CM+ Concepts and Overview Guide

Introduction to CM+

Neuma Technology has been providing Configuration Management (CM) and Application Lifecycle Management (ALM) solutions for over 20 years. Neuma's flagship product, CM+ Enterprise, is the world's most advanced CM/ALM Suite and the only full 4th Generation CM/ALM Suite. At the time of it's release, CM+ was identified by the CMII Research Institute as the only CMII-certified SCM tool. A closer look at CM+ indicates why it is well suited to support the highest CM maturity levels in the industry.

With deep roots in the software intensive telecommunications industry of the 1970's and 1980's, and with over a quarter century of subsequent evolution, CM+ provides a level of CM maturity unmatched in the rest of the SCM industry. CM+ supports stream-based change management out-of-the-box, with main branches per release. It supports globally distributed teams as if they were located at a single site. It provides unprecedented reliability and availability to support mission-critical operations. CM+ is easily scaled from a couple of users to a couple of thousand users, without having to scale up your servers. And it is easy to customize to specific process and user requirements.

CM+ seamlessly integrates all of the ALM functions, from Requirements Management down to Build and Release Management, in a single tool suite. A single role-based user interface eliminates the need for training across multiple tools. All data resides on a Next Generation Hybrid repository (the STS Engine), specifically designed to meet engineering and business management requirements. This simplifies traceability, reporting and data analysis, while allowing rapid point-and-click data navigation. CM+ MultiSite works across all CM+ management functions giving up-to-the-minute information to all of your project sites around the world. The near-zero administration is the same across all functions, and all sites, greatly reducing support overhead.

CM+ provides unmatched reliability and data security. Warm standby disaster recovery capabilities may double to provide emergency, up-to-the-minute backups. A multi-layered data granularity allows easy checkpoint and recovery of all project data. Logically read-only data partitions permit full backups with a fraction of the resources (time and storage) needed to create them. A fully traceable easy-to-read transaction journal allows for recovery from potential data sabotage, while providing further insurance against disk errors and errant backups. As well as front end access protection, CM+ provides both at-rest and message-based data encryption on top of your secure environment. As well, data read and write access can be controlled down to the finest levels of detail.

This guide provides an overview of CM+ capabilities as well as a description of basic user operation in the out-of-the-box configuration. With the reduced learning-curve expectations in the technology industry, Neuma recommends that each customer publish a quick-reference card specific to each role in the customized deployment. A customized half-day or full-day course for each user is also recommended to supplement this guide.

CM+ 7 Primary New Features

This section deals with a summary of recently added CM+ features. If you are not yet familiar with CM+, you may skip over this section. If you require more details about the features, please reference the release notes for CM+ 7 and earlier releases.

CM+ 6 was a significant advance in CM technology. It introduced Test Suite Management, permitted flexible history insertion into source files and added dozens of additional CM/ALM features. At the same time, it added numerous ease-of-use capabilities, including the addition of dynamically configurable prompt dialogs, along with data segregation functionality, automated email triggers, and interactive diagram-based state-flow editing. The underlying STS Engine continued to evolve to support more secure messaging, additional backup options, improved data import capabilites and full interoperability between 32 and 64 bit platforms.

CM+ 7 continues to extend the capabilities of Neuma's CM/ALM technology, in several areas: CM/ALM Maturity and Process Support, User Interface Ease-of-Use, Customization Capabilities, and STS Engine/Repository Advances.

In the areas of CM/ALM Maturity and Process, there have been some significant developments. Electronic Signatures (password-protected) have been introduced to help eliminate paperwork and to provide easier traceability. Staging directories allow updates to be staged into globally available common areas before they are checked in to the repository. This simplifies backups, co-development and peer reviews. Code searches, across any subset of the current context, or across revisions of a file or set of files, can be performed directly from the CM+ interface. Color highlighting has been added to delta reports and to state-based data, such as gantt charts, state flows, data pane displays and graphs, so that a quick glance can be used to get a general status update. CMII process support has been added, along with several supporting generic enhancements available to non-CMII process customers.

The user interface has gone through a significant upgrade to improve ease of use. The availability of more advanced user interfaces for Unix/Linux systems brings them in line with Windows clients. The introduction of Interactive, Dynamic Dashboards and Work Stations allow for simpler role execution, whether monitoring project status across multiple releases of your products, or performing peer reviews of a specified set of updates. The introduction of HTML support along with Tabbed Panels (or Tabs), allows for single-click role-based availability of reports, and summaries, as well as easy, customized, process guidance navigation. Menus have been reworked to make use of new customization capabiities which reduce key clicks and menu searches. The capability has been added to filter the data displayed in the data pane. All of these customizations can be further enhanced for the specific needs of the customer. Greater flexibility in the layout of graphs and charts, as well as forms and dialog panels, allow for more efficient use of screen real estate and better information access.

Customization Capabilities have been improved in a number of areas. The introduction of GUI Components simplifies panel and dashboard specifications, and allow for re-use of components in a consistent manner. User-defined buttons are easily added to forms, display panels, text windows, tabs, etc, allowing actions to be located on the information panels and dashboards rather than being hidden in a menu or in a different area of the application. Search term highlighting may be specified for various text displays, including source search functions, and text popup menus may be defined to use the data of a text line to feed a follow-on action. Data browsers may be configured in a number of ways to support partitioning of information or viewing information in collapsable tree formats. Widget type, sizing and layout capabilities are greatly expanded to support easy dashboard creation. Complex data values, such as records or tables, can be displayed as a collection of widgets rather than as a single text entry.

STS Engine and repository advances have also continued. Functional syntax can now be interchanged with command syntax for macro usage, especially where nested macro calls might otherwise cause confusion. The introduction of virtual list fields allows for the automatic inclusion of backward traceability links, reverse tree traversal, etc., while providing a performance improvement when dealing with very large data sets. Improvements to the data query language include easier access to parent records, and additional operators to enhance query, including query of workspace content and code content. CM+ MultiSite has been extended to

allow for improved monitoring and to provide selective file distribution so that sensitive data does not have to be physically distributed to all sites.

Along with dozens of other improvements, CM+ 7 has also addressed many outstanding issues and preferences. The user interface will continue to be streamlined throughout the release cycle to allow easier customization, greater ease-of-use, and additional functionality. Along with functionality provided in earlier releases, this complement of features makes CM+ 7 the only 4th Generation CM/ALM tool commercially available. That implies lower operating costs, low support requirements, better ease-of-use and an overall more complete ALM offering.

