

SCM and Release Engineering: A consultant's hard-earned methodology revealed

Copyright © 2001 D. Sandy Currier. All Rights Reserved.
Permission is granted for Perforce Software, Inc. to copy and distribute.

1. Abstract

In the past ten years, Software Configuration Management (SCM) has become an accepted and documented field in the domain of computer software engineering. There are currently dozens of sophisticated SCM tools from which to choose, and almost all software enterprises now maintain SCM/release engineers on staff. However, it still appears that the SCM/release engineering (SCM/RE) solutions being implemented tend to cause some amount of grief and undue overhead to most companies as evidenced by high demand for SCM/RE consultants. Given that consultants seem to offer the same basic advice and implement the same fundamental concepts time and time again, the author believes that most SCM and release engineering problems can be understood and solved with the help of a simple model.

This paper will attempt to describe a simple method successfully applied by the author both as an employee and as an independent consultant to solve SCM and release engineering issues. The solution decomposes the 'software development manufacturing problem' into separate SCM and RE problems. It will describe the power of Perforce as a SCM tool solution and will introduce ReleasePro as a commercial release engineering solution. However, it is a tenet of this paper that the solutions described herein should be applicable to almost every organization independent of the specific SCM/RE tools.

2. Background – the Software Development Manufacturing Problem

There has been much research in the area of SCM. Several universities offer courses and programs in the field of Software Configuration and Change Management. A few good references are [APPL98], [WING98], and [VANC98], as well as the various user guides and reference manuals of SCM tool vendors, such as Perforce, ClearCase, Accurev, and TrueChange. Advanced users of these products have occasionally also produced valuable public works, such as [GOET99]. However, there is not much research or guidance available regarding release engineering. The reference book "Software Release Methodology" [BAYS99] is available but includes no references.

One short definition of release engineering is:

The engineering steps necessary to move computer files that come directly from the SCM tool (source objects), computer files derived from these versioned files (derived objects), and computer files that come from third party sources, to the end user.

Here the 'end user' is any consumer of the final software product, be it internal to the enterprise such as a QA or operations department, or external to the enterprise such as the final end user of the product. This definition of release engineering is 'mechanical'. It does not include the very real politics and company dynamics that can greatly influence the act of moving these computer files. However, this paper's purpose is to describe a basic and fundamental model to solve the 'mechanics' of SCM and release engineering and thus excludes cultural issues.

Ten years ago, when many software development organizations followed a waterfall development model, managing release engineering was a less significant problem. It occurred once at the end of the development process. However, with today's rapid prototype, iterative development, or extreme programming process models, release engineering is no longer a minor endeavor. As software development increasingly becomes more of an issue of component re-use and third-party product leveraging, all at a higher and higher release frequency, release engineering is becoming more significant. As sophisticated SCM tools allow for a small team to support many software releases in the field, today's enterprises find themselves with significant release engineering problems and bottlenecks on their hands.

Together, the 'SCM problem' and the 'release engineering problem' form what the author refers to as the '*software development manufacturing problem*'. Every software manufacturing enterprise must solve the 'software development manufacturing problem' in a competitive manner to profitably produce a software product. In short, the enterprise must learn how to manufacture software. And at a minimum, the mechanics of both problems need to be solved by the

modern SCM/RE engineer. By addressing the two components of the problem separately with simple 2 dimensional graphs as described below, truly complex software manufacturing problems can be understood, communicated, solved, and automated. This last step, automation, can have profound positive effects on the cost and time lines of software production.

3. Hypothesis: there is a solution for each problem domain

The basic tenet of this paper is that there are two domains that comprise the ‘software development manufacturing problem’: the SCM problem and the release engineering problem. The author’s experience has shown that when the domains are bisected and the two process flows clearly communicated to the players within the organization, simple, elegant, and effective SCM/RE solutions can be more quickly and easily delineated and implemented.

The first domain is the SCM domain as it relates to Parallel Software Development Environments (PSDE’s). A Parallel Software Development Environment is that system nominally comprised of an SCM tool and SCM processes that support multiple active codelines of software development (code as well as documentation, etc.). An important deliverable of the SCM engineering ‘hat’ (role) is an effective PSDE. By effective PSDE, it is meant for example that all the players (developers, managers, QA personal, operations, documentation, etc.) experience minimal SCM overhead while performing their activities. The PSDE tells them what codeline to work on, what codeline is the next codeline to be delivered as a release, which defects have been fixed in which codeline, who is working on what, etc.

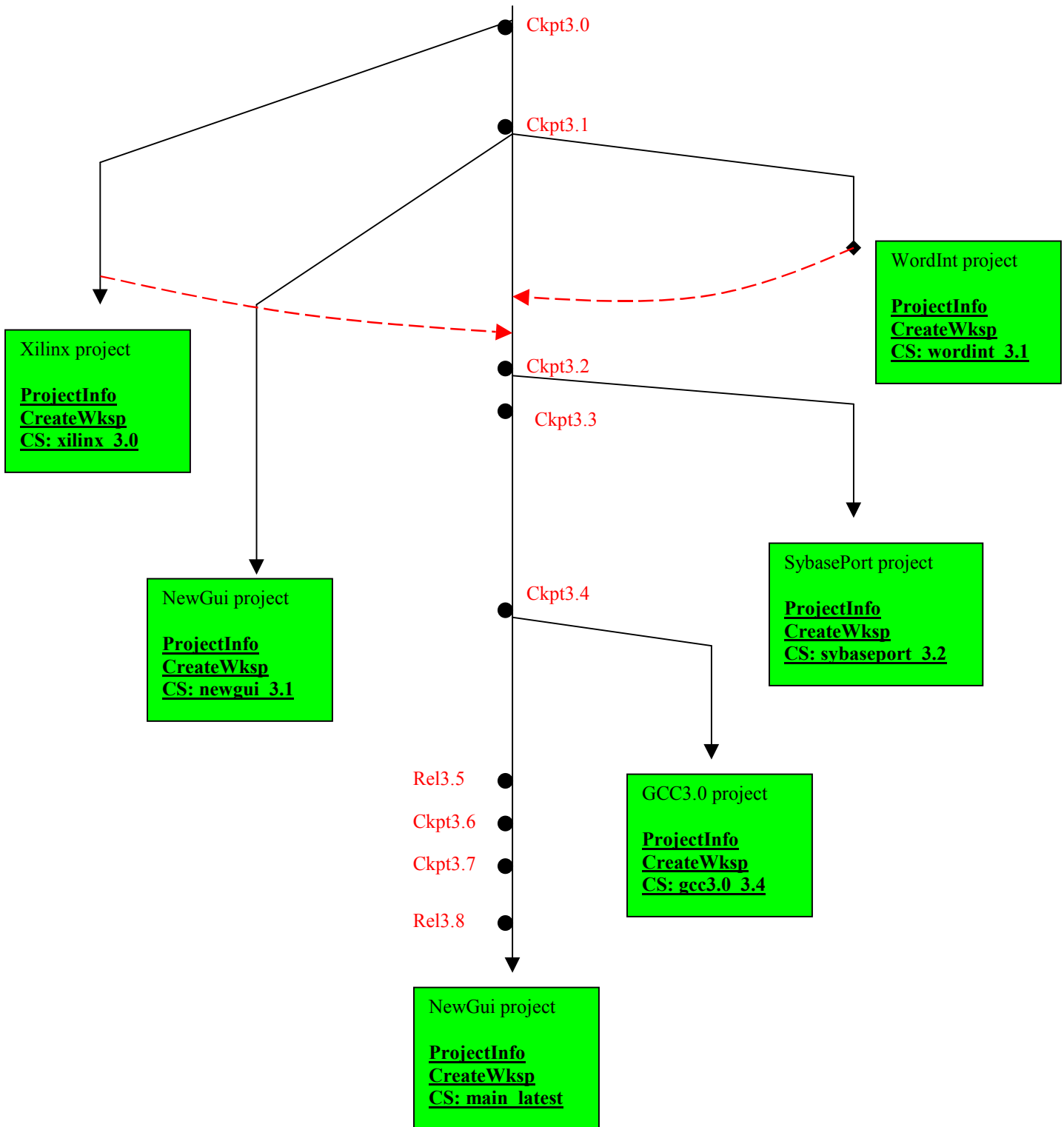
The second domain is the release engineering (RE) workflow domain. This domain contains the solutions for moving a specific version of the software as modified by the development organization to the end user of the software (code as well as documentation, etc.). Quality RE solutions will automate and track these activities and support the ability to determine such release characteristics as lists of changes, lists of defects, etc. Such RE solutions implement the mechanics required to move computer files from developer to end user. An important deliverable of the release engineering ‘hat’ (role) is an automated, audited, and repeatable release engineering solution.

