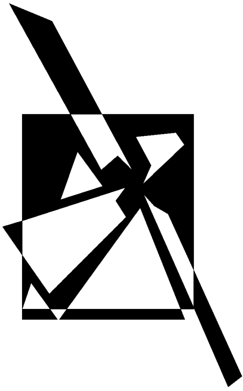


Taking the Complexity Out of Release Management

NEUMA

Copyright Information

Taking the Complexity Out of Release Management

CM+ is a trademark of Neuma Technology Inc.

Neuma Technology provides this document "as is" without warranty of any kind, either expressed or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. Neuma reserves the right to make changes to the product described in this document at any time and without prior notice.

Taking the Complexity Out of Release Management

by: Neuma Technology Inc.
#51 - 5450 Canotek Rd.
Ottawa, ON K1J 9G3
Canada

Tel: (613) 749-9450

http://www.neuma.com
email: sales@neuma.com

©2004 Neuma Technology Inc. All rights reserved.

Printed in Canada

DEC VAX, DEC VMS, DEC ALPHA are trademarks of Compaq Computer Corporation. Intel is a registered trademark of Intel Corporation. Microsoft Windows, Microsoft Windows XP, Microsoft Windows 2000, Microsoft Windows NT, Microsoft Windows 98, Microsoft Windows 95 and MS-DOS are trademarks of Microsoft Corporation. Sun/Solaris is a trademark of Sun Microsystems, Inc. UNIX is a registered trademark, licensed exclusively through X/Open Company, Ltd. UNIX and XWindow System are registered trademarks of X/Open Company Ltd. PostScript is a registered trademark of Adobe Systems Incorporated.

About Neuma Technology Inc.

Neuma Technology Inc. provides an integrated tool suite that helps manage the automation of the software development lifecycle. Customers can rely on Neuma's wealth of experience and understanding of the complexities of software lifecycle management to assist them in delivering quality products to their markets.

The company's flagship product, CM+ is a high-performance Software Configuration Management System which provides an automated environment for the management of quality software projects. Based on a process-oriented database and workflow technology, CM+ provides reliable configuration management capabilities, within a tightly integrated set of applications. Unique approaches to managing software releases and change packages, and a fully scaleable suite of applications differentiates CM+.

The integrated applications of CM+ include:

- Version Control
- Change Control
- Configuration Management
- Build and Release Support
- Problem Tracking
- Activity Tracking
- Requirements Management
- Document Management

Contacting Neuma Technology Inc.

For additional information, e-mail Neuma at *support@neuma.com* or visit our Web site at *www.neuma.com*.

Abstract

This article was originally published in the August 2004 issue of *CM Crossroads Journal* at *www.cmcrossroads.com/cmjournal*. This article outlines a clear strategy for Release Management which will reduce complexity, minimize branching and ensure accurate identification and descriptions of your releases.

© Neuma Technology Inc. All rights reserved.

Taking the Complexity out of Release Management

CM is complex enough without having to worry about managing releases! Release Management, however, isn't just part of CM, it should be driving your CM solution.

Release management deals with defining, using and managing the set of deliverables (the Build), for all of your customers. This includes the creation of records to subsequently identify release contents, the creation of variant builds, patch releases, incremental releases, and the support of parallel streams of releases (older product releases, current release(s) and future releases). It also deals with the ability to know what's in a release and to compare one release (e.g. one being sent to a customer) to another (e.g. the one the customer currently has so that the customer is well aware of the changes being made to his environment and how they match up against his requirements). In an end-to-end product management environment, release management spans the entire spectrum from requirements management through to product retirement.