CM+ Overview

CM+ is an extensive product offering, which, along with the underlying STS Engine and its Next Generation Hybrid Repository, can be used to manage your Application Lifecycle. CM+ is delivered with a default out-of-the-box process and configuration, but is easily modified to support your own process requirements.

From day one, CM+ was designed to be easy to operate, with near-zero administration. It was also designed to be easily customized to fit the most demanding of engineering and business management requirements. With prior insight as to the difficulties of integrating separate tools, on different upgrade timelines, CM+ was designed as an extensible ALM environment, so that all of the underlying functions could share the same repository, the same user interface, the same process engine, the same multiple site operation capabilities, and even the same administration. The result is a seamlessly integrated tool suite, with rapid, point-and-click navigation, mission-critical reliability and near-zero administration.

CM+ evolved from a background of over a decade of experience in high-performance real-time repositories supporting large communications systems, along with a similar background of supporting hundreds to thousands of users in a collaborative design effort on the computer technology of the 1970s and 1980s. With prior exposure to change-based and stream-based CM, and hardware resource restrictions, CM+ was designed, starting in the late 1980s, from the ground up to support 1000's of concurrent users in a change-package based system. The comprehensive, yet simple, CM model supported the appropriate set of management objects, including build definitions, baseline definitions, change packages (a.k.a. Updates), promotion levels, development streams, branches, and related ALM concepts including tasks/activities/projects, problem reports, and organization charts.

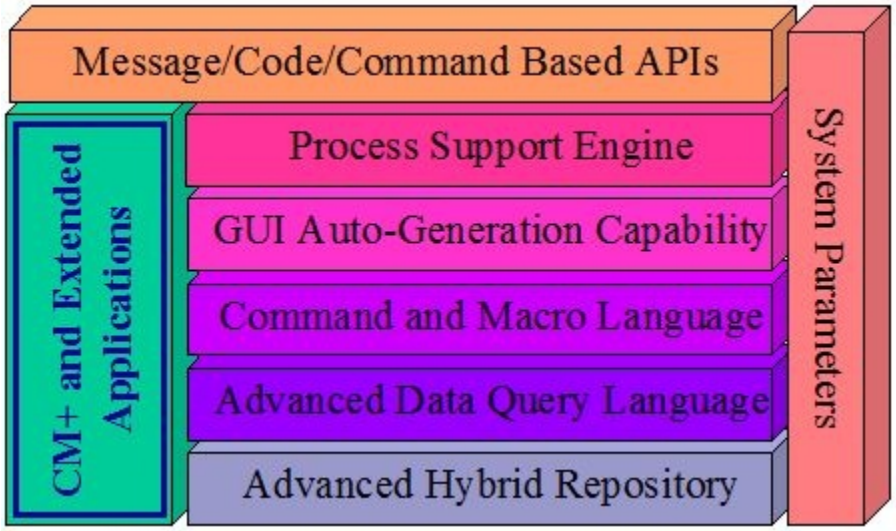
It is this extensive experience base, and exceptional software and customization architectures, that allowed CM+ to first emerge (under the label STS) in the late 1980s. With almost a quarter century of evolution, in the midst of significant underlying technology change, Neuma's CM+ Enterprise is far and away the most capable, most cost effective, and easiest to use and to customize CM/ALM tool suite available.

CM+ was designed to a target of four hours downtime over 40 years, a target well beyond that of the supporting platforms. It was designed to be easy to upgrade, without loss of service, and to be fail-safe against disasters, whether natural or through sudden or subtle sabotage. It was designed to be responsive to every query, independent of the number of users. The smart client technology of the underlying STS Engine ensures that a single server per site is sufficient to support hundreds to thousands of users at that site. And it was designed to be fully traceable so that any change to any data is easily traceable to the person, date and time of occurrence.

CM+ Enterprise may be used out-of-the-box as a rapidly deployed CM/ALM solution, or may be used by a customer or consultant to provide customized high-end ALM solutions for an organization. The ability to load in multiple baselines from a previous system in rapid succession, as well as the related requirement, request, task and problem report data, allows a transition from your current tool in a few days. The short course structure, typically a day or less for end users, two days for administrators and CM managers, supports a rapid deploy-and-go scenario.

User-interfaces and process specifications are role-based, allowing you to ensure each user has the right set of menus, functions, reports and to-do lists. A process-driven capability helps ensure that your procedures are being properly executed. Access to properly configured to-do lists eliminate the need for email notifications and mail box clutter, moving your database from personal mail boxes into a central repository.

As you refine your processes, you may refine your CM+ customizations. You may also maximize the level of automation of those processes, eliminating sources of human error.



The various layers of the CM+ application, including the STS Engine, are shown here. Many systems are built with these types of components. However, only the STS Engine integrates all of these functions in an orthogonal fashion so that, for example, the GUI Auto-Generation Capability can use the Command and Macro Language, as well as the Data Query Language to allow intelligent, yet complex dashboards to be created in a handful of lines of code, or through a point and click user interface.

It is precisely this advantage that gives CM+ its longevity, easy customization capability, and extensive feature set, unrivalled in both the CM/ALM industry and the Database industry. And because the CM+ MultiSite capability is built into the repository, all applications benefit from the unprecedented level of reliability and global access.