4. Introducing the StreamTree Graph

The StreamTree Graph is a fundamental tool used widely in the SCM domain but known by many different names and with many different properties. The author does not know of any single or original source or claim to be the originator as many people have contributed to its current form. By whatever name, a major goal of the StreamTree Graph is to understand and communicate to all players the codeline nature of the PSDE. The StreamTree Graph described herein contains some specific properties found to be quite useful. It is a two dimensional graph fundamentally representing the various codelines as implemented with the SCM tool. The y-axis is code change or code delta; every change to the computer files within the SCM tool contributes to a downward tick. The y-axis can also be time though not at a uniform metric. The x-axis is the parallelization factor. That is, each codeline will be a vertical line in the graph, and every time a change is made to it, it grows down by a delta. Each time a codeline is branched, a new codeline is created to the left or right of the parent. See Figure 1.

FIGURE 1

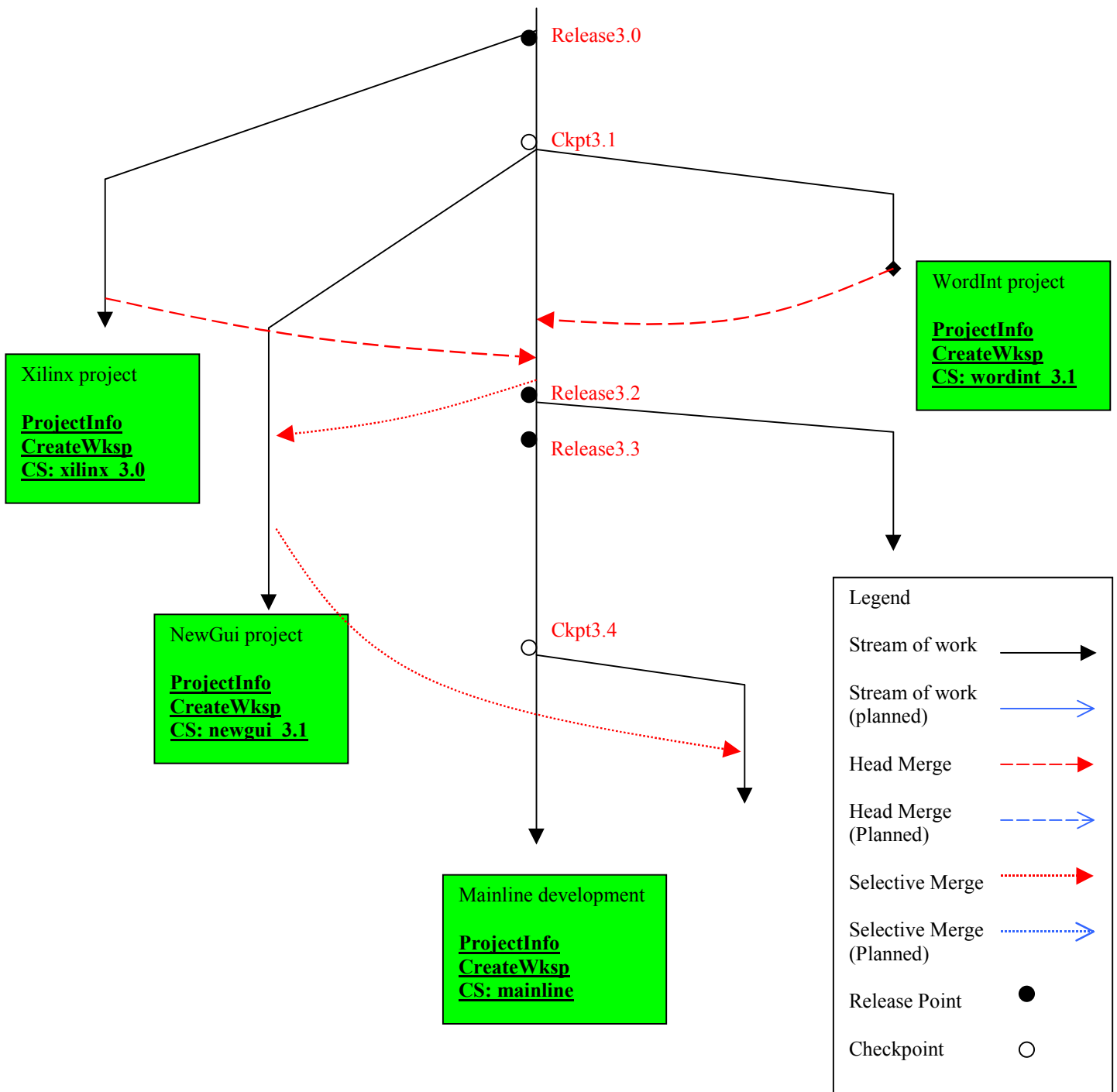
A simple StreamTree Graph with six codelines



Note that a StreamTree Graph is not a version tree of a single file. It is basically a 'sum' of all the version trees of all the files in the SCM system. If any file in the depot/VOB is branched, then the branch is displayed. But, it is also much more than just the sum. It is a summary of all the branching and merging activities performed within the SCM tool. It can record the head merges (full codeline merging) as well as selective merging (merging a specific change from one codeline to another). It can record checkpoints and releases of a codeline. See Figure 2.

FIGURE 2

A StreamTree Graph with selective and head merging records



The StreamTree Graph is produced by the SCM/RE engineer for consumption by the entire development community as well as other departments such as QA, operations, etc. It is a comprehensive and concise communication tool and, in the case of a web page or GUI application implementation, facilitates basic development activities and queries. However, simply maintaining it as an MSWord document or such is also quite useful, particularly if one hangs it by the copier with a date on it.

Additional benefits accrue when the StreamTree Graph is implemented as web page or GUI application. For example, users can click various hyperlinks to run certain queries or execute certain SCM events. In both Figure 1 and Figure 2, the green (shaded) Stream description text boxes actually contain hyperlinks. The **CreateWksp** hyperlink will cause a client spec (workspace) to be created so that the developer can quickly start work on a project. The **CS: <codeline>** hyperlink will display the clientspec (or config-spec, etc.). The **ProjectInfo** hyperlink will take the user to the SCM project info page which contains such info as codeline owner, completion date, etc.

Other interesting hyperlinks also include **ListCRs** (query on the CR's – Change Requests or Modification Requests originating from a defect-tracking system - of a certain point in the StreamTree Graph) and **ListChanges** (query on the changes of a certain point in the StreamTree Graph). For quick deployment, these queries can just point to P4DB page [a free web cgi based tool created by Fredric Fredriksson and available at <http://www.mydata.se/ftp/P4DB/>] in the case of Perforce.

One SCM process model that a StreamTree Graph is not good at depicting is that of the 'moving label'. A moving label process is a method for using a single codeline for two different purposes, such as open development work and stabilization work for an upcoming release. That is, group A works on the codeline in open development mode while group B works on the same codeline, but via a moving label, to stabilize a release of the software. This process does not perform well for two basic reasons: 1) a stabilization change may (unexpectedly or unavoidably) pull in a open development change or vice versa, etc.; and 2) the label creation, management, and communication requires time and effort - time and effort that is usually greater than the time and effort to branch, especially with a SCM tool proficient at branching and merging.

Another SCM branching process that a StreamTree Graph is not good at depicting is the 'sliding baseline' or 'dynamic backing' process. Sliding baselines can be common in tools that can branch on demand such as ClearCase. A sliding baseline is when a branch is created from a dynamic view of the parent or backing branch, such as a "/main/LATEST – mkbranch foo" in ClearCase. Some tools such as TRUEChange do not support this model. Dynamic backing is a tricky and perhaps dangerous model: it does not support easy repeatability since some files will change via one process and other files change via another process as a function of the time of file checkout, as opposed to a static point in time. As a result a user may quickly lose repeatability of her workspace as files are checked out. It is better to statically subdivide the repository into two (or more) processes as a function of directory structure than as a function of time of file checkout. For example, directory projectA becomes a private branch while directory projectB follows the mainline. It is interesting to note that UCM, the process model being rolled out by Rational that wraps basic ClearCase, as well as Christian Goetze's [GOET99] (and other) ClearCase wrappers, either prohibit or do not support dynamic backing.