In this article we will focus on aspects of a Release Management strategy which reduce complexity, minimize branching and ensure accurate identification and descriptions of your releases. We'll explore the need to:

- Establish a Product Road Map and use it to minimize branching
- Manage the Superset in your product component tree, but Build and Deliver Subsets
- Make Baselines meaningful milestones and use Build Increments between Baselines
- Establish Clear Consistent Release, Baseline and Build Identification
- Select CM tools that make it easy to compare, report on and browse differences between Builds and Baselines

Establish a Product Road Map

It's too easy to use ad hoc Branch Labelling or Directory Copying to define and freeze releases. As support issues arise and as you realize that your verification failed to uncover some significant bugs, the multiplication of branches and re-releases sets in. What you really want is an overall game plan. Your product needs a release strategy that is driven by market and support requirements, not by your tool, team or process capabilities - the latter should fall out of your requirements. Your design architecture will play a significant role in how quickly you can react to these requirements, but you need a Product Road Map up front, one that is simple, but flexible.

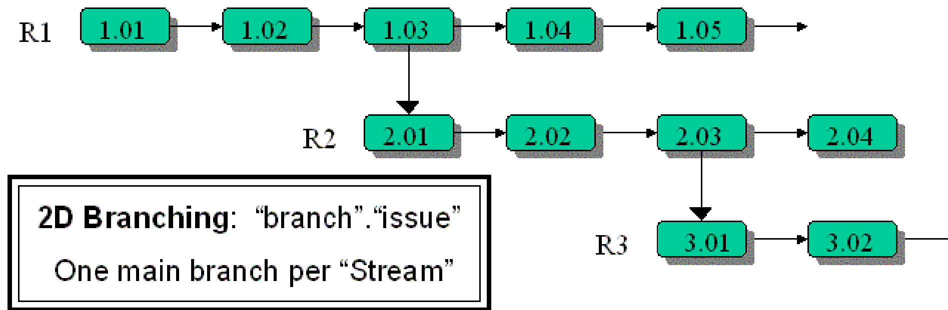
Many (most?) shops define a "main" branch and develop out of the branch, merging back in, and perhaps branching again to define a release or to support a release. It seems like a simple strategy because there's a single main branch. And if release dates and criteria could be cast in stone and customer variant requirements minimized, this might be a successful model. But in the real world, the release process is a long parallel-stream process. The single main branch model is very complex precisely because it does not map on to the real world requirements where development teams need to be working on a new release even before the previous release has made it out of QA.

So let's start with a different framework. Look at the Product Road Map - jump 5 years ahead if you need to. Release 3 is in the field. Release 4 is in verification, almost ready for beta, and work on release 5 has started. Three parallel release streams, all needing support - perhaps more in the future. The support team may be trying to move users on to the latest release, but it knows that it will have to maintain parallel streams. And it doesn't

© Neuma Technology Inc. All rights reserved.

want the support process for one release stream to change just because a new release stream has started.

Rather than using a main-branch centric process, try the following. Look at the Product Road Map. What does it say about how you need to roll out and support releases of your product. The roads can actually run in parallel. They don't have to run end to end (i.e. main branch) with a sudden left turn every time you need to define a support point.



Minimize Branching

Start with your current release branch, say the R3 stream. When its time to start working on R4, start working on it, but don't "branch the world" - create an R4 branch (if none yet exists) for each component/file when the first non-R3 changes go in. Continue to make changes to R3 in the R3 stream. The R3 stream should live forever on its own. It doesn't have to merge back into some artificial "main" trunk. The process for supporting it doesn't need to change when R4 comes out - it just slows down as things stabilize.

Changes to R3 required for R4 can be merged into R4. Or even better, choose tools that permit the R3 changes to be automatically picked up by R4 until you explicitly branch the associated file(s) into R4, in which case a merge may be required. Most work is going on in R4 and perhaps work on some major areas of R5 are starting. Your tools should be able to tell you when a change to R3 or R4 may need to be applied to other release streams.