CM+ Library Organization and ALM Process

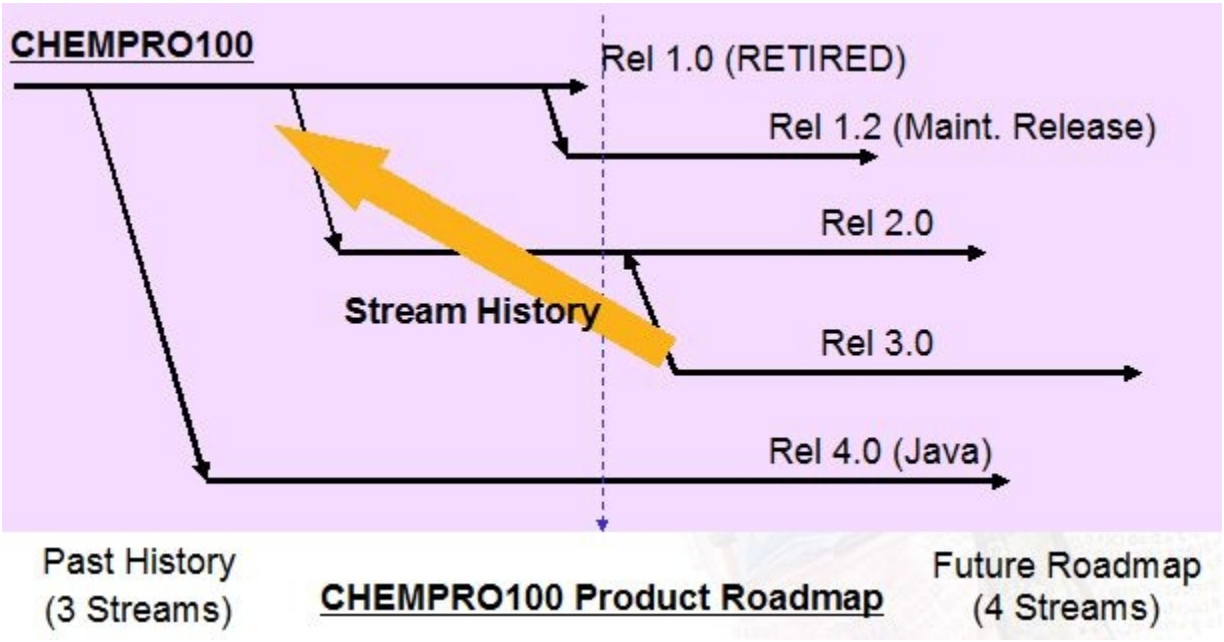
A CM+ Library is an entire CM+ Repository along with the set of customization files which help to give a project it's specific user interface. Each library contains all the data pertinent to the project. This includes documentation, source code, test cases, requirements, problem reports, planning data, and so forth. The out-of-the-box configuration of CM+ is meant to hold most Application Lifecycle Management (ALM) data. It may be extended to meet the additional requirements of any product development team.

The CM+ Repository is part of the STS Engine which provides not only a Hybrid Database capability (i.e. relational data, revisioned data, hierarchical data, large objects, etc.), but also a Process Engine, Command and Macro Language, and a GUI Generation capability.

Each CM+ Library can be used to manage one or more Products. Typically, independent development teams have their own CM+ libraries, while products which are administered by a single team or which have some level of data sharing or common planning are grouped into a single library. Views of the library contents can be easily partitioned by product, so that any particular user can see only the products for which that user has been assigned access. By default users can see all products but typically work within the context of a single product at a time.

A Product is a container for all of the data associated with a product's application life cycle. Products may be hierarchically arranged, both for navigation and relationship purposes. When navigating data within the CM+ Library, it is normal to establish a Product context so that the data shown in queries, reports, source trees, to-do lists etc. reflect the product (and its sub-products).

Each product has a "root node" which denotes the root directory for all product source code. Source code is arranged hierarchically below this root node and typically will be grouped according to the various target deliverables to be generated from the code. There may or may not be specific conventions used for grouping source code in a given product team.



Product Road Map, Releases and Streams

A product evolves along successive, or even parallel, development streams. This evolution is identified as the

Product Road Map, which pertains both to the past (already released product) and to the future (planned releases). Most data is tagged with a release stream to which it pertains. In particular, file branches, tasks/activities/change requests, and build records are all tagged with the stream, as well as the product. The stream is a common component of a context setting that helps to focus the view of data to a working context, as most users are focused on a particular release stream of a particular product. A release is a culmination of the work and artifacts produced within a stream.

One Library or Many

One of the more difficult decisions, when starting to use CM+, is the allocation of existing code to libraries and products. Because of the various business and engineering architectures, there is not always a clear choice. However a few different views may help in this decision making.

First of all, think of the potential domain of your product. If you were to sell off your project/product technology, what information would be packaged together? This is a good starting point for deciding what should go into a single library. If the company doing the buying is going to need the information/assets, they probably all belong together. If your company produces an Operating System (OS) on top of which an Office Suite sits, it's quite conceivable that the Office Suite could be sold off without the OS. As such the Office Suite would likely form one or more products within a single library, while the OS would have its own library. Another clue: normally, when work is aligned to a common delivery schedule, the components of that work belong in a single library. So if the Office Suite has several “product” components, such as a Word Processor and a Spreadsheet, but are being packaged as a Suite, there would be a tendency to put these product components together into a single library..

Unless we're talking about very large programs (10s of millions of lines of code each), performance is not likely to be a factor in this decision. What tends to be a more compelling factor is that the programs share components or have similar behaviors which make them candidates for a lot of code sharing. As well, if the various programs of the suite are all designed using the same processes, primarily, than they would likely benefit from being together in the same library. However, sometimes the shared code is itself managed in one library, while multiple layered software components which share this code are each in separate libraries.

Generally, with CM+, it's easier to split things up later (assuming the really are independent) than it is to combine them later. It helps to have an overall architectural view – from various perspectives: run-time, design, training, documentation, process, schedules, etc. A suite will naturally want to fit into a single library because of the commonality within these architectural views. Having them together does not preclude them from being separate products or having separate schedules. So if you're going to err, err on the side of putting them together, and split them up later if necessary.

Traceability navigation is also much easier within a library, rather than across libraries. So if there is a lot of cross-product traceability, the tendency to use a single library might be the right one.

