository manager. If you do find a problem, you can now fix it in one place.

But things can still drift. Frequent builds with continuous integration will give you the feedback to validate that your repository manager is helping you to maintain consistent builds. When you have modules that aren't being changed frequently, you lose this automatic feedback mechanism.

One way to avoid surprises when you revisit a stable piece of code is to exercise the build. Consider setting up an automated process that runs the build process periodically but perhaps doesn't update any stored artifacts, so you can catch build errors that external changes over time may introduce.

Build and build scripts are code, and the principles of good design and feedback apply to them, too.