What follows are some StreamTree rules of thumbs found to be quite useful:

- Supporting the above-mentioned hyperlinks within a StreamTree Graph can be very powerful and enabling for users of the StreamTree.
- A release or an explicit checkpoint of a codeline in the StreamTree Graph is a dot. Sometimes a filled in dot can represent a shipped or supported release, and a hollow dot can represent an internal only release. Such release dots anchor the second graph, the Release Engineering Workflow Graph, to the StreamTree Graph. Branches are always based on a dot.
- A StreamTree Graph can be zoomed in and out as need arises. For complex codelines, the SCM/RE engineer can just display that piece of the overall StreamTree Graph pertinent to the user group at hand.
- Over time, the old codelines and past information is scrolled off the top of the graph. Since the StreamTree is itself stored in SCM, old versions are always available. However, like other process documents, the StreamTree is itself rarely branched – it basically resides in the 'latest only' process depot/VOB.
- It is useful to describe planned future events with the StreamTree Graph such as merges or releases.

For reasons of completeness, what follows are some SCM rules of thumb that effect the StreamTree Graph:

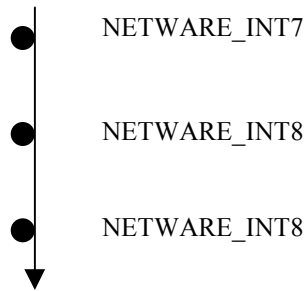
- Only support one major type of development process per codeline – described in the hyper linked project info pages.
- Always try to use time consistent cross-sections for release points and checkpoints. A time consistent cross-section is any set of files and versions where the versions can be represented by a simple time rule. For Perforce, any single change number also represents a time consistent cross-section. In ClearCase, a time rule in a config-spec represents a time consistent cross-section.
- Avoid labels except when handling time inconsistent cross-sections or when the organization requires something more than just a change number or a time rule to denote a dot on the StreamTree Graph. However, it is usually worth at least understanding why and how much an organization ‘clings’ to label usage.
- Always be conscious of the tradeoff between creating a new codeline versus modifying the SCM processes associated with an existing one - as both require effort and time:
 - The cost of communicating and changing how developers interact with a codeline.
 - The cost of communicating and managing a new codeline.
- Codeline divergence costs total about 1.5x the amount of change divergence AND 1.5x the duration in time of the divergence. That is, there is an overhead cost to parallel development, and this overhead is a function of both the size of the change as well as the duration the change remains unmerged.
- ‘There can be only one’ - a single sacrosanct codeline is much better than two or more sacrosanct codelines
 - The development-to-release or development-to-QA-to-release codeline *promotion* models for a Product Development scenario works best in the general case (particularly with Perforce).
 - The gold plus N-development codelines model works best in the general case for the fast and furious IT scenario.
- Codeline lifecycle transitions changes the codeline process – support the necessary ones at least on paper:
 - Support embryonic lifecycle development.
 - Support phased lifecycle development (a.k.a. stabilized releasing).
 - Support enterprise lifecycle development.
 - Support maintenance lifecycle development.
 - For web site based development enterprises, support a continuous evolution model.
- Always have a good place for midnight innovation.
- Have scripts, to the extent possible, be codeline independent (or support a codeline parameter).
- Do not mix latest-only process files with those native source files being versioned as these are two different types of versioned objects. The former, such as the StreamTree Graph, triggers, daemons, web documentation, etc., should be versioned, but only the latest version is of interest. Native source files can have different versions being of interest at the same time and are described with the primary StreamTree Graph.

Using a StreamTree Graph as a communication tool, an SCM/RE engineer can communicate different SCM codeline processes that a software product may undergo. Four common SCM processes that are a function of codeline lifecycle are: embryonic, phased, enterprise, and maintenance. An embryonic development process is one where there is only a single codeline and no branches. The project is just experimenting with code and ideas. Phased (or stabilized releasing) development is when the project is producing stabilization codelines usually just prior to a product shipment. At this point the product is usually behind schedule and a relatively company wide interest is being focused on SCM and release engineering. This SCM process is simple: create a codeline off the single development codeline to stabilize the features for a release.

Enterprise development is when various projects or activities are isolated from each other so that they can proceed relatively independently from each other. This type of development is usually characterized with a single sacrosanct codeline that accepts no individual checkins. All activities occur on either individual activity codelines off the sacrosanct codeline or in a shared development codeline off the sacrosanct codeline. The shared development codeline has the advantages of lower SCM overhead coupled with faster sharing of changes between activities. Which actual development codeline ships next may be unknown due to changing project requirements, unknown complex bugs being discovered in real time by the customer base, etc. The prioritization of which project ships next is largely a function of events impacting the enterprise in real time. (The SCM/RE engineer should by this time have the SCM/RE mechanics well understood and automated and as independent as possible from the real time events that can occur.)

The maintenance codeline process is usually a simple process where submits are closely managed and immediately merged forward. In the case of multiple maintenance codelines there can be multiple forward merges through each active and supported maintenance codeline. Figures 3, 4, 5, and 6 depict these general development models.

Figure 3
Embryonic Development

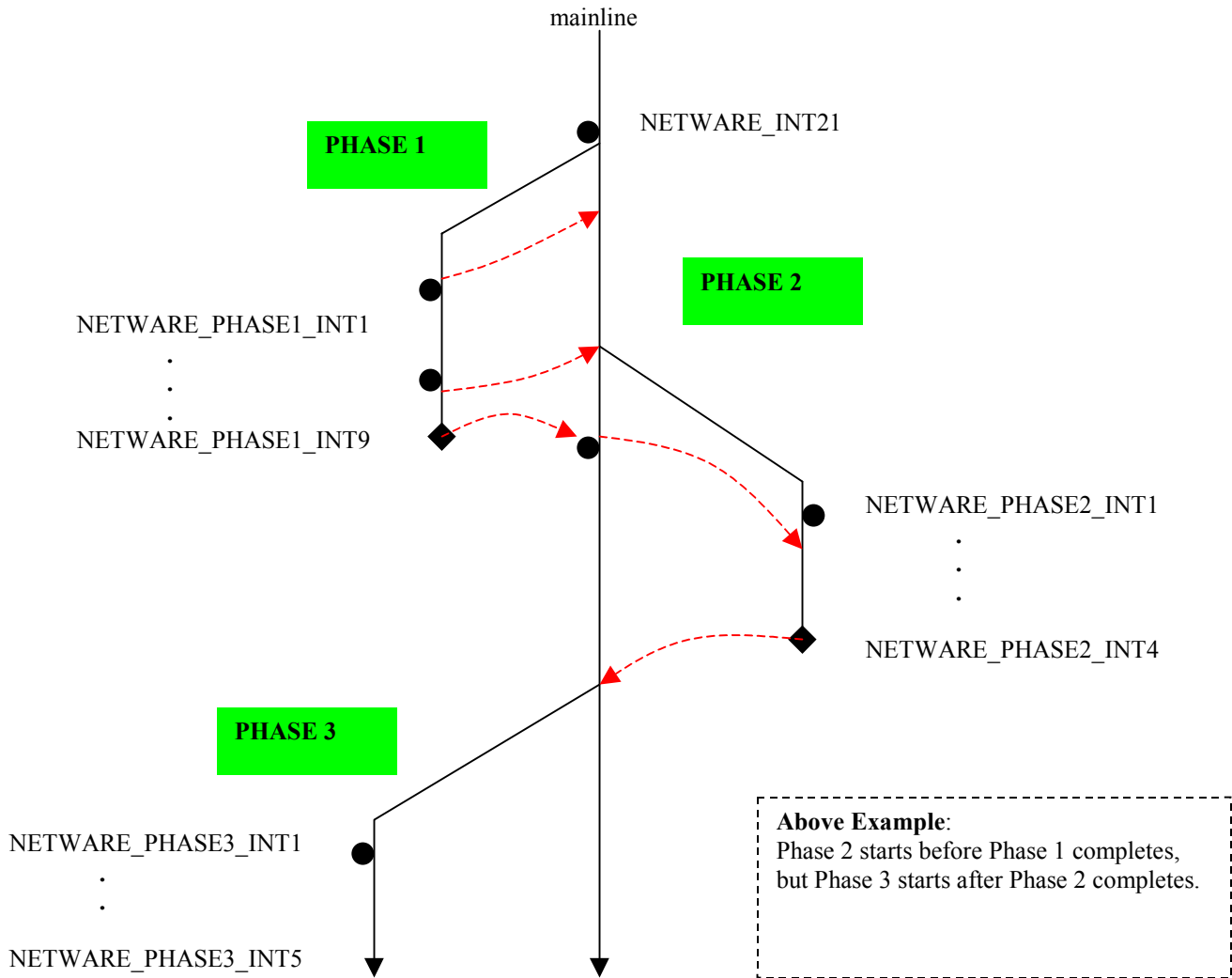


Rules of the Road

- 1) All development occurs the on the mainline.
- 2) Branches are not normally employed.
- 3) Releases (to SQA or wherever) are via regular checkpoints.