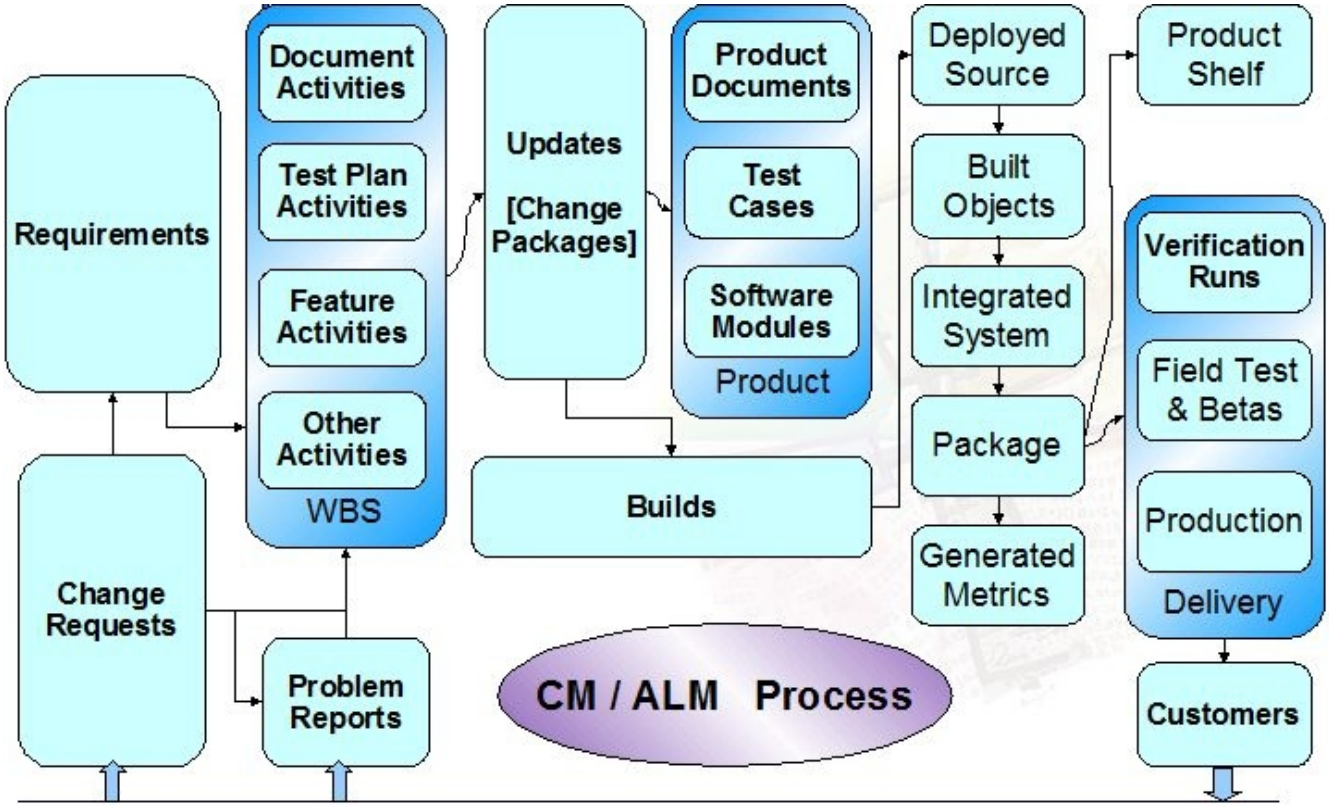
For projects that have significantly differing processes, they will normally be a tendency to put them into separate libraries, assuming that the separate processes is a desired attribute. Different processes will often mean different types of data, different roles, and a need to control things in different ways. First make sure that the projects really shouldn't be sharing the same processes, and if not, err on the side of having them in separate libraries.

What's a Project

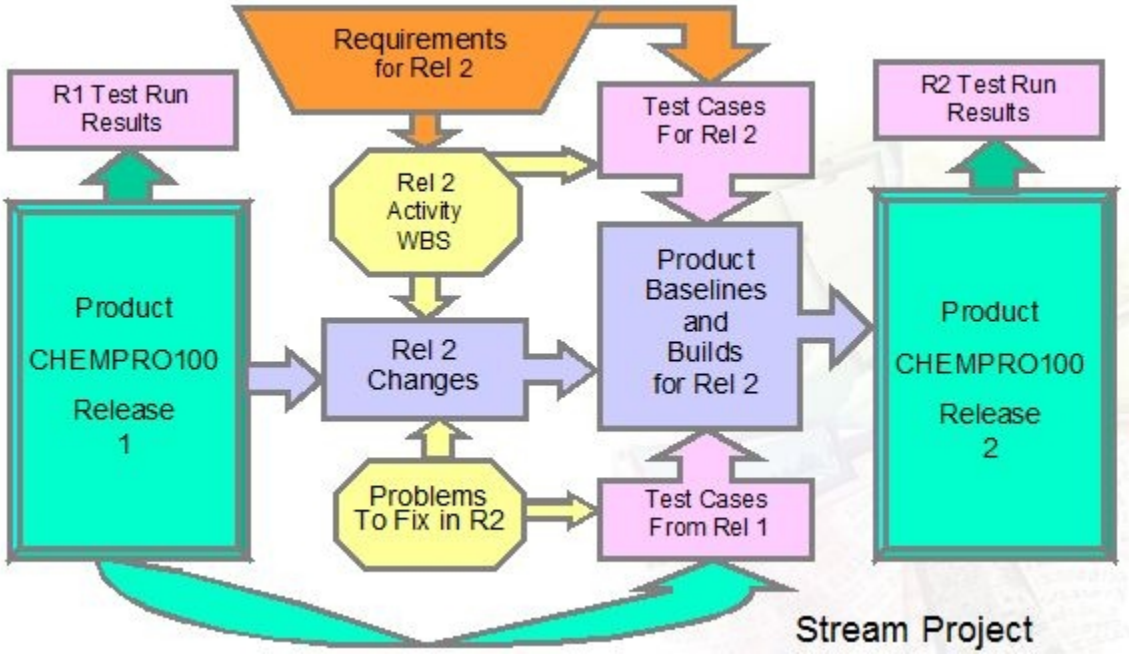
Note that the word project, here, is used loosely. More formally, CM+ uses the term project as the “root” node of a Work Breakdown Structure (WBS), which defines a set of tasks to be completed in order to complete the project. This differs from the use of the term “project” by some IDEs (Interactive Development Environments), and falls more in line with the traditional Project Management usage. Each project belongs to a product (i.e. works towards definition of the product), as we'll see later.

The CM/ALM Process

The Application Lifecycle Management (ALM) functions for a product work together to take a product through a set of release iterations, with Requirements spawning Activities which result in Design, Implementation, Documentation and Test Case construction. [See CM/ALM Process diagram.] The result is a series of Baselines and Builds which move successively closer to the target requirements and product quality, until verification activities identify that the product has reached an acceptable production quality and can be shipped to the customer.



