A Branching & Merging Primer

Introduction

For some reason "Software Configuration Management" is still a poorly understood concept, perhaps because it is largely invisible, perceived to be restrictive or perhaps because it's just boring.

I am not sure why, but the fact is that enormous amounts of development inefficiencies (and other down-stream activities such as operations) are caused by poor implementations of software configuration management practices.

Software configuration management in its entirety is such a broad topic[1] that I decided to focus on one specific aspect in this paper – branching and merging. The topic is so fundamental, yet not well understood.

This paper explains the concept – not the mechanics of branching and merging, this paper is ***tool agnostic.***

Why does it matter?

Software Configuration Management practices provides *the foundation* for all members of the development team to collaborate effectively. No one doubts the importance of having a solid foundation to support a high-rise building, but few seem to grasp that Software Configuration Management plays a very similar role in software development.

I can't remember how often I discussed testing or deployment inefficiencies with customers, only to discover the root cause was poor software configuration management practices. Testing wrong configurations, using wrong test environments, deploying faulty build, etc. seem to be the norm rather than the exception.

To summarise, the benefits of good Software Configuration Management practices are:

- it safe guards your intellectual property – the software assets!
- it helps improve communication among team members

[1] Software Configuration Management covers a lot more than what is discussed in this paper, here is a definition what it entails: *The process of identifying and defining the configuration items in a software system, controlling the release, versioning and change of these items though out the software system life cycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items. (Source: Infosys)*

- it provides a way to establish clear responsibilities and accountabilities
- it provides traceability and reproducibility
- it facilitates reusability of software assets
- it provides consistency, reliability and integrity of software assets

The "Risk versus Productivity Trade off" dilemma

Branching and merging is really a *trade-off* between *risk* and *productivity*. You are trading off the safety of working in isolation from other team members, for the additional effort required to merge software assets at some point in the future.

Using branches provides better *isolation* and *control* of individual software assets and *increases productivity* because teams or individuals can work in parallel. But it also implies an *increase of merge activities and therefore risk* because branches have to be reassembled into a whole at some point in time.

Unfortunately there is no such thing as a free lunch!

Branches

When deciding on a software configuration management strategy, it is important to come to a decision what a branch represents. Branches are often closely aligned with system architectures, organisational units and/or Work Breakdown Structures (WBS) – see figure 1.

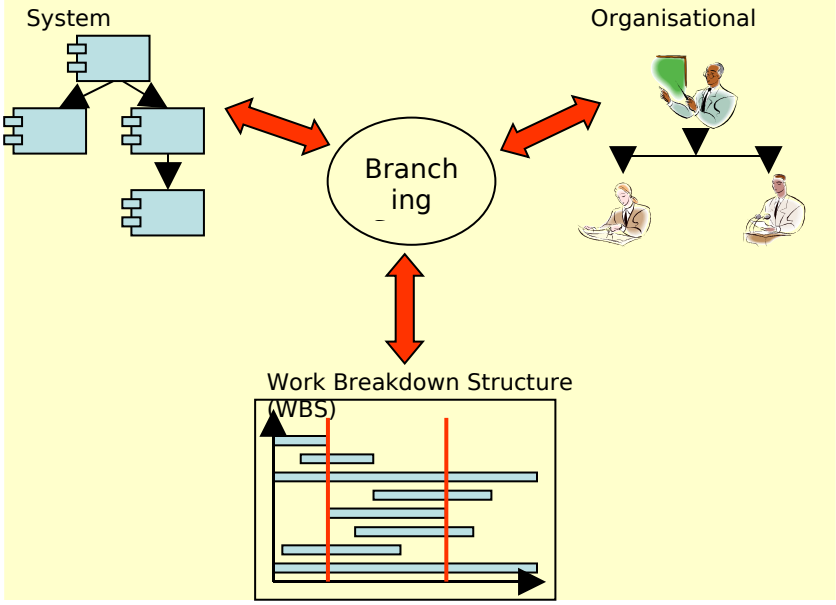

Figure 1

The following table lists a few examples of what a branch may represent:

