Crossroads News
A Monthly Publication for Software and CM Professionals

ABCs of a Branching and Merging Strategy

by Mario Moreira – November 2003

Branching is both simple and complex. For many, it is challenging to know where to begin. This article hopes to provide a starting point, by highlighting branching concepts, providing reasons for branching, and suggesting an approach to establish a branching and merging strategy.

When establishing a branching and merging strategy and corresponding model, it is important to involve project management since this strategy directly impacts application development. They should be part of the branching strategy decision process, understand the pros and cons to branching, and have knowledge of the effort involved in branching and not branching. This ensures project expectations are set from the beginning.

It is also important to note that the CM tool being used plays a key role in branching and merging. Some CM tools are better at branching and merging than others. It is important to select and use a CM tool that can support the strategy for application development.

Key Concepts

What is a 'branch'? A branch is an isolated instance of an existing development baseline. Within a branching model, the 'root' development baseline may be known as the 'trunk' and sometimes referred to as the 'main branch'. A new branch can be based on the trunk or another branch. When a new branch is created from the trunk or another branch, the trunk or branch is known as the 'parent' of the new branch.

The items within a branch (whether physical or virtual) are initially identical with the parent. But over time, the items in a branch will evolve and become different to the parent due to the modification, creation, or deletion of the items within the branch.

The reason why the terms 'branch' and 'trunk' are used is that they mirror the look of a tree when shown graphically. The figure below illustrates this:

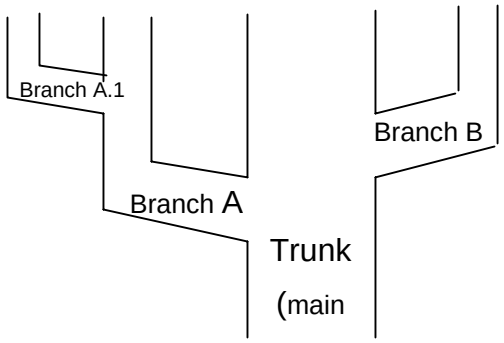

Figure 1

Each branch should have a unique name (e.g., 'Branch A.1') to differentiate it from other branches. The overall branch path should include the trunk name, any parent branch name(s), and the unique branch name (e.g., '/Trunk/Branch A/Branch A.1') for clear identification. This not only uniquely identifies the branch, but provides the ancestry of the branch back to the trunk. This information will be beneficial in understanding the relationship amongst branches particularly when items within a branch are to merge with items from another ancestor branch.

What is a 'merge'? A merge is an action that combines an item on one branch with the same named item that resides on another branch. In order for the merge to occur properly, the branches in which the items

live, must have a common ancestor. For example, foo.c resides on Branch A.1 and has been modified. The same foo.c file also resides in Branch A, and has also been modified. Because both versions reside on branches with the same ancestor (in this case, Branch A is the parent branch for Branch A.1), then merging may occur.

What is a 'label'? A label is a tag that affixes itself to specific versions of items within a branch. A branch by itself is generally dynamic meaning that items can change within it. Applying a label to the items within a branch provides a static identification of the baseline at a point in time. Typically, labels are placed on items within a branch that represents a milestone however small. This may include a successfully completed engineering integration build, a package for test, or release deliverables targeted for production.

Reasons for Branching

Should you Branch? In a nutshell, the primary reason to branch is if concurrent or parallel development must occur. This is described as the need to perform two or more isolated lines of development on the same baseline of code but for different purposes.

If there is a need for branching, a project release branch may be used to isolate the project work from the main line which may represent what is in production (therefore, you would not want to corrupt production with less than production-ready code). This is branching in its most simple form. However, there are more complex reasons for branching.

Reasons for parallel development at a project level may include (but not limited too):

- Working on two or more project releases (this may include major releases, minor release, customer specials, prototypes, platform conversions, etc.)
- Working on a project release and on bugfixes/patches for a previous release(s)

Reasons for parallel development within a project may include (but not limited too):

- Working with other sites (site specific branch)
- Working with several colleagues at the same site (shared branch)
- within a private workspaces (user specific private branch)

There is no limit to the number of branches that can exist. However, a branch and merge strategy should be designed prior to creating branches to first ensure that branching is needed and secondly to ensure that the branch structure under consideration will work for the project. Too many branches may make development too complex to follow and too few branches may constrain development.