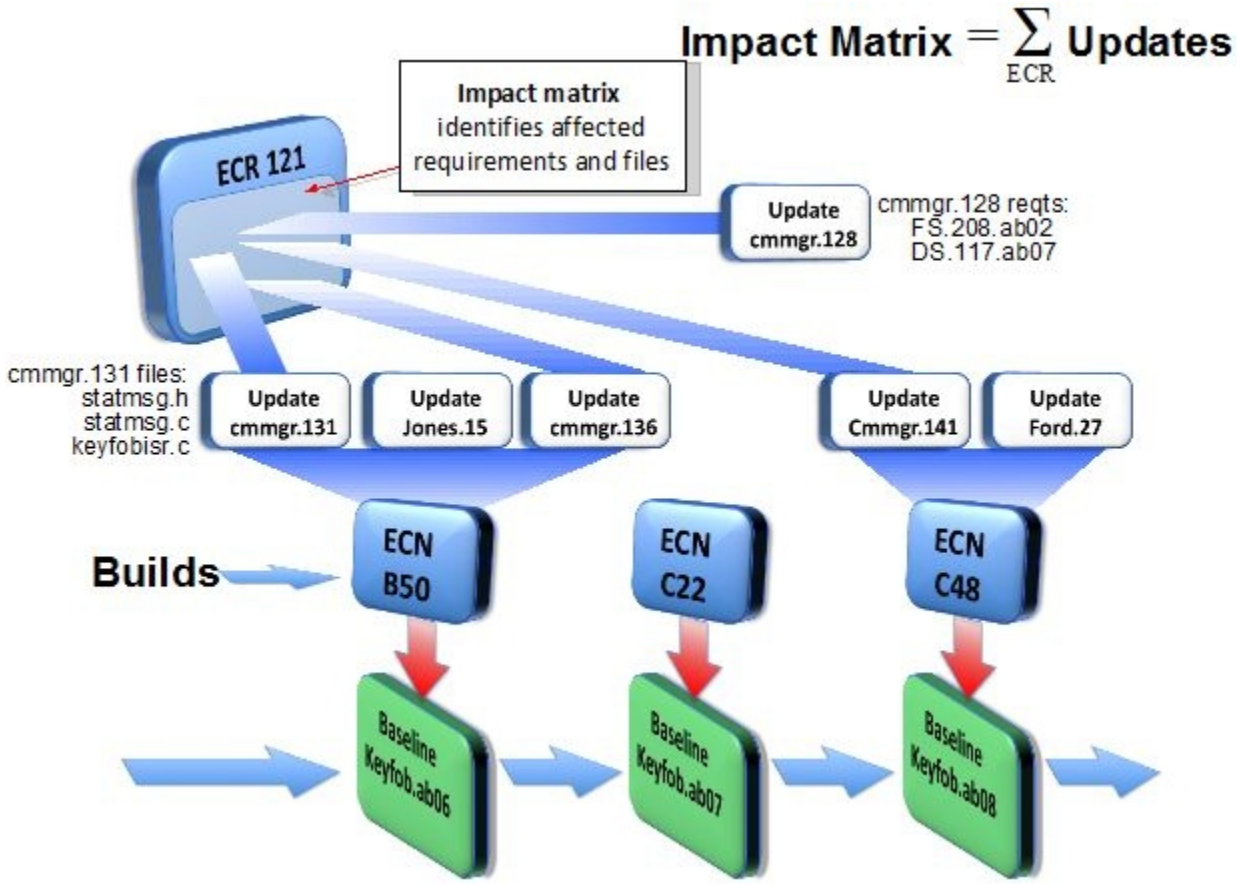
Typically, a top-level Activity, known as a Project, is used to define the tasks/activities required to transform one release of a product into another. This Project Activity, along with its Work Breakdown Structure (WBS), form a tree structure indicating the work to be done as part of the project. In some cases the work is fully pre-defined prior to design and implementation. But more commonly, the WBS grows through the development stream as complex activities are broken down hierarchically into a set of less complex activities, and as activity implementation spawns additional activities (e.g. a design activity spawning documentation or test case activities). In some agile development methodologies, the project is simply defined by all of the activities identified for a particular product stream, without a formal Project activity or the formal arrangement into a WBS tree structure.



The first level of project activities are usually generated from a set of requirements for the project, typically for a new release. The mapping of the new (i.e. new for this release) customer/market requirements onto product functional specification activities (a.k.a. product requirements) generates this set of activities, typically through a requirements analysis phase. Requirements are themselves generated as a result of customer requests, engineering input, market forces and a number of other stimuli.

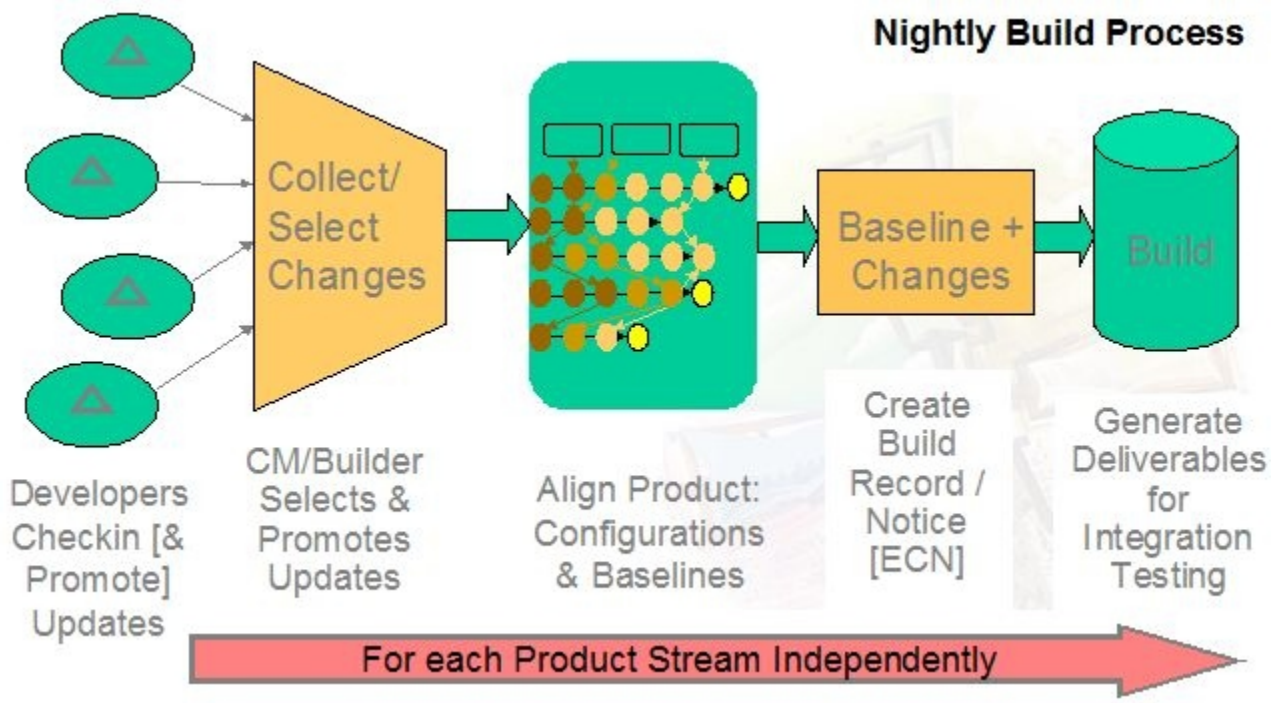
In some projects, functional activities are known as features or feature activities, while in others, as change requests, and still in others as tasks. Sometimes these include problem reports, but often it is the case that problem reports, defects with respect to the established requirements, are tracked separately. It is common in some organizations to refer to all change requests as CRs/ECRs ([Engineering] Change Requests) or CR Activities and to tag each CR Activity as a feature or a problem. This may be customized in CM+, but most customizations have an Activity WBS capability, with Activities labelled as CR/Feature Activities, Test Activities, etc. In any event, the change request will be implemented in one or more change packages known in CM+ as Updates. Software Updates are used to track changes to source code and documentation, while Requirements Updates can be used for the change control of Requirements.

