JOIN OVER 100,000 PROS! Join Log In | Member Benefits

STICKYMINDS

Agile Testing DevOps Security AI

Branching to Distraction

ARTICLE

By Steve Berczuk | July 10, 2011

SUMMARY

Branching can be an effective solution for managing change, enabling parallel development and improved productivity. But, working on a branch is a distraction and can decrease agility, productivity, and code robustness. Learn when the value of working on a branch outweighs the cost.

Branching is a source code management technique for enabling parallel development. When you branch, you create a code line that is essentially a copy of its parent. By keeping track of the ancestry of the new code line, you can identify what has changed since the branch point and apply changes from the branch to the parent code line and vice versa. A branch allows you to work on a variation of the code without immediately affecting—or being affected by—changes on the main code line. (Streamed Lines has a detailed description of other reasons for branching.)

Figure 1 illustrates some branching patterns. Assume that you are working on a main line, and you are scheduled to release version 1 in a week. At some point, you might decide that there are people on your team who can start working on a new feature for version 2 while the rest of the team finishes version 1. You can create a task branch, where a group of developers can work on the new code without interfering with the soon-to-be-released version 1. Once version 1 is ready to ship, you can create a release branch to deliver fixes for released software. In theory, the changes required to deliver the fix from a release branch will be smaller and more easily tested than if you were to attempt to deliver from the main development line. The team can address the problem to be fixed by committing changes to the branch without interfering with the work of the rest of team. You can then merge the task branch changes back into the main line so that everyone is working on one code stream again.

Without branching, it would be tricky for the team to work on two tasks at once. And, there are times when working on a branch can help you be more agile by enabling you to work around roadblocks and keep moving while other work is being completed. A branch

We use cookies to ensure you get the best experience on our website.

Accept

Read More

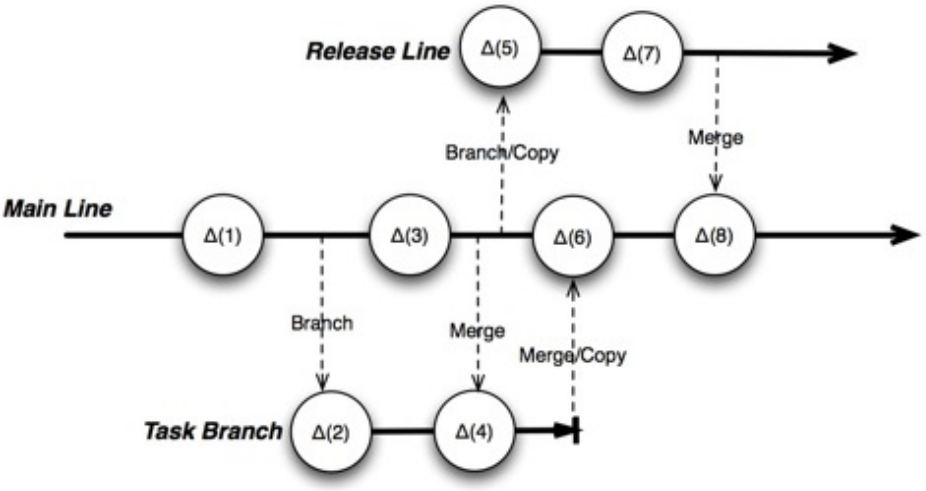

Figure 1. Common branching patterns

Branching makes it easy to defer concerns about the impact of a change on the main code line and to focus on the code variant you are working on. By working on a branch to deliver a fix, for example, you address the following concerns:

- The perception that it's too costly to QA the new code line before delivery and that making a small change to "stable" code will be easier to validate.
- The desire to limit the visibility of new (incomplete) features, since new work is being done on the main line.

Another example is using staged integration branches, where a change does not appear in the main line until it has passed through a variety of merges. In this case, you are using branching to make it less likely that someone will check out broken code from the main line.

While these approaches have merit on the surface, you also must think about the basic problem that you are solving and consider the costs of working on a branch.