Branch Type	Aligns with
A change request (or enhancement)	Work breakdown structure (WBS)
A task (or an activity)	Work breakdown structure (WBS)
A line of development for a particular (sub)project	Work breakdown structure (WBS)
A specific integration effort for a (sub)project	Work breakdown structure (WBS)
A particular release of a product	Work breakdown structure (WBS)
A maintenance effort for a specific (sub)project	Work breakdown structure (WBS)
A single product feature or function	System Architecture
A system, a subsystem or a system component	System Architecture
A entire project team	Organisational Structure
A project team member	Organisational Structure

As discussed before, deciding on the right branching strategy is a balancing act of trading off productivity versus risk[2]. One way to validate a chosen strategy is to consider a scenario of *change*.

For example if you decide aligning branches with the system architecture (e.g. a branch represents a system component) and you anticipate significant architectural changes, you may need to restructure your branches, associated processes and policies, etc. – perhaps you should choose a better strategy!

Choosing an inadequate branching strategy can result in process overheads, lengthy integration and release cycles and often ends up being a frustrating experience for the entire team.

[2] This paper is tool agnostic, but obviously it is also important to consider the capabilities of your tool when deciding on how to implement a software configuration management strategy.

Common Branching Strategies

This part of the document outlines common branching strategies used in the industry. The intention is *not* to provide specific guidance, but to stimulate ideas and demonstrate the importance of carefully planning the branching strategy.

Branch-per-Release

One of the most common branching strategies is to align branches with *product releases* (figure 2). A branch holds all the software development assets for a single release. Occasionally, updates need to be merged from one release to another, but they usually never merge. Branches will be discontinued when a release is retired.

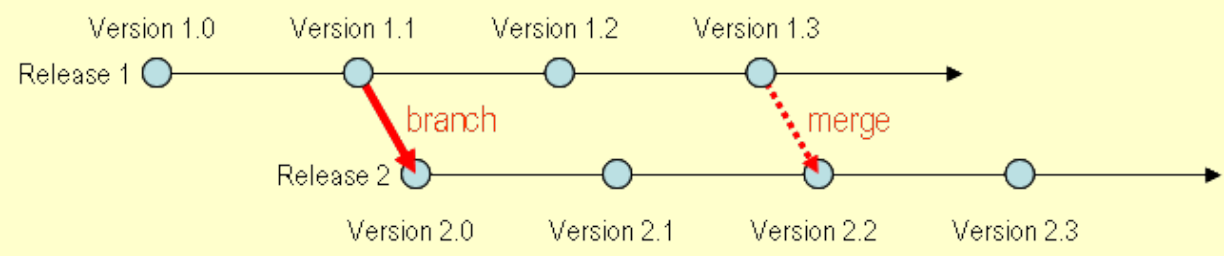

Figure 2

Code-Promotion-Branches

Another very common approach is to use branches to represent software asset *promotion levels* (figure 3). A particular development version is branched of into a "Test" branch, where all the integration and system testing is performed. Once the testing effort is completed, the software development assets are branched into the "Production" branch and ultimately deployed into production.