Many project teams perform development from the trunk or main branch because they follow the serial development approach. However, some project teams end up performing some level of parallel development on the trunk or main branch and this is where problems may begin. The important point is to recognize the need for branching before and prepare a strategy for it. Conversely, many project managers make a decision to perform parallel development, but are not sufficiently informed of the complexity and effort. It is the job of the SCM professional to raise this awareness so they can understand the appropriate reasons for branching (or not branching).

Types of Branches

There can be various types of branches. As mentioned in the previous section, the branch structure starts with the trunk or main branch. Beyond this, the branch type should depend on the need. Some branch types may include (but not limited too):

- Project Branch – if the project is large, then this branch may be used to merge stable pieces of code. If the project is small, than this branch may be used as the integration branch for all

development changes. This branch type may be used when stability is needed and would be used in conjunction with an integration branch. It is typically backed by the trunk.

- Integration Branch – may be used as the active development line to integrate development changes. This line of development may not be stable depending on the amount of merging occurring into it. It is usually backed by a project branch.
- Shared branch – similar to an integration branch but used by a subset of developers working on a more volatile set of code such as performing prototyping so it does not impact others until it is sufficiently tested. It may be backed by an integration branch.
- Site Branch – similar to a shared branch, it may be used when there are other sites involved in development. This isolates their work but still allows for merging to the integration or project branch. It is usually backed by an integration or project branch.
- Private Branch – may be used to isolate individual developer's changes from each other. This may be backed by a site branch, shared branch, integration branch, or project branch. In fact some of the private branches may be backed by a shared branch and others may be backed by an integration branch.
- BugFix/Patch Branch – a branch that is used to perform bugfixes or patches to an existing release. Any changes in this branch should not only be merged 'in' to the trunk (presumably for production), but they should be merged 'out' to any new project development (via the project branch, integration branch, site branch, or private branch) so that the fix is included in the new release so regression in functionality and stability does not occur.

Below is an example of a branching and merging model which uses some of the branch types listed above.

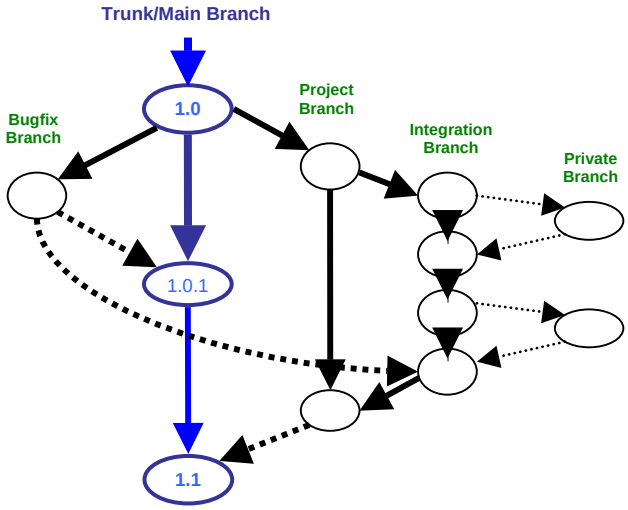

Figure 2

Establishing a Branching and Merging Strategy

When considering a branching and merging strategy, you must first understand that the application branching requirements will evolve and what is adequate today, will not necessarily meet the branching needs in the future. Therefore, it is important to think about the application's short-term and long-term branching needs when defining a branching strategy. This involves considering three aspects: complexity over time, effort, and risk to stability. Once these aspects are considered, a branching and merging model may be created that can support the application.

Complexity over Time

How complex is the application development and how complex might it become in the future? Some key components to complexity are (and not limited too):

- The number of users involved.
- The type of users that may be involved (developers, testers, build/release engineers, etc.)
- The amount of parallel development that may occur (# of releases occurring in parallel, bugfixes occurring on past releases, whether other sites are involved, and how much prototyping is occurring).

With the complexity in mind, consider how the complexity will change over time. For example:

- Within the first 6 months, "project release 1" may include only 5 developers, no parallel development, and only 1 site involved in development.
- After 1 year, "project release 2" may include 15 developers with parallel development to manage Release 2, bugfixes for Release 1, and a prototype for Release 3, and includes 2 sites involved in development.

Clearly, there will be a need for a more advanced strategy needed for the second scenario. This is why capturing complexity over time is important. The complexity will drive the branching needed and thinking ahead will ensure what is in place today can be easily extended to support the application needs tomorrow.