Figure 4

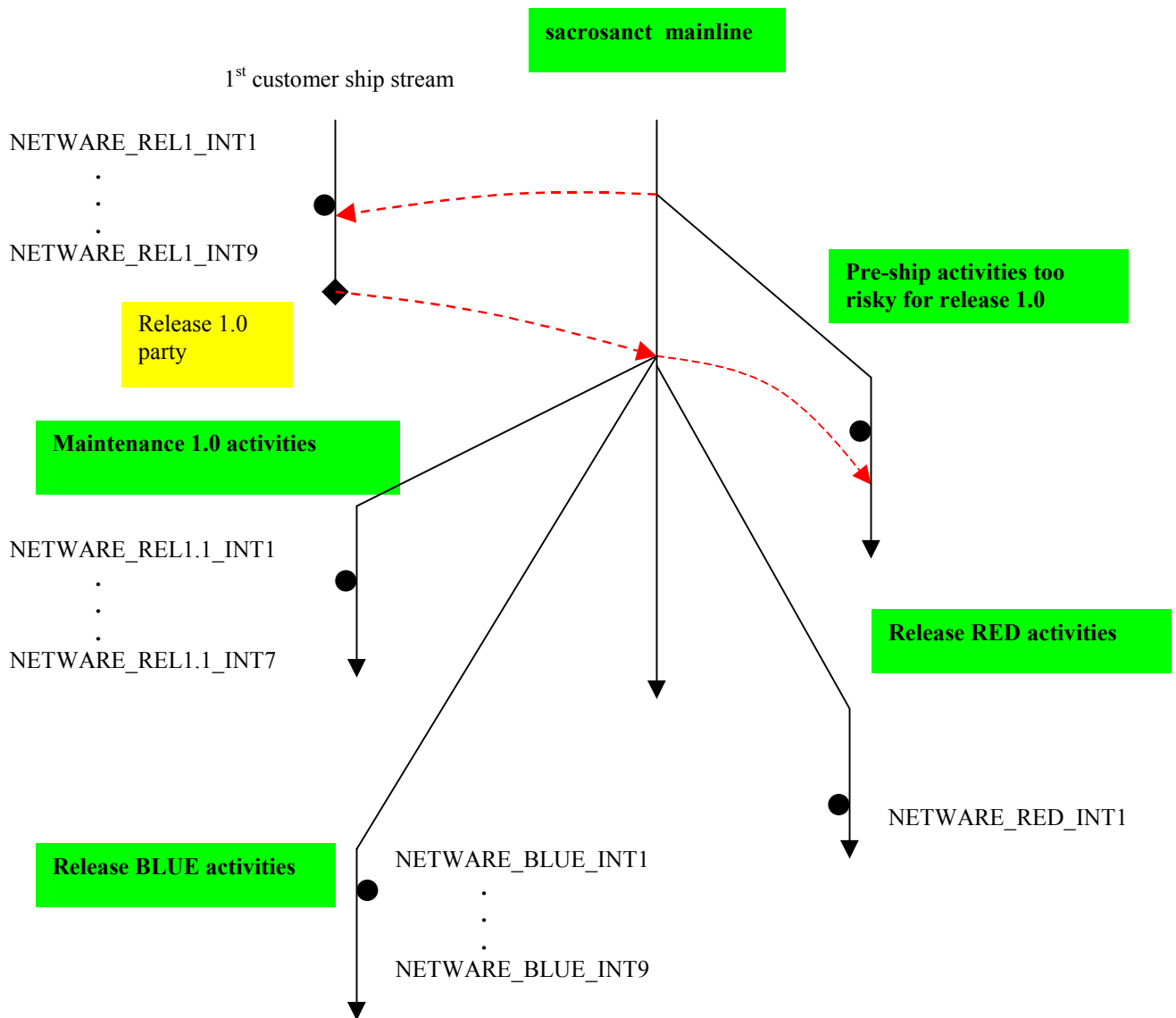
Phased Development (a.k.a. Stabilized Releasing)



Rules of the Road

- 1) Each stabilization codeline (each phase) is branched from main.
- 2) Any completed and tested work on a phase branch is immediately merged to mainline.
- 3) When a phase is completed, it is shutdown except for emergency work.
- 4) Phases can share individual changes.
- 5) Phases are rebased from the mainline as frequently as possible.
- 6) The same basic graph can be used for either creating actual releases to the field or intermediate releases for internal testing.

**Figure 5
Enterprise Development**

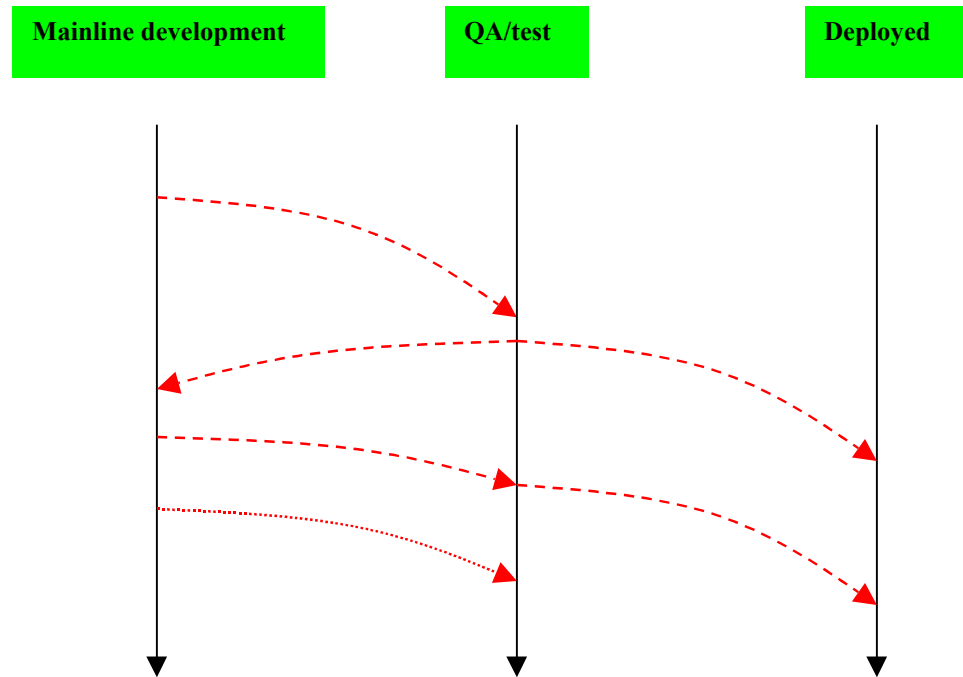


Rules of the Road

- 1) The mainline is always sacrosanct: no open development, no kitchen sink.
- 2) Release streams **MUST** rebase from the head of the mainline before releasing and then merge to after. Proof is the demonstrated assertion that no changes (defects or changes) have been dropped from a previous release, and no compatibility issues exist with other 'live' components (if relevant). Project/stream lead and/or R&D lead can decide.
- 3) All streams **MUST** rebase (merge out again) with mainline as soon as possible when the mainline changes. [Reason: better to share code and discover bugs now than later]
- 4) Streams can share individual changes (changes can be selectively merged).
- 5) Streams can be merged together if desired.