The Distractions of Branching

When working on a branch, you are splitting your attention between multiple streams of work. There is a context switch to working on the "old" code from the branch, and often the changes you make on a branch (or equivalent functionality) need to be merged to the main code line. Doing this merge creates extra work over what you might do if you are able to simply change the main code line.

A branch can be a useful tool in some situations, allowing your team to respond to issues with agility even when your main code line is not agile enough to allow the change. A branch can help you be more agile by allowing you to focus on a task, or it can impair agility by causing a split in attention.

The paradox of branching is that while branching is meant to help you to focus, a branch can be a source of distraction to the team for reasons ranging from context shifting to confusion about where a change was made. To using branching effectively, you need to acknowledge the potential distractions a branch can cause and determine whether the

We use cookies to ensure you get the best experience on our website.

Example: The Release Line
Consider the situation where a customer reports an issue in the code between the time it was initially released and when the next release is scheduled to ship. Working on a release line branch can be an appealing solution, because branches provide isolation and a perception of simplicity and safety, since you are starting out with code you believe to be tested and release quality.

The belief that you are starting out with more stable code does not remove all of the perceived added cost over delivering the code on the main line for the following reasons:

- Working on a branch creates extra work.
- Working on a released code line usually does not save as much effort as you might have thought.

Regardless of the nature of the fix, you are splitting your attention from the current project work. You may be fixing a problem that has already been addressed on the active development line, and you are constrained to using versions of frameworks and tools that the released code was delivered with, even though using the newer versions would make work easier. Developers have to deal with the frustration and cognitive costs of working with a code base that is different than what they work with daily.

In many cases, you may want to migrate the work you did on the branch to the main development line by merging the code changes. The complexity of a merge can range from simple to impossible. Simple changes involving small text changes when the main line has changed little can be done quickly and easily. In other cases, merging a complex change to a heavily changed main line may not be possible without significant manual intervention (though there is some research about **tools that support refactoring**).

Since software systems are complex, all but the most trivial changes will require that you extensively test the software before you ship it. You might overestimate how much delivering from a branch saves you, but having good automated test coverage can make this easier. Good automated test coverage can also reduce the need to branch, as discussed below.

Creating a branch will add some infrastructure work (creating a new continuous build, for example). Working on a branch also delays integration with the main code line.
When you are working on a change that you will need to migrate to the main line, you have two choices:

- Integrate frequently between the branch and the main line as the work is being done.
- Wait until the work is done on the branch and attempt to integrate later.

Frequent integration has the advantage of identifying integration issues quickly, but the disadvantage of requiring the person doing the merge to shift context frequently. Merging after work is done simplifies the context-switching overhead, but a rapid rate of change on the main line can complicate the merge task.

This problem becomes more complex the longer the time between the branch point and the fix. And, regardless of which approach you choose, you've added complexity and risk to even the smallest change to the branch. If changes to a branch do not need to be

We use cookies to ensure you get the best experience on our website.

Using branching to provide stability and isolation is based in a development model that assumes fragile and inflexible code, and it is possible that your code is not as stable or flexible as you'd like it to be. In this case, branching can be a way to enable you to keep your code working until you can make it agile. As you establish development practices that help your code be more agile, you can use branches in a way that has less impact on your ability to deliver code.

Paradigm Shift: Enabling Less Branching
Avoiding the distractions of branching requires changing your approach to development. Branching can protect you in the short term when you have unstable code lines and long release cycles with little change between releases. But, as your team adopts more agile practices and is able to deliver reliable code more quickly, the release branch pattern becomes less and less necessary.

To make less branching possible, you need to work towards:

- *More frequent releases*—the shorter the time between releases, the more likely you can deliver a fix from the existing code line. And, in the rare case that you need to deliver a fix between releases, you can have shorter-lived branches.
- *Automated testing*—so that you can keep your code lines stable and reduce the cost of delivering from the main line (or a short-lived branch).
- *Continuous integration*—so that you detect problems quickly.
- *A DevOps mindset*—where you develop deployment and operations procedures early so that you can release as soon as the code is "ready."

