Code Integration: When Moving Slowly Actually Has More Risk

By Steve Berczuk - May 12, 2020

The core ideas behind any agile process are demonstrating value through working software, and the ability to adapt to meet new requirements. Doing these things requires frequent integration so that there is a clear sense of what the state of the code is.

While there isn't one exact approach to code integration that will work for every team, some approaches lend themselves to agility more than others. But even agile teams often have long discussions about which code line and branching models to use, and they often end up with more complex strategies than necessary.

Many decisions about branching models are made in the name of managing risk, and teams sometimes pick models that make integration harder in the name of safety. But managing risk in this way can be counterproductive.

Teams may try to manage risk by moving slowly and placing barriers to change, such as various branches, review steps, testing stages, and manual approval processes. Moving slowly can seem safer, and sometimes it is, but only if the steps you use are guaranteed to be correct. These processes are often manual, and thus also error-prone, so rather than help you manage change, they simply give you an illusion of control.

Agile teams work best when they acknowledge that there is also risk in deferring change. Any validation process will have flaws. This means focusing on processes that improve the time from code change to delivery so that feedback can happen quickly, and when there is a problem, teams can fix it immediately.

Having branching models that keep work isolated means that work is integrated into the delivery code line less frequently. While this may be more efficient for those working on a feature in the short term, it can hurt the flow of the team in the long term.

Martin Fowler illustrates the impact of integration frequency on speed of change. Deferring integration can increase the risk of merge conflicts, which causes you to move more slowly as you spend more energy addressing those conflicts. Slow change can sometimes be more risky than you expect because of the costs of extra work needed to reconcile conflicts, as well as the technical debt that results from bypassing the normal process to fix critical errors.

Using agile branching patterns alone won't make your code more agile. For frequent integration to work well, you need to have confidence that your code line still works. Healthy code lines also require that thought be given to testing and deployment practices.

When Brad Appleton and I were working on capturing patterns for version management, first in the Streamed Lines paper and later in our book, *Software Configuration Management Patterns*, we were trying to capture patterns for how teams used their code-line management process to collaborate effectively. We were not thinking about agile software development per se, but we quickly realized that the patterns had agility in mind. One of the recurring themes in these patterns is that frequent integration is key to an effective development process, which is especially true for agile teams.