Figure 6

Web-based Continuous Evolution Development

**Rules of the Road**

- 1) All development occurs on the Mainline.
- 2) Note the absence of labels or checkpoints – the Deployed codeline may be in a state of continuous update.
- 3) There may be different *promotion* processes present that are a function of directory, file type, etc. For example, all .gif files in the ‘picture’ directory may be able to be *promoted* without QA testing. By *promoted* it is meant merging the change to the next most stable codeline.
- 4) The mainline can be branched for isolated development, but all *promotion* of code to the QA codeline occurs from the Mainline.
- 5) The Deployed codeline is optional – there may only be a two-step promotion.
- 6) Individual changes can be selectively merged back and forth.
- 7) Any changes done on QA or Deployed are immediately merged backwards.
- 8) The default method of integration from main to QA (or from QA to Deployed) can be a copy-merge (-at). This assures that what the developer has tested is exactly what QA will test (and will be exactly what is deployed) and that binary files (such as .gif’s and .jpg’s etc.) are promoted easily. Real merging can be the exception when needed. Note: this copy-merge solution may require scripting support to handle dependent changes.

5. The Power of Perforce

Perforce is one of the best SCM tools for implementing a StreamTree Graph for several reasons. One is its ability to natively manage changes (changesets/changelists) out of the box. When a SCM tool does not natively manage changesets, the SCM/RE inevitably will spend a large amount of time implementing them by hand. Why?

- Because changesets offer a very effective association between files edited for the same purpose. It is very effective for a developer to see what files another developer has changed for a given activity - it speeds learning, etc. It is best when this capability is directly part of the SCM system as opposed to an integration with a defect tracking system. Integrations with a defect tracking system can be a very powerful integration in terms of defect and enhancement tasks, but using defect tracking to implement changesets adds process overhead when it is not needed (e.g. embryonic development). Additionally, it is unnaturally splitting an object that can be considered a pure SCM object across the two databases, the SCM database and the defect database. And it may lead to non-atomic transactions.
- Because changesets reduce the amount of information that players must sort through. For example, looking at a list of 5 changesets is easier on the eyes than looking at a list of 25 files.
- Because in many cases, it makes the act of merging easier. It is usually easier to merge a change or a group of changes than it is to merge a manually constructed list of files.
- Because changesets also support the easy ability to selectively merge a single change in isolation of the other changes on a codeline. However, they do not necessarily prevent merging problems with respect to algorithm conflicts. [Note: there are two types of conflicts that can occur when merging – blatant and algorithmic. Blatant conflicts are when the same line of code has been edited differently in both variants when compared to the base. Algorithmic conflicts are conflicts that result from the algorithm as implemented in the code being broken due to non-overlapping lines changing in the two variants.]

Another power of Perforce is its ability to track selective merges. Very few tools can track selective merging, and Perforce does it well. Perforce also tracks head merging when the codelines are directly parent-child related. Though it does not work well when head merging between non parent-child codelines, this SCM usage model can be minimized in most scenarios. And since Perforce can successfully track selective integration records, it can track what changes have been integrated into what codelines. That is, if a change 555 on codeline C has been merged to codeline B and then merged to codeline A, codeline A can be queried to report that it contains change 555.

Perforce also easily supports the three basic merge types: ignore-merge, copy-merge, and normal merge. Ignore-merge is when the incoming version of the file is ignored but the merge is still recorded. Copy-merge is when the target version is ignored, and normal merge is when the two versions are merged with or without conflicts. Being able to easily specify the type of merge can be quite enabling, as seen in Figure 6 with the use of the `-at copy-merge` switch.

Another powerful aspect of Perforce is its ability to support defect queries via the perforce `'p4 jobs'` command. This query, like the `'p4 changes'` query, will follow all the integration records of the files in a codeline to report which jobs (linked defects) of a certain characteristic are associated with the codeline. It is this query that allows any player to determine which defects are fixed in a specific release, or the list of differences in defects between the two specific releases. Even if Perforce jobs functionality is not used as the defect tracking tool, by mirroring the defect tracking database with Perforce jobs one can then directly support these queries. The author has used the P4DB package for this purpose.

What follows are some rules of thumb for successful Perforce usage models focusing on the StreamTree Graph:

- Implement branches at the level below depot name. The branch (codeline) becomes a pseudo first class object.
 - Codelines being first class objects are good; e.g., being able to quickly tell what changes a codeline contains.
 - Watch out for delete-only changes and the have list - the have list does not contain them.
- Use change numbers instead of labels
 - Branch on demand is easy when change numbers are used - change numbers are immutable unlike labels
 - Without labels, one does not need to manage a label naming process
 - When change numbers are used for release points (StreamTree graph dots) instead of labels, it allows easy delineation of what changes are in or out.
 - Note that time inconsistent cross-sections add complexity, particularly with change queries. It is best to avoid them and to get early buy-off for this. Use labels when time inconsistent cross sections are required.

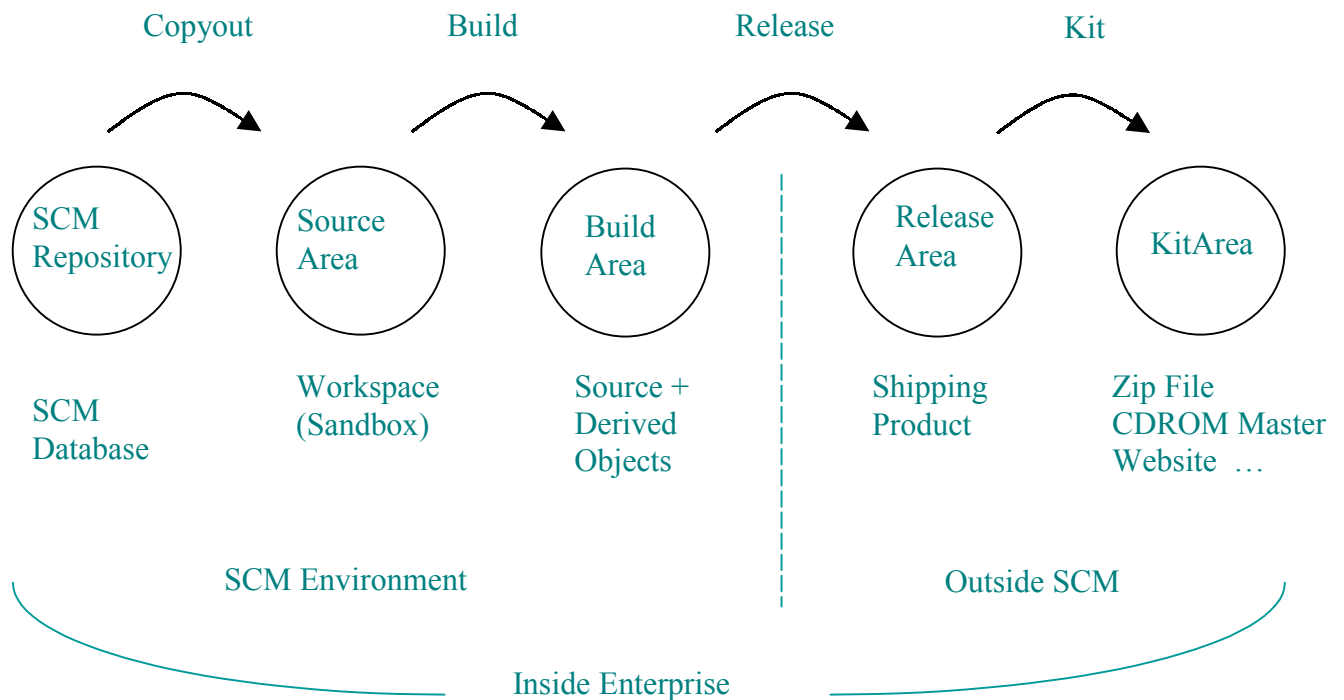
6. Introducing the release engineering (RE) Workflow Graph

The release engineering (RE) Workflow Graph is employed to understand and communicate to the relevant players, usually not the majority of the players, how the release mechanics have been implemented. It describes the workflow of a specific set of versions of files from the StreamTree Graph (a dot on the StreamTree Graph) to the files as installed on an end user system. By having a clear and accurate RE workflow graph, certain classes of questions by the development staff and other players will be easily answered by simple inspection of the graph.

It is a two dimensional graph of a sequence of circles and arrows with time going from left to right. See Figure 7 and 8.