There may well be times when working on a branch makes sense, especially as you are transitioning to practices to make your team more agile. Try to understand the costs of branching and whether working on branches distracts you from your business objectives. Shifting your development process will involve both technical and organizational change.

Branching as a Gateway to Agility
Even though branching has downsides when you have processes in place to enable frequent delivery, there are situations where branching can provide you the space you need to develop agile practices.

Keeping a legacy release on a branch while you refactor the main line to be more testable allows you to focus effort on making code better while at the same time minimizing the cost of changes to the delivered code. The costs of working on the branch could be more than made up by the reduced support costs for future releases.

Some of these changes are organizational, and organizational change can be difficult and slow. If you want to move towards a more agile code line at a team level, there are some steps that you can take:

- Branch only when, on net, the branch minimizes distraction and effort rather than increases it. Use the cost analysis to make a case for the organizational changes you need to establish a more frequent release model.
- Consider the need to branch as an indication that your code may not be as agile as you need it to be.
- Remember that tools that make the mechanics of branching easy can be helpful, but

We use cookies to ensure you get the best
experience on our website.

- When you do branch, keep branches short lived and integrate changes between branches frequently.
- Increase your automated test coverage so that you can improve your confidence in your ability to deliver code from the main line.
- Consider applying the patterns in ***Software Configuration Management Patterns*** that discusses how to develop using a minimalist branching model.

When to Branch
Branching is both a distraction and a way to limit distractions. Identifying the difference is tricky. In most cases, you are better off avoiding branching except when the branch is truly a divergent, parallel effort—for example, support for a legacy release that you will not migrate to the main development line. A more practical reason for a branch is when your code base is such that regression testing is risky. Here are some guidelines to minimize the cost of branching when you need to branch:

- Create release branches for delivering fixes, but work to minimize the time between releases. Focus on merging functionality (and tests) rather than code back to the main line.
- If you have a business need to support prior releases for a period of time (and customers who won't upgrade) using release branches is unavoidable from a technical perspective, but be sure that the business costs are clear to those negotiating contracts.
- Use task branches only for truly exploratory work that might be abandoned, or for the initial phases of an exploration. Try to make task branches short lived.
- When considering branching, be sure that you understand why you are branching. In the cases of released code, consider tagging the release point and branching only when you need to deliver a fix.
- When you do create a branch, make sure that you have tooling to make creating a branch workspace easy and quick.

Branching can be a powerful and useful tool when used appropriately. Remember that working on a branch means that you are deferring integration and pushing risk downstream, not avoiding it. Don't branch thoughtlessly. Instead, be sure to understand the cost of branching vs. the benefits, and your code and organization can become more agile more quickly.

Topics: agile, development lifecycles, process improvement, testing,

We use cookies to ensure you get the best experience on our website.

ABOUT THE AUTHOR

Steve Berczuk

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at blog.berczuk.com.

UPCOMING WEBINAR

LETS HANG!

FEATURED RESOURCES

AGILE

Implementing Agile eGuide

White Paper | StickyMinds

We use cookies to ensure you get the best experience on our website.

AGILE

Scrum Master Essentials eGuide

White Paper | StickyMinds

Log In or Register to post comments

USER COMMENTS

0 comments

YOU MAY ALSO LIKE

AGILE

From Silence to Self-Organization: What Social Interaction Games Achieve That AI Cannot

Dinesh Sharma | June 17, 2026

AGILE

Why Your Agile Team Is Failing Even When Everyone Knows Their Stuff

Ella Mitkin | March 2, 2026

AGILE

Solving the Problem vs Treating the Symptoms

Payson Hall | January 16, 2026

Join the crew for FREE! Access exclusive content,

We use cookies to ensure you get the best experience on our website.

See All Member Benefits

We use cookies to ensure you get the best experience on our website.