Each Update is tied to one (or more) CR/Feature Activities and/or Problems, and collects all the files (or possibly all the requirements) which are part of the logical change into one package, with context, integration and unit test information attached. The requirements and files contained in the set of updates tied to a given CR Activity identify the Impact Matrix for the CR. Updates are collected by the Build team to produce successive builds for a product.

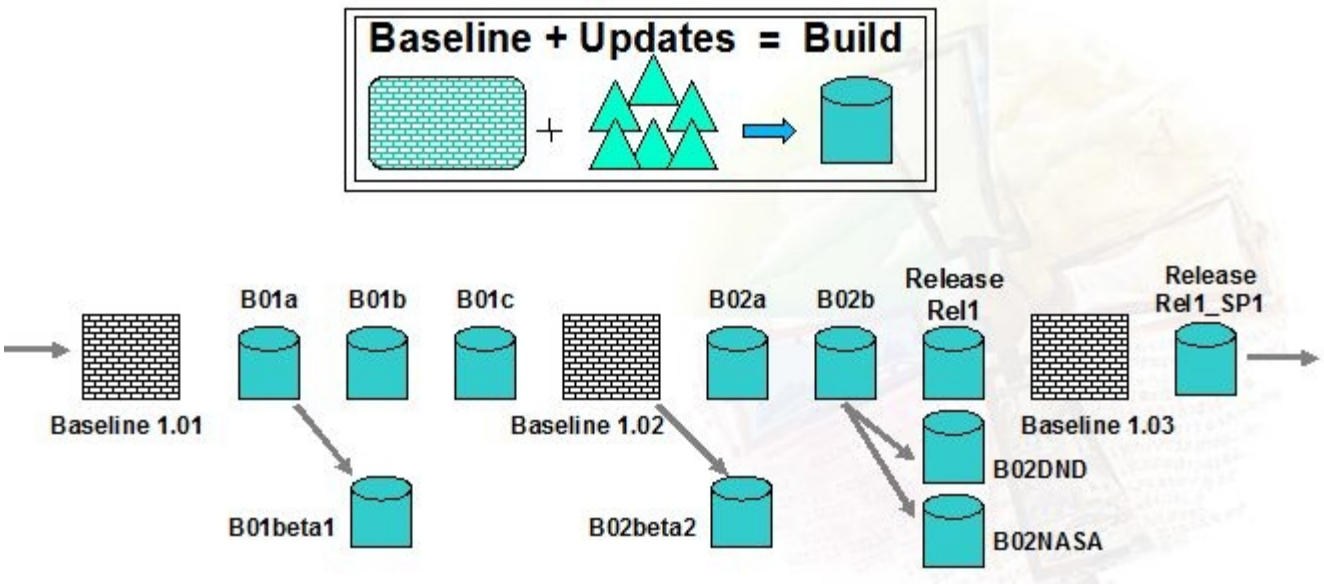


Within each Product, within each development Stream, on a regular basis and typically nightly, newly created updates are rolled into the next integration build cycle. This may be an automated process, or it may be a manual selection process. In some shops, early on in a stream's development, all updates that have been marked as “ready” are rolled into the "nightly build", while later on, as the release date approaches, updates are hand-picked. Perhaps even better, update creation can be restricted to approved problem reports and features, with approval becoming more strict as the release date nears.

Peer code reviews and unit testing are essential to successful iterative build processes. As well, unit testing (where the unit is the Update), and peer reviews of the test results, along with demos in some cases, will help ensure a higher quality release. The general process, from a build and integration perspective is shown in the following diagram.



The product configurations are aligned to reflect the selected updates. Every so often, these configurations are frozen to establish a new Baseline against which to measure change. Baselines are used to establish reference points, typically reflecting significant milestones or build maturity. Baselines are normally established from the product root node downward, but any directory underneath the product root signifies a sub-baseline and can be frozen independent of the full baseline. It is generally easier to manage a single baseline for the entire product, and to produce deliverables as subsets of this baseline. For this reason, Build records may be produced directly from the baseline, or from some subset or variant of the baseline, typically by specifying options and Updates to be applied to the Baseline to produce the Build.



Based on schedules, feature completion, resource allocation and build stability, a subset of builds are promoted to the verification testing level, where new feature testing, as well as regression testing, is used to assess the quality of the build, and to document problems found. Test cases can be arranged hierarchically into functional groupings and into complete test suites. Each test case is tied back to the original requirements so that full test case coverage may be ensured. The test cases stored in CM+ may be automated test cases

(e.g. batch scripts) or manual test cases (e.g. manual scripts).