Figure 7

Product Development RE Workflow



The above is a typical workflow for an enterprise that produces a commercial product. It is referred to as a **Product Development (PD) Workflow**. It is geared to enterprises that produce software that is consumed by end users outside the enterprise. The end user completes the installation process with the following workflow (a continuation of the above figure):

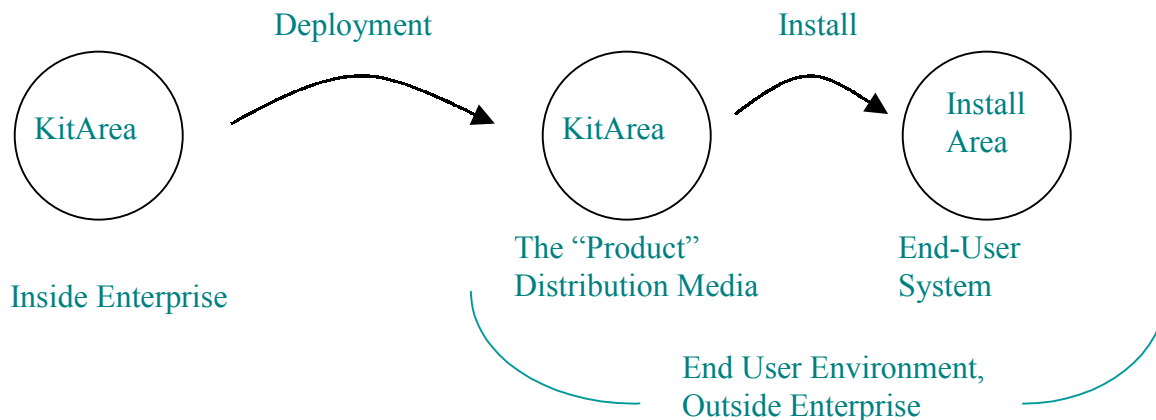
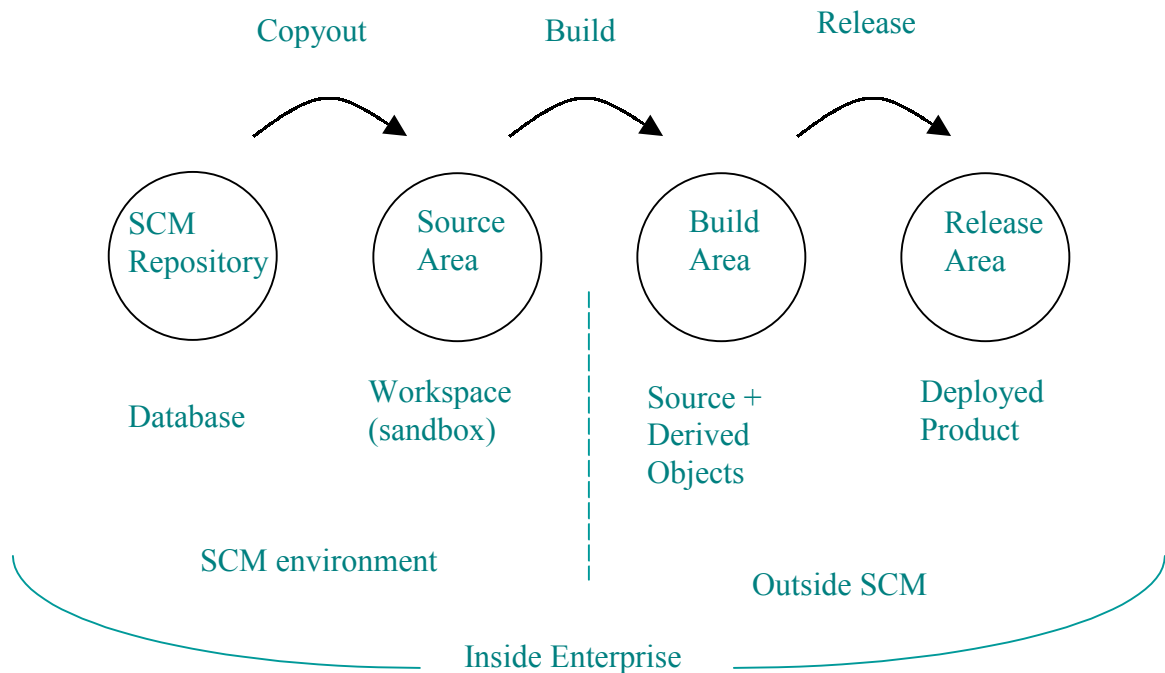


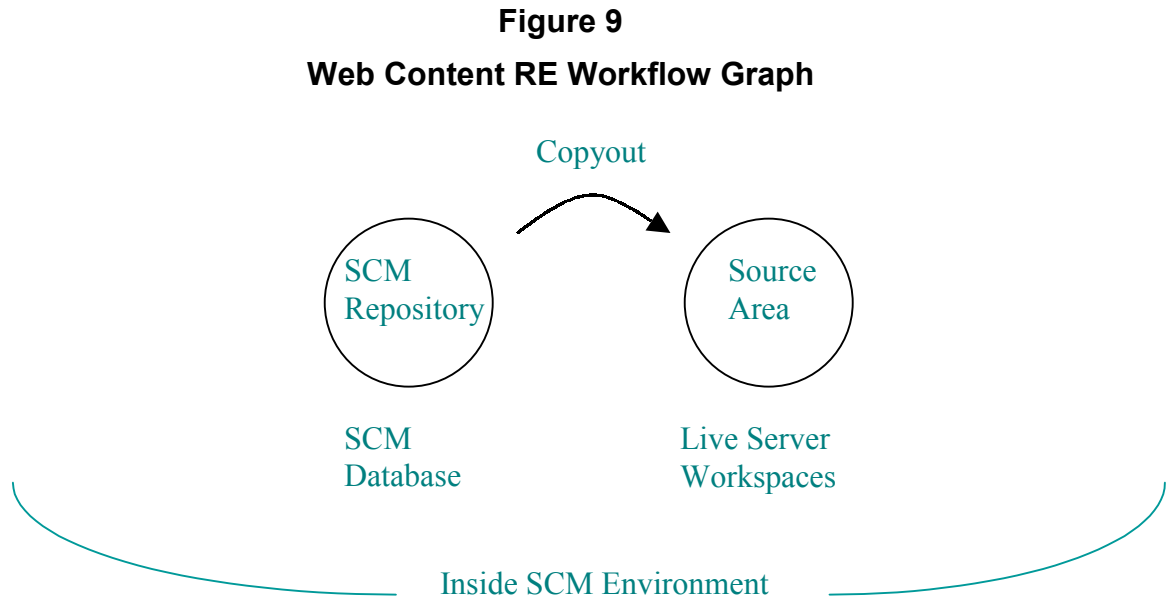
Figure 8
Information Technology RE Workflow



The above is a second variation sometimes referred to the **Information Technology (IT) Workflow**. Here, the enterprise owns and manages the end user systems of the product (which may be database applications, web pages, etc.). The installation process is typically lightweight, possibly comprised solely of file copying. Note that though the SCM promotion process is of utmost importance (e.g., to keep software that supports real time applications, such as financial applications, etc., from being broken), a complicated multi-step SCM process is not part of the RE workflow. The RE workflow simply moves a dot on the StreamTree Graph in a repeatable and automated manner to the end user system.

The IT Workflow can become interesting if there are numerous products being supported that have their own release schedule or if there is a complex run-time dependency. In such cases each individual product may undergo a separate PD or IT workflow allowing the QA/Operations group to choose between various releases of each product when assembling the final product. QA/Operations may in fact select the specific release by pulling it out of the SCM system if the PD Workflow stores the Release Area in the SCM system (referred to as *staging* the release). The SCM data generated when staging should not be included in the StreamTree Graph because the StreamTree Graph concerns SCM data of only the versioned sources. For example only the release hyperlink (the StreamTree dot) that points to the release information should be present in the StreamTree. Also, when staging one should stage into a different depot/VOB then the source depots/VOB's since the staging depot/VOB typically follows a different branching, merging, and directory naming process. See Section 7C for more information.

A third variation of a RE workflow graph is the Web Content Workflow. The workflow of updating the *front-end* web content of a website can be different in that the entire publication process can be implemented within the SCM environment. See Figure 9.



It is possible to design the RE workflow for the live web servers so that the SCM system is directly employed for updates. This design has several fundamental advantages:

- The same SCM system that is used for code development can be used for the update process. This can greatly simplify learning curves and reduce maintenance costs. However, many systems may require some amount of scripting to create such functionality as server update logs, email notification, etc.
- The SCM system is ideal for quickly regressing a change in the live servers, or verifying that a change is present.
- All the players can have access to the exact state of the live servers via the SCM system and can duplicate it on a local test/development environment. This decreases time to debug problems, etc., without requiring direct access to the live servers themselves.