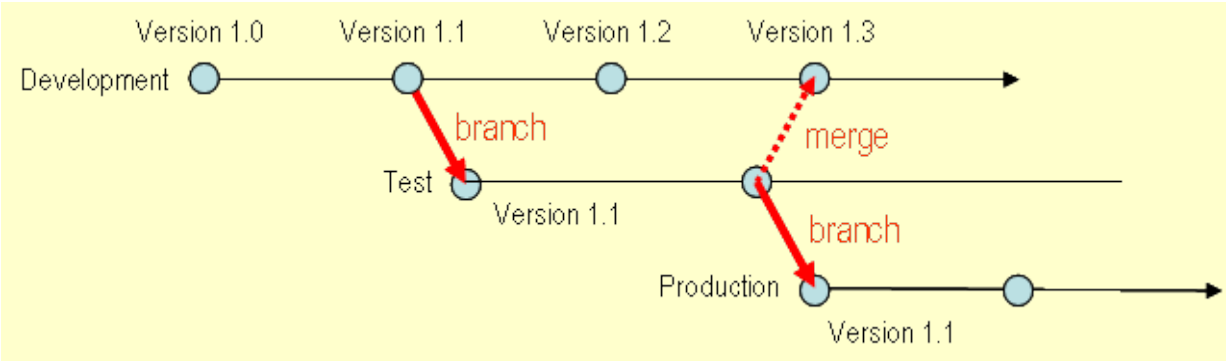

Figure 3

During the testing effort, the development code base is updated (I assume there will be bugs…). This requires the changes to be merged back into the Development branch when the tested code is promoted into production.

Branch-per-Task

To avoid overlapping tasks (or activities) and a loss in productivity, you can isolated them on a separate branch (figure 4). Keep in mind that these are *short-term* branches that should be merged as soon as the task is completed, for otherwise the merging effort required may exceed the productivity benefits of creating them in the first place.

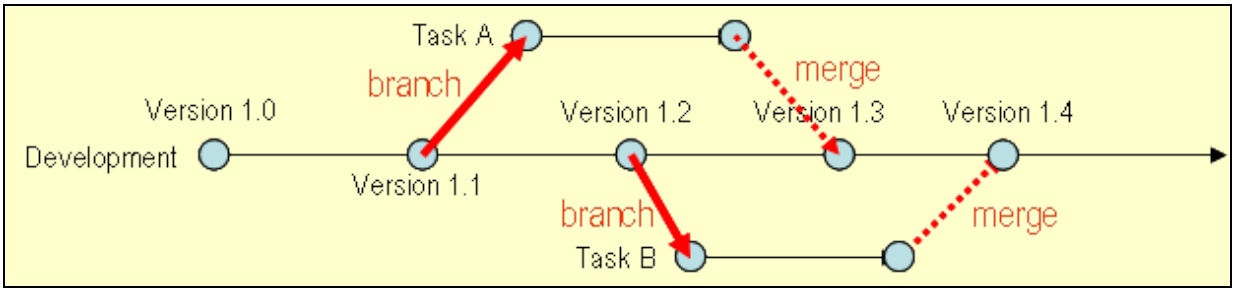

Figure 4

Branch-per-Component

In this example, the branching strategy is aligned with the system architecture (figure 5). Individual components (or subsystems) are branched off and the individual teams developing a component decide when to merge it back into the development line which essentially serves as the integration branch.

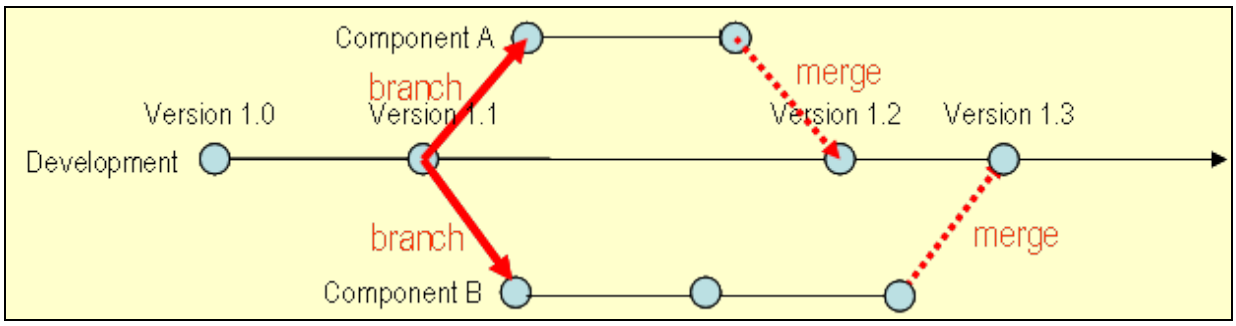

Figure 5

This strategy can work very well if a system architecture is in place and the individual components have well defined interfaces. The fact that components are developed on branches allows for more fine-grained control over software development assets.