Effort

As parallel development is introduced, the level of branching complexity increases. As complexity increases, the amount of effort to manage change within a more complex branching environment increases. This should be an expectation that project management understands and plans. If they do not, ensure you advise them of this fact.

This is the trade-off of going to a parallel development model. While it may enable faster-time-to market, it may also increase the amount of effort and coordination in certain stages of the project and in particular adds merging and testing tasks to the project plan. Keep in mind that merging typically requires retesting to ensure that the code merged is tested appropriately, particularly where logical lines of conflict have to be reconciled.

Another aspect to considering effort is the project management strategy as it relates to the delegation of development work. If specific chunks of work are allocated to specific groups, then less merging and testing is likely to occur. However if the project manager is indiscriminate in the way the work is being dolled out then a significant amount of merging may occur increasing the amount of effort and quite possibly the project schedule. Even if no branching is used this latter scenarios (e.g., indiscriminately allocating work) will significantly increase the amount of testing that will be needed, therefore impacting the project schedule.

If there is a need to work on several lines of code at the same time, then 1 line of code may force a level of serial development and mostly likely cause significant testing, therefore impacting the release schedule. This may also cause many developers to work outside of the CM system in order to isolate themselves from others. On the other hand, if there is little need for parallel development, then too many branches may be confusing and causes more effort in merging than is needed.

The balance is to consider the level of complexity and identify several branching options to support the complexity. Then a level of effort can be considered per option to determine what may be acceptable.

Risk to Stability

As the complexity (as described above) of application development increases, so does the risk of negatively impacting stability on the project if it is not managed effectively. At this point, you understand the complexity of the application development in the short-term and long-term and you understand the effort as they relate to the branching options. Now review the branching options and determine how much risk to stability you are willing to accept. Risk to stability may have a direct impact on the project schedule it is important to involve project management in this decision.

A low risk tolerance to impacting project stability suggests that an integration branch should exist to support a stable project branch. While all changes are placed into the integration branch, only those that have passed certain milestone builds and tests should go into the project branch. Also, low risk suggests that if you are working with other sites, then separate site branches should be considered so that each site can be isolated from the local site. But what comes with a low risk tolerance is a potentially higher level of effort.

A high risk tolerance to impacting project stability does not mean you have to abandon branching, it just means that you will accept the risk associated with having more people work on less branches. For example, a high risk tolerance may allow remote sites to merge directly to the integration branch or even the project branch. Also, this scenario may have developer's private workspaces back directly to the project branch.

The End of the Beginning

Overall, the question is, do you want to control the changes or do you want the changes to control you. By constraining yourself to 1 line of code (working off of the main branch or trunk) or creating too many lines of code can significantly impact the amount of effort involved in branching and merging and the amount of testing that is needed to manage changes. Finding the balance is the key.

To summarize the approach specified in this article for preparing a branching and merging strategy, consider the following steps:

- Understand the terminology. Either borrow terminology from existing materials or create a consistent branching and merging terminology for your organization, application, or project.
- Understand the reasons for branching. Ensure they are legitimate and can be explained to others.
- Determine the complexity of the application development as it relates to branching and merging to help determine the types of branches that may be needed and when merging should occur. Establish several branching options.
- Assess the levels of effort for the potential branching options.
- Decide the level of risk you are willing to accept as it relates to the stability of the various branches you are using.
- Prepare a branching model (similar to the one in **Figure 2**) so that those you support can visually understand the branching and merging strategy and where they are working. Walk-through a scenario using this model.

With this information, a branching model may be designed with the type of branches needed and how they are positioned. This provides project personnel with an understanding of the branch structure and the merge tasks (and associated testing). Having a long-term branching strategy in place can help you manage change now and into the future.

References

- "Software Configuration Management Patterns: Effective Teamwork, Practical Integration" by Stephen P. Berczuk with Brad Appleton, 2003, Addison Wesley.
- "Software Release Methodology" by Michael E. Bays, 1999 Prentice Hall PTR.

Mario Moreira is a contributing editor for Crossroads News and Director/Architect of Technology for Fidelity Investments Systems Company and has worked in the SCM field since 1986. He has experience with numerous SCM technologies and processes and has implemented SCM on over 75 applications/products that include establishing global SCM infrastructures. He has an MA in Mass Communication with an emphasis on communication technologies. Mario also brings years of Project Management, Software Quality Assurance, Requirement Management, facilitation, and team building skills and experience.

You may reach Mr. Moreira by email at Mario.Moreira@cmcrossroads.com