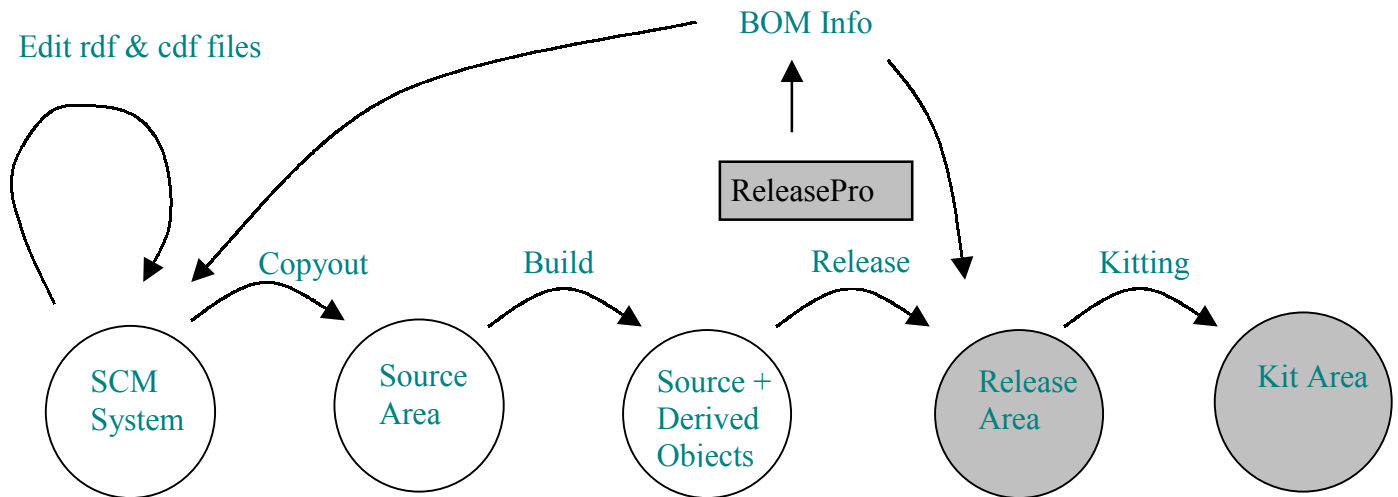
Regardless of the specific flow, the engineering workflow graph depicts the steps to produce a product release, including the major steps of copyout, building, releasing, packaging, deployment, and installation if applicable. In a similar manner to the StreamTree Graph, the workflow graph will contain detailed information specific to an enterprise.

It is interesting to note that kitting and unkitting appear to exist basically to protect the deployment step of the workflow. For some IT enterprises and website content scenarios where the deployment channel is already secure and not bandwidth limited, one would not necessarily need to spend resources kitting and unkitting.

7. Introducing ReleasePro

ReleasePro is a commercially available release engineering product. ReleasePro automates the release portion of the RE workflow graph. Figure 10 depicts a RE workflow graph incorporating ReleasePro:

Figure 10



ReleasePro is currently integrated with Perforce and ClearCase. It reads codeline and version information from the SCM system, generates sealed bill-of-material files in a patent pending manner, and writes data back into the SCM system. ReleasePro is data driven in the form of ASCII text release information contained in Release Description Files (rdf files) and Component Description Files (cdf files). These files are stored in the SCM system along with normal source code. They follow whatever codeline policy is present, and can be merged like any other file. When a release is created, ReleasePro can store release information back into the SCM system as well as the release itself (storing a release into the SCM system is frequently called staging the release).

With Bug tracking integrations enabled, ReleasePro will store information into the bug tracking system to support defect queries that can be run against a 'release'. Thus, as changesets and codelines have become first class objects in SCM tools, ReleasePro adds the capability of a 'release' becoming a first class object. In addition, when integrated with installation software, ReleasePro can generate installation specific files, such as Install Shield .fgl files (fgl files list the files and their attributes to be included in the self extracting archive). On the installation side, ReleasePro utilities can validate that the release has been correctly installed or if the installation has been compromised. The utilities can also reveal release specific and SCM specific information associated with the installation in isolation from the originating SCM environment.

By using ReleasePro, the SCM/RE engineer can quickly automate the release step in the RE workflow graph. For some companies with an existing SCM solution, this step alone can bring the company to CMM Level I compliance. But regardless of CMM or ISO rating, or if Extreme Programming (XP) is being employed, the automation and management that ReleasePro offers can drastically reduce release engineering bottlenecks and free SCM/RE engineers to spend more time on other tasks.

8. Some Generic Examples of using the StreamTree and RE workflow graphs

This section focuses on using a StreamTree and RE workflow graph to understand and solve some standard SCM/RE questions and problems.

A) What does it mean to promote a codeline?

Your manager wants to stabilize a codeline for SQA testing while still allowing for open development to occur. In a sense, s/he desires a codeline to be promoted for testing to see if it can qualify for an official release.

The challenge is to determine which is better - to create a new codeline with new process, or to change the process on the existing one. From the author's experience as well as other references, is it usually better to spawn a new codeline as this will only directly affect those people working on the stabilization codeline. However, it will also affect other developers when they need to fix a problem on this codeline - they will need to create a new client to do the work, and then merge the work back into the parent branch. The StreamTree graph gives them an easy way to create the client and submit the changes. A trigger or a review daemon can enforce (to the necessary degree) that a merge is completed from the promoted codeline to the parent.

Note that even if the change is not destined for the parent codeline, it is best to have a ignore merge done in any case. This will allow a) automatic needs-to-be-integrated scripts to not flag this change as still needing to be integrated; and b) will allow a following change to be integrated over without merging this change.

Note that the workflow does not change! That is, the same automatic release process used to make a release from the parent codeline (from a release 'dot') is also used to create a release from the stabilization codeline. In the case of ReleasePro, since the release lists as well as the ReleasePro executable are stored directly in the SCM tool, any branching and merging issues with respect to releasing are handled via the PSDE.

B) What does it mean to make a release of the software?

A customer needs an emergency release to a fix a problem. Or, perhaps QA needs a new release that includes last week's development work.

The interesting aspect of making a release is that the SCM process does not change nor is it affected. The same automatic, repeatable process can be used. On the StreamTree Graph a new release dot is recorded. In the case of ReleasePro, the program is simply executed once again. With a release management database, ReleasePro will also record the fact that a release has been created. This will support future queries and various drop down lists, such as diff'ing releases in terms of either changes or defects.

C) What does it mean to stage a release of a sub-product of the software?

Your company produces many independent sub-products or components that make up a larger product, but each group that produces the component is isolated and independent from each other. For business reasons, each group only hands-off to operations (or SQA) their isolated piece.

Sometimes, an enterprise is organized as mostly independent and autonomous small groups. Each group is responsible for a small sub-piece of the entire deliverable which could be a company (web) portal, a complex test system, etc. In particular, the build process for each piece may be mostly independent of the other pieces, and the release may be comprised of derived objects instead of source objects and the build process. This is probably one of the more difficult solutions to design and implement since it requires a RE workflow for each component and perhaps an additional RE workflow for the entire product. And there may be many pieces (100+) with frequent releases.

Ideally what is desired is a *release management system* such that when each group manufactures a new release of their component, the release management system tracks the details of the release as well as the release contents itself. In this manner all the players of the enterprise can query the *release management system* for the release details as well as obtaining any release itself. However, an independent and native solution would introduce yet another infrastructure tool and database. TRUERelease by True Software attempted this but, in part because it added a layer of complication, it failed commercially.

There are two general solutions that leverage existing infrastructure and databases for a release management solution. One is to craft an ordinary computer directory structure that maintains the contents of released components. The component releases are released into this directory structure with the necessary native file system protections to prevent accidental modification. In addition to the directory based repository a text file or real database is employed to track the release meta-data (such as component name, codeline information, file system location, comments, etc.). Current ReleasePro functionality can be used to automatically maintain either type of meta-data as releases are generated.

The second general solution is to use the SCM system itself as the release repository. The SCM system can be used to implement the necessary and perhaps finer grained control (Access Control Lists - ACL's) that will secure the release from accidental modifications. Any existing SCM infrastructure with regards to email triggers or other such processes can be leveraged. A separate depot/VOB from the source depots/VOB's should be used for staging. And once released,

the SCM/RE engineer can certify that the release will not be lost and will continue to be available via the SCM system. Again, current ReleasePro functionality includes this ability to stage a release back into the SCM system.