Test Runs record the results of running a set of test cases against a given Build, typically by a single tester. Test Runs may be arranged in a hierarchy showing the various test runs that were part of a verification effort. This allows easy identification of which Requirements have failed in which Builds.

CM/ALM Functional Overview

The CM/ALM cycle starts with requirements and follows through to customer delivery. Customer requests result in the cycle repeating itself, sometimes in shorter cycles, sometimes in major release cycles.

Requirements

Product development proceeds from a set of requirements. Sometimes these are very informal, such as: "We need a product to compete with ACME's WIDGET". Such a requirement would naturally decompose into a the set of features the WIDGET has, along with an additional set to give a competitive edge. More often, hopefully, requirements are defined by a set of system and/or application experts after evaluating business decisions and looking at a number of alternative directions.

Either way, there is a need to capture requirements for a number of reasons:
(1) To identify what is to be built, as in the initial baseline of requirements.
(2) To allow the generation of test cases which will prove the requirements to be satisfied
(3) To measure changes in direction, normally given as a new baseline of requirements.

If we look at the product development cycle, we will see that most products are designed and released over a number of release cycles. The requirements are the marching orders for the release. The release activity, which forms a development stream for the release, transforms a product reflecting an initial set of requirements into one reflecting a new set of requirements.

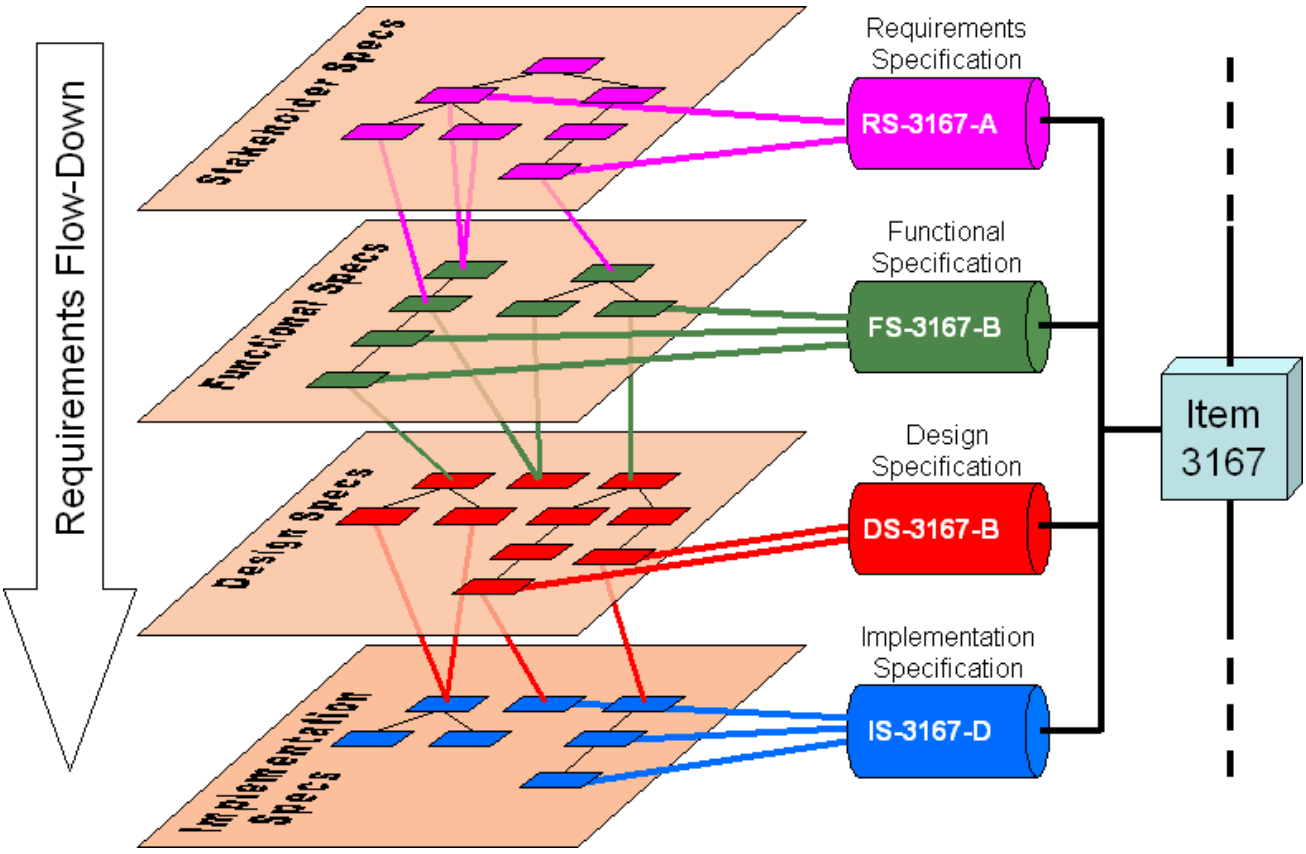
Each development cycle involves looking at outstanding requests and establishing release-specific requirements, looking at these requirements and outstanding problem reports (a.k.a. defects, issues) to establish a set of engineering goals, often in the form of software requirements, and then establishing a revised set of documentation, source code and test cases which accomplish these goals. Builds are created and test runs are used to ascertain conformance to the requirements, with incremental changes and builds used to eliminate any non-conformance. When requirements have been verified, the deliverables are packaged and delivered to the customer. Customer feedback then results in the start of the next cycle.

Requirements are generally gathered functionally into groups, and further decomposed into more detailed requirements. The entire set of requirements, usually referred to as customer requirements, provide a baseline for the product development team.

From this point, there are two distinct sets of terminology and procedures used. They accomplish essentially the same task, which is to identify developer implementation tasks.

In the first case, we have customer requirements transformed, or allocated, into product requirements. Instead of detailing what the customer requires, the product requirements specify how the product will work, from a black box perspective. Such a specification must satisfy all of the customer requirements, and depending on the customer or market, may support a much wider set of requirements.

The product requirements, in aggregate form, are often referred to as the product specification. Once the product specification is completed, that is, the product requirements, a Design phase is used to allocate product requirements to software requirements. In each of these allocations, project management must identify and track the tasks needed to perform the specification, design and then the implementation.



In the second case, the product team allocates customer requirements onto feature specification activities. These activities (or tasks) are used both to document the product specification (i.e. Product requirements) and to track the activity. Once specifications are done for each feature, it can be decomposed into design activities, which both specify the design and track its progress. Depending on the architecture, the design activities may be the result of individual specifications, or may flow from an overall mapping of features onto design, whereby certain design tasks are used to support multiple features.