Branch-per-Technology

Another branching strategy aligned with the system architecture is shown in figure 6. In this case the branches are aligned to technology platforms. Common code is managed on a separate branch.

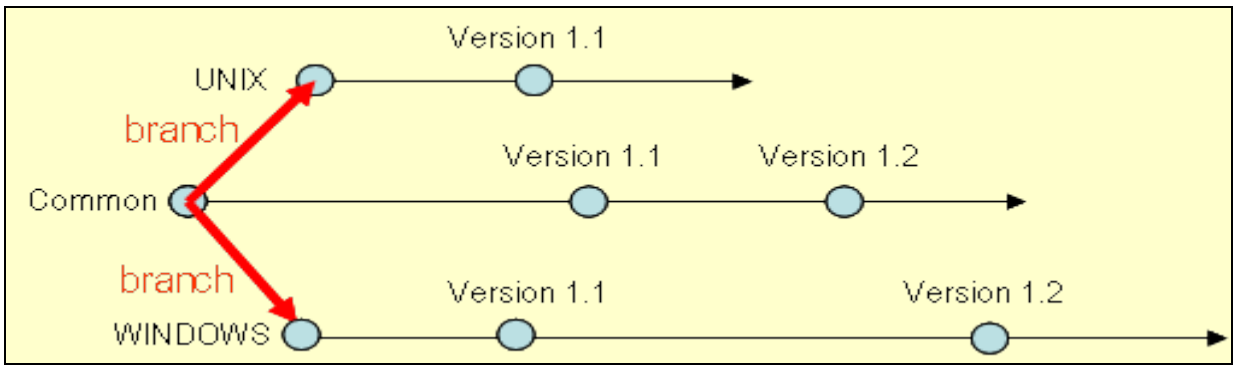

Figure 6

Due to the unique nature of the software development assets managed on the branches, they probably never merge.

There are countless other options how to structure branches, but I hope it is obvious by now that planning the implementation of a branching strategy is worthwhile and can return significant benefits.

Branching & Merging Anti-Patterns

You know you are on the *wrong track* if you experience on or more of the following symptoms in your development environment:

- *"Merge-Paranoia"*
  - a situation where merging is avoided at all cost, usually due to a fear of the consequences.

- *"Merge-Mania"*
  - a situation where the team spends an inappropriate time on merging software assets rather than developing them.

- *"Big-Bang-Merge"*
  - a situation where merging has been deferred to the very end of the development effort and an attempt is made to merge all branches simultaneously.

- *"Never Ending-Merge"*
  - a situation where the merge activity never seems to end, there is always more to merge.

- *"Wrong-Way-Merge"*
  - a situation where a software asset version is merged with a *previous* version.

- *"Branch-Mania"*
  - a situation where branches are created often and for no apparent reason.

- *"Cascading-Branches"*
  - a situation where branches are never merged back to the mainline.

- *"Mysterious-Branches"*
  - a situation where the purpose of a branch is unknown.

- *"Temporary-Branches"*
  - a situation where the purpose of a branch keeps changing and effectively serves as a permanent "temporary workspace".

- *"Volatile-Branches"*
  - a situation where a branch with "unstable" software assets is *shared* by other branches or *merged* into another branch[3].

- *"Development-Freeze"*
  - a situation where all development activities are stopped during branching, merging and building new baselines.

- *"Berlin-Wall"*
  - A situation where branches are used to divide the development team members, rather than divide the work they are performing.

References:

- Here you can find anything you ever wanted to know about software configuration management: http://www.cmcrossroads.com/

- Read this book by Brian White for a more detailed discussion about the topic. Yes it is ClearCase specific, but he was also instrumental in shaping the Microsoft SCM solution: http://www.amazon.com/gp/product/0201604787/102-3486560-6981702?v=glance&n=283155

Sydney, 28/04/2006

Chris Birmele
Developer Tools Technical Specialist

Microsoft Pty Ltd
Australia / New Zealand
cbirmele@microsoft.com

[3] *Branches are volatile* most of the time while they exist as independent branches, that is the point of having them. The difference is that they should *not* be shared or merged while they are in an unstable state.