So what we have is a 2-Dimensional (2D) release branching strategy. Instead of the traditional "main" branch, there are Release Streams. There's really no need to hold up R4 while R3 stabilizes. You may wish to focus initial R4 work in more stable areas to reduce the need to merge R3 changes into R4. But the 2D model allows you to have true parallel development, and with the right tools you'll have minimal branching. If your tool, team or process forces you to branch or re-label all of the components in order to start R4 development, you need to make some changes.

Are you wondering if your project might be too complex for a 2D model? Well I created my first real CM tool for a 25-million line telecom product in the '70s and '80s, with various product variants and thousands of developers. Not only was the 2D model sufficient, it was necessary!

As a side note, if you really want to minimize branching and merging, make sure your CM tools let you check in your changes before the build manager is ready for them. If developers have to artificially hold back on submitting finished changes, for whatever reason, the requirement for parallel checkouts (and in most tools additional parallel

© Neuma Technology Inc. All rights reserved.

branching and merging) will grow dramatically. A good CM tool will eliminate the need for most parallel checkouts, including branching, labeling and merging, and will permit your components/files to follow a your product road map. Investigate the change promotion model carefully to ensure that software is checked in one or two stages before it may promoted into a build.

Manage the Superset - Build and Deliver Subsets

With a 2D release framework which is easily mapped to the Product Road Map, consider variants, customizations, etc. Do not get into the business of managing multiple baselines for these. If you do, you'll multiply your work and decrease quality. Be assured that when you finally release R4, it will not be finally. Design and CM need to work together here. Start with a strategy that says you're going to manage the Superset of your product component tree, but build and deliver Subsets. Your tree is going to branch for each release stream, and it should contain all of the files/components for that stream (even though they don't have to branch yet). If you have independently developed sub-products such as EPROMs which encode the latest version of an IEEE standard then you may want a separate baseline for it (if it's on a different release schedule).

Tag components, subsystems, etc. which are not common to every Build/Delivery. What you want to achieve is the ability to define your variants as a set of common components plus a set of components containing one or more of the tags required for that variant. For example, your common components might be augmented with English and Professional tags for your default build variant. Another variant will have English, German and Enterprise tags. Work with your Design team to let them know that this is how you want to do your builds. An even better design would have your variants specified at run-time, minimizing the number of test builds you'll require. It's actually not hard to design to these criteria and it will cut down the number of builds you're doing, along with the associated testing and test resource logistics.

Your source tree browser (in your CM tool) should be able to browse the Superset of all components. If its a good browser, you'll be able to modify the view just by specifying the variant tags to the browser. So instead of having R4EnglishProf R4EnglishEnterprise R4GermanProf etc. branches and baselines, you'll have a single R4 stream with evolving baselines, and tag sets to identify the variant builds that you need. If your variants are all run-time configurable, you may not even need any tags - just a single build per release. This is where working with the Design team really pays off - to minimize complexities, not just for the build team but for the developers who need to do their own builds as well.

Another side note - having a single build for multiple variants won't necessarily reduce the amount of testing, but it will eliminate the logistics for your test bed (i.e. which build components are loaded etc.) - you just reconfigure what you've got, hopefully using a variant configuration file which doesn't have to change from release to release.

Make Baselines Meaningful - Define Build Increments in Terms of Changes to the Baseline

The release is delivered from development. You've defined and frozen a baseline. Patches will occur. There may be variant builds. Perhaps customized upgrades are required. There may be a seemingly endless train of builds for a release stream. But rather than defining new baselines every other day, define Build Increments on top of your baseline.

© Neuma Technology Inc. All rights reserved.

A series of 40 baselines makes it difficult to relate to any of them. If you instead have 5 or 10 baselines in a release stream, with several build increments in between, each baseline becomes a reference point for your whole team.

Build Increments are defined in terms of a baseline plus a set of changes. If you have change-based CM system you'll readily be able to identify the differences between each increment as a set of a few to a few dozen changes. If not, you're likely to see dozens to hundreds of changed files instead - do yourself a favour and move to a change-based solution. Comparing two Build Increments built on the same baseline is generally an easy task, and a meaningful one, resulting in a list of changes (and hopefully their descriptions).

