Agile Advocate: Establish Your Branching Strategy

And do it before development takes off.

Bill Drew
Thu, 02/01/2018 - 10:37

The Agile Advocate is a series of thoughts or musings on the agile movement and DevOps in particular. It was motivated by the seminal work of social philosopher Eric Hoffer, “The True Believer” (1951). It's not meant to be a manifesto, but simply a series of thoughts and reflections coming from over three decades of developing and delivering software and systems of a wide variety — to an even wider variety of users and customers. My aim is not to evangelize or convert, but to provoke and stimulate discussion.

Every software development effort should use a version control system. Whether
it’s a simple prototyping effort to prove out an idea or a large, mission critical systems development effort, they can all benefit from the capabilities of a VCS.

However, despite the critical nature of your VCS, it’s rarely used in strategic discussions in terms of software development and delivery. But if you’re expecting to meet your end users' expectations in terms of delivery and at a cadence that supports their needs, it’s imperative you establish your branching and merging strategy up front — before development takes off.

What Your VCS Does

The primary role of a VCS is to provide a persistent store for your source code and related files required to build your project along with a historical record of all changes to your code. In addition to this historical archive, any commercial grade VCS also supports the concepts of branching and merging.

The structure of the code base maintained by the VCS is analogous to a tree. At the initiation of a development project, a code repository is created with a default branch called the master or trunk branch. Going forward, we’ll use the term trunk to describe both.

Branching allows for the creation of a complete copy of the trunk or of another branch and is completely isolated from other branches. Changes made to code in one branch have no affect on other branches, including the trunk. This allows code in different branches to progress independently without interfering, depending upon or hindering the progress of other branches.

Merging is the act of applying changes from one branch into one or more different branches. Normally, this is done to apply bugs fixes identified and resolved in one branch to other branches. Or it can be used to apply new or refactored functionality that may be seen as useful for one or more branches.

The First Cut

There are many approaches you can take in establishing your branching and merging strategy. The branching strategy we will propose and outline here is derived from industry best practices, motivated by simplicity and proven out over the years in many projects and domains. It supports multiple, simultaneous
development streams and provides for a controlled, disciplined and structured deployment of release candidates into production and maintenance and support of those releases once deployed.

Figure 1 - Development Trunk

Figure 1 above shows the single, default trunk branch with development teams committing to it over time as they complete functionality. Development teams will always commit any new code and changes to the trunk as they build out the functionality that will comprise the next release. When the functionality contained in the trunk reaches the point where it provides all or nearly all of the expected functionality for the first release of the application, a decision is made by product owners, development leads and product management to create a branch of the code from the trunk branch as shown in step #1 of Figure 2. At step #1, the trunk and Release 1.0.x branch contain the same code and functionality.

The functionality in the Release 1.0.x branch represents a candidate for release into production — or a release candidate. It’s expected POs and further testing along the migration path into production will subject it to additional interrogation.
Assuming the release candidate passes through the phases of the deployment pipeline, it will be deployed to production as Release 1.0.0. After the release is deployed, it’s reasonable to expect that bugs will be uncovered and shortcomings in functionality will be discovered. These will be addressed by maintenance teams, which may be separate from the development teams as shown above, with bug fixes as shown in step #2 of Figure 2.

These modifications are committed to the release branch and tested in the same manner as any release. Assuming that it passes all PO reviews and testing, the new release candidate will migrate along the deployment pipeline and into production as Release 1.0.1. If these modifications or bug fixes are deemed useful or necessary for the next release, then they are also merged back into the trunk as shown in step #3 of Figure 2. This process of issue resolution and deployment continues for this long-lived release branch. Any issues after Release 1.0.1 are addressed, tested and deployed as Release 1.0.2 and relevant and required changes merged back into the trunk if required, as shown in steps #4 and #5 of Figure 2.
At this point in time, we now have a Release 1.0.x branch that has produced three releases and merged multiple bug fixes back to the code base in trunk. Note that any 1.0.x release issues can be addressed independently without affecting the progress towards the next release that is occurring in the trunk. Development teams have continued to add new code and functionality to the trunk for the next release.

**Multiple Streams**

As with the previous release, at some point, the functionality added to the trunk will be deemed sufficient enough for the next release and a Release 1.1.x branch will be cut as shown in step #1 of Figure 3. We now have two simultaneous release streams for Release 1.0.x and 1.1.x. Each of these streams is independent and allows for full support of their respective releases without affecting the other release stream.

Production support can continue for Release 1.0 while Release 1.1 is being tested and vetted throughout the deployment process. As with Release 1.0.x, as bugs and defects are discovered in production, they are addressed in the code to be included in subsequent releases as in #3 and #6 below.
Figure 3 - Multiple Release Branches

Note when bug fixes are addressed, the code is merged into the trunk initially and then into other branches — if those branches deem those changes are necessary for the release. This can be seen in #2 and then #5 below.

Release 1.1.x will eventually be deployed to production replacing Release 1.0.x.

Note once again that the development for future releases has not been impacted and developers move forward, adding functionality to the trunk for future releases. At no time is the development on the trunk or for that matter any branch halted by code freezes, subjected to toggle switches or other agile anti-patterns.
And each of these release branches is long lived and is maintained indefinitely in the VCS. They may be retired in the future once they been superseded in production by another release, but only in the sense that they are made read-only and essentially archived.

**Establish Your Strategy Early**

Your code base is one of the most, if not the most, important resource in your development effort. So, it makes sense to put the requisite effort up forth to develop the policies and procedures that will allow you to effectively manage your code base. But more than simply managing your code base, your strategy should be aligned with your user expectations in terms of a delivery cadence.

The strategy laid out here is simple and straightforward, and coupled with a high-functioning developer’s sandbox, supports a robust development environment. It should integrate easily with your continuous integration process to support a fine-grained feedback loop between POs and tester and development teams.

It provides for multiple, simultaneous release streams that support a high-production delivery cadence. It also supports the ability to quickly react to production issues in your current production environment through the same reliable, predictable, repeatable process all production deployments follow.

-- Sign up for our weekly newsletter to receive the latest analysis and insights on emerging federal technologies and IT modernization.

View printer friendly version
The Agile Advocate
agile
Non-feature
GCIO Focus