If a text file is used to store the release meta-data, it should also be stored in the SCM system. If a database is used, it is probably a good idea to use an existing database, such as a defect-tracking database. If the defect-tracking database is used, then such queries as "What bugs are fixed in this release?" can be implemented since the defect database will now contain release information.

D) What does it mean to support a released version of the software?

The development team will need to be able to repair a released version of the software sometime in the future.

Note that on the StreamTree Graph there is already a dot that marks a release point. At any time in the future, if a maintenance codeline is required, a codeline can be created from that point. If there is a likelihood that the need may arise and it may be urgent, the codeline can be created beforehand. Otherwise, the SCM/RE engineer can create it on demand.

Note that a maintenance codeline will follow the maintenance process, which is different from other codeline lifecycle processes.

E) Where does the unscheduled innovation go?

There are several very creative developers who are friends of the CEO and are generally anti-process.

Developers need a place that is very near the tip of the single sacrosanct codeline (if there is one). For embryonic development, all development can occur on a single codeline. In enterprise development, it is not the sacrosanct codeline, but close to it. But regardless of the existence of a sacrosanct codeline, there almost always needs to be a midnight oil codeline near the tip of the sacrosanct codeline.

Also, note that the RE workflow graph does not change even for such a midnight oil codeline.

F) What about iterative process model?

The development team delivers a release to SQA or operations every week.

In this model, the release cycle is almost continuous. The RE workflow must be automated. The StreamTree Graph also becomes more critical in that the information it contains is used on a daily basis by project management to determine what changes are in and what are out for the next release.

Regardless of the development cycle (the time a development team needs to complete an activity) or the QA test cycle (the time it takes the QA/Operations team to validate a release for deployment), releases are continuously being created to fully utilize the testing resources.

With an automated RE workflow, the SCM/RE staff does not become a bottleneck for this type of model, and the StreamTree Graph keeps the communication at a high bandwidth.

G) Extreme Parallel Software Development Environments - will the real 'next' release please identify itself?

If the development cycle is longer than the SQA test cycle while the business model requires continuous releasing so to have constant updates to the software, then things become very interesting. In this case instead of a single 'release train' codeline, there may be multiple 'release trains' codelines. For example, if the development cycle is 3 weeks and the QA test cycle is 1 week, then there may be 3 development codelines (release trains) active at any given moment. This situation usually results in an IT scenario where development is supporting an IT product, such as stock market trading program, real time investment programs, and real time web based operations, etc.

In this model, the StreamTree Graph is very important in that a developer or manager will need to determine which codeline to use to perform the work. Project management will also need tight coordination with both the development teams and the QA test teams. A fast and efficient RE workflow will be required. The SCM design will need to support a release being delivered from one of several competing development codelines. Which codeline will be the next release

is unknown when the codeline is created (since the business model is based so much on real time events). In addition, changes from one development codeline may need to be merged to another codeline at the last minute.

From a SCM point of view, the ‘there can only be one’ sacrosanct codeline works best. Here, each development codeline is spawned from this gold codeline. Which ever codeline really ends up winning gets merged to the gold codeline first. This merge will be a conflict free merge. The other codelines then need to rebase before their official release is created. Process can be implemented that prevents an official release from a development codeline unless it has been rebased from the gold codeline.

A variation of this is that the shipping codeline gets merged to the gold before actual shipping, but is usually less optimal since it breaks some of the fundamental good SCM tenets listed above; namely, it changes the process of a codeline (the gold codeline changes from sacrosanct to pre-shipping and back). It also temporarily pollutes the gold codeline, which may be problematic.

This ‘there can only be one’ model can be documented via the StreamTree Graph as seen in Figure 5.

H) 24x7 building

A separate process not directly related to StreamTrees and RE Workflow Graphs that is worth mentioning concerns the build phase of the RE workflow. This is the 24x7 build daemon. Based on the original Perforce Review daemon concept, a 24x7 build system is a review daemon that constantly monitors relevant submits. When one or more are found, the process kicks off and monitors an incremental build of the product. [Note: an incremental build is one where the build is not done from scratch. Derived objects are used as much as possible. Dynamic Views in ClearCase are excellent for implementing this. However, with a little skill, accurate and minimal-time incremental build systems can be created with just make.] The 24x7 build daemon then sends email to the owners of all the Perforce submits. If the build is good, the email simply states it. If the build is broken, it emails the (pertinent) output of the build but fails to advance the Perforce 24x7 build counter. In this manner, more and more submit owner’s get the email of the broken build until the build is finally good. A 24x7 log directory is used to store the various build logs and the persistent good/bad build status information.

This keeps up peer pressure to not break the build while offering immediate feedback as to whether a submit has broken the build. This also works well when a developer may not have access to all the build platforms or build environments that the build engineer may have access to.

A 24x7 incremental build daemon and a nightly build-from-clean daemon are a good way to insure correctly building codelines with minimal down time.

9. Summary SCM/RE Observations

Probably the two most important observations that the author has learned as a SCM/RE consultant is: 1) to solve PSDE type problems with the StreamTree Graph employing a dozen or so rules of thumb; and 2) to automate the RE workflow with a tool like ReleasePro. It has allowed the author to walk away from many a successfully completed contract.

Perforce has been incredibly useful as the SCM tool solution because of its support of changes and tracking of changes and defects. By making codelines first class objects the power of Perforce can be leveraged to a high degree, and many powerful queries can be easily executed.

ReleasePro has been invaluable as well. Having a release engineering tool fully integrated with the SCM system such that branching and merging designs continue to work for the RE workflow is a very powerful consulting tool. As described above, having an automated RE process is important since the RE workflow rarely changes even when the StreamTree Graph does change and vice versa.

10. ReleasePro Description

What follows is a short description of ReleasePro (see www.releng.com)

- Advantages
 - Dramatically reduces the engineering labor hours required to generate a product release. The computer generates a release, and the SCM/RE engineer is freed to perform other tasks.

- Allows for fully automated releasing and production of a product.
- Allow developers to self manage what files are being released on what codeline.
- Handles file permissions, ownerships, ACL's, MD5 signatures, product/component definitions, SCM information, etc.
- Increases the efficiency of software development by enabling anyone to create releases in the same manner as release/SCM engineers. Such release process become data driven and repeatable, etc.
- The release list files are stored in the SCM environment and follow the PSDE rules (note: they are text files and can be merged).
- A reverse association is established between the installed software and the SCM environment.
- Good install time capabilities are supported
- Integrated with Perforce, ClearCase, InstallShield (hopefully Starteam and ClearQuest soon)
- The fact that a release has been created is recorded in either the SCM environment or the defect-tracking environment. The release becomes a first class object.

Trademarks and References

ReleasePro is a trademark of Release Engineering Inc.

ClearCase and ClearQuest are registered trademarks of Rational Software Corporation.

Perforce is a trademark of Perforce Software.

AccuRev is a registered trademark of EDE Development Enterprises, Inc.

InstallShield is a registered trademark of Stirling Technologies, Inc.

TRUERelease is a registered trademark of True Software, Inc.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Windows is a registered trademark of Microsoft Corporation.

All other trademarks or product names are trademarks or registered trademarks of their respective companies or organizations.

- [APPL98] Appleton, Brad, Stephen P. Berczuk, Ralph Cabrera, and Robert Orenstein, "Streamed Lines: Branching Patterns for Parallel Software Development," Submitted to the 1998 Conference on Pattern Languages of Program Design (PLoP'98), Allerton Park, IL, August 1998.
<http://www.interact.com/~bradapp/acme/branching/>
- [WING98] Wingerd, Laura and Christopher Seiwald, "High-level Best Practices in Software Configuration Management," draft of a paper to be presented at the Eighth International Workshop on Software Configuration Management, Brussels, 1998. Available at
<http://www.perforce.com/perforce/bestpractices.html>
- [VANC98] Vance, Stephen, "Advanced SCM Branching Strategies" Submitted to the 1998 Perforce User's Group Conference, San Francisco, CA 1998. http://www.vance.com/steve/perforce/Branching_Strategies.html
(Slides can be found at: <http://www.vance.com/steve/perforce/sld001.html>)
- [BAYS99] Bay, Michael E., "Software Release Methodology" Prentice Hall PTR, ISBN 0-13-636564-7, Copyright 1999
- [GOET99] Goetze, Christian, <http://www.miaow.com/clearcase>, a ClearCase usage model distributed under the GNU G.P.L., Copyright 1999