It's difficult, at best, to manage releases in terms of changed files. A change-based system will allow your Change Review Board to review and approve requests for a particular product stream and they will be able to clearly identify which of those changes are in each successive build. The promotion model will work on changes, build comparisons will be done in terms of changes, etc.

Establish Clear Consistent Release, Baseline and Build Identification

Assign regular identifiers (Build Ids) to the Build Increments and use them to track your testing and problem reports. Put the Build Ids in the customer executable (along with the tags used to define the variant subset) so that it is easy to go from a customer site back to the specific build information. This also makes it easy to track your customer sites in the delivery portion of your CM solution.

For release identifiers, I prefer short names - you're going to want to combine them with other attributes and they'll take up valuable real estate on your reports and browsing tools. For example, R1, R2, R3 is sufficient. "Release One" may be more readable on its own, but its certain to shift other valuable info off the screen or page. Note that the release identifier is actually identifying the release stream. You will likely release more than one build from this stream to your customer base.

Make sure you have a CM tool that can help you with release identification. It should manage release identifiers, baseline identification, build identification (superset and variant), and it should do so in a regular, yet flexible manner.

Use your baseline numbering to identify major feature or quality differences. Tell the customer he has build R4 baseline 6 increment 12 [R4-6.12] and will be moving to R4-8.3. Give a major feature/fix summary of R4-7 and R4-8 to the customer ahead of time - let them know what they're getting. Then fine tune your process so that you move significant changes to the next significant baseline rather than upsetting the user by delivering a major change in the increment from R4-8.3 to R4-8.4.

It is also important to understand the differences between marketing's product identification and the development and support team's identification. Likely, the release stream identifiers (e.g. R3, R4 etc.) will relate directly to marketing, especially in an end-to-end tool. But build and baseline identification will likely be invisible, except to the customer who wants to understand what he has.

© Neuma Technology Inc. All rights reserved.

Select CM tools that make it easy to compare, report on and browse differences between Builds and Baselines

When the customer is on the phone, you want to be able to tell him or her when the problem was introduced and when the fix will be delivered. When you deliver the fix, you want to be able to compare the customer's old Build Id with the new one and send the customer a document that details the problems fixed, the features added, perhaps even the data environment files differences.

It doesn't really do a lot of good to say you have build 3421 and the problem fix is in build 3487. The customer will be scared off by the other 65 increments in the build numbering. You need to tell the customer what you're delivering. This can be time consuming and error prone if your tool doesn't support differencing at the Build Increment level. And we are not just talking about code differencing. With a good CM tool with integrated Customer Tracking, you can automatically generate customer-specific reports every time you deliver to the customer - without the need for a full-time customer tracking team.

To Sum Up

So, just to re-cap:

- Establish a Product Road Map - the 2D parallel release stream map will fit most projects nicely.
- Minimize your branching requirements so that your file/component branching models the product road map.
- Manage supersets and build and deliver subsets. Work with the design team to facilitate this.
- Define key baselines with several build increments (baseline + a set of changes) in between each.
- Ensure your Release Identification supports your Product Road Map. Use your tool to identify builds and baselines.
- Make sure your CM tool will translate two builds into a difference set of features, fixes, changes, etc.

If you do these things, you'll come through on top - otherwise you may find yourself buried in complexities!

Joe Farah is the President and CEO of Neuma Technology. Prior to co-founding Neuma in 1990, Joe was Director of Software Architecture and Technology at Mitel, and in the 1970s a Development Manager at Nortel (Bell-Northern Research) where he developed the Program Library System (PLS) still heavily in use by Nortel's largest projects. A software developer since the late 1960s, Joe holds a B.A.Sc. degree in Engineering Science from the University of Toronto.

© Neuma Technology Inc. All rights reserved.