In either case, note the difference between decomposition of requirements into more detailed requirements, versus allocation of requirements from a customer to a product domain or from a product to a design domain. In some cases, especially some Agile shops, the customer is satisfied to have limited decomposition until the product domain is specified. Decomposition gives more detail on what a requirement is. Allocation gives more detail on how the requirement is going to be met.

In general, there two different requirements processes are used in industry. One transforms customer requirements into product requirements and then into software requirements. The other uses the term requirements for customer requirements and transforms these into (product) feature specification activities/tasks, and then into design tasks, from which software design documentation is produced. CM+ has two different requirement management modules to track requirements in these two different ways.

Because requirements may be modified, they are under revision control. Each requirement baseline can be modified only by creating requirement change packages (i.e. requirement updates) which collect the modifications to multiple requirements related to a change request. New requirements may be added, requirements may be removed, or requirements may be checked out and modified. When complete, the entire requirement update is checked in.

Typically, there are revisions within a release stream, and there are revisions made which are for a subsequent release stream. Just as for source code, individual requirements that change from one release to another have a new stream branch created for them. In order to change requirements, they are checked out, changed and checked in. By default, requirement checkouts are exclusive, meaning only one person can change a

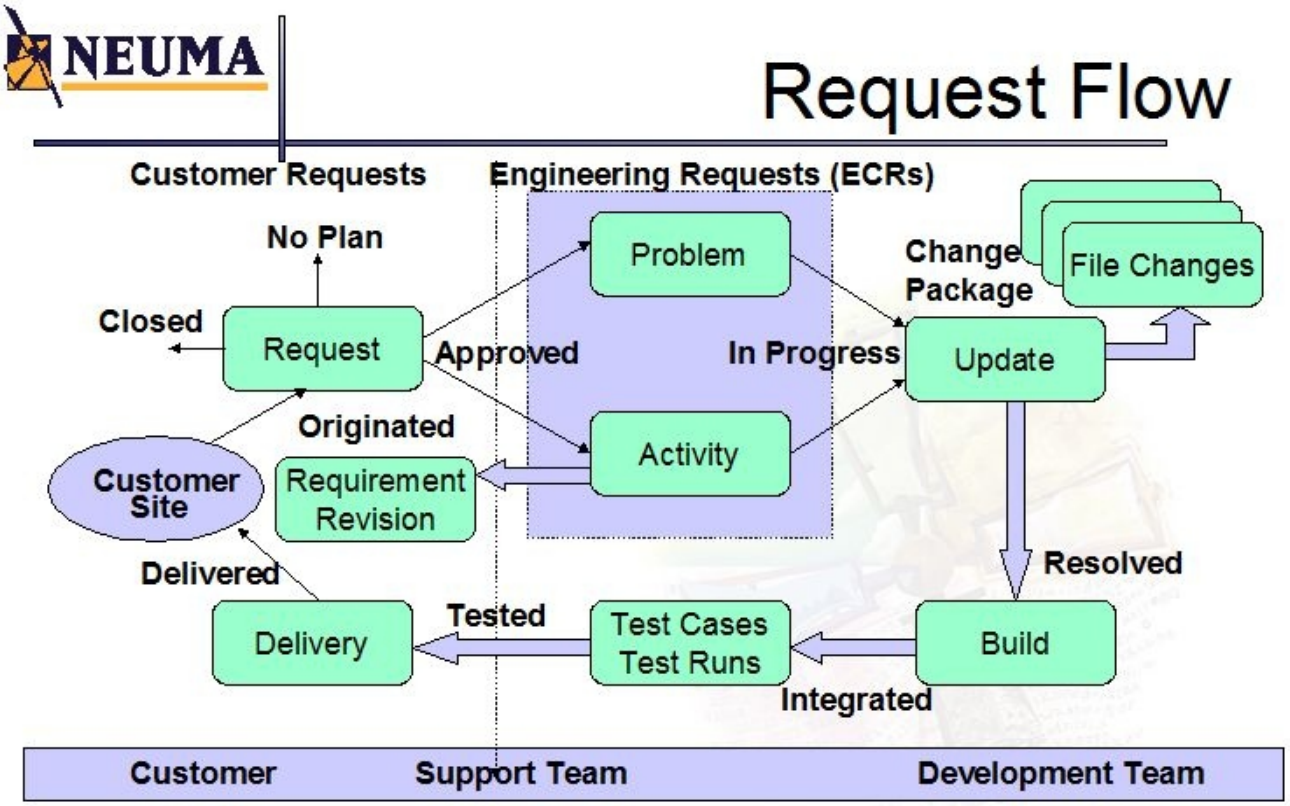
given requirement at a time. Because requirements are generally small (as compared to source code), are administered by a very small group, and are generally easily changed, exclusive checkout for requirements is an effective strategy, even in large shops.

In a fashion similar to that for source code baselines, requirement updates are promoted and aligned into a (soon-to-be baseline) configuration. Other requirement updates may be subsequently promoted and aligned into the configuration until a decision is made to freeze the configuration into a baseline, after which its contents are fixed. Two baselines may be compared to identify the changes between them. This is typically the way a product team will identify changes to its marching orders, either for new release development, or for a mid-course correction within a release development cycle..

Customer Change Requests

A Customer Change Request, or CCR, is a request from a customer or field representative, for a change to the product. The request may be the result of required new functionality, or a deficiency in the product functionality, whether or not the deficiency is in non-conformance with the requirements.

Typically, there will be multiple sources for the change request, primarily from different customers. The same request may be expressed in different ways. The customer is asked to classify the request as a problem or a feature request, and to give a clear specification of the request. Often the specification will make it clear that the problem is an operating, rather than a product problem. Or it may be a request for a feature that already exists in the product. In these cases the help desk will short circuit the response by providing the necessary information that the customer is missing.



Customer requests must be tracked against each customer (or customer rep) and responses (e.g. quarterly) provided to help the customer stay up to date with the status of its requests. The help desk must review each request for clarity and completeness and request additional information when necessary. Once a complete request is received that must be handled by the product team, the request is accepted and placed on the product manager's queue. The product manager decides if the request is a problem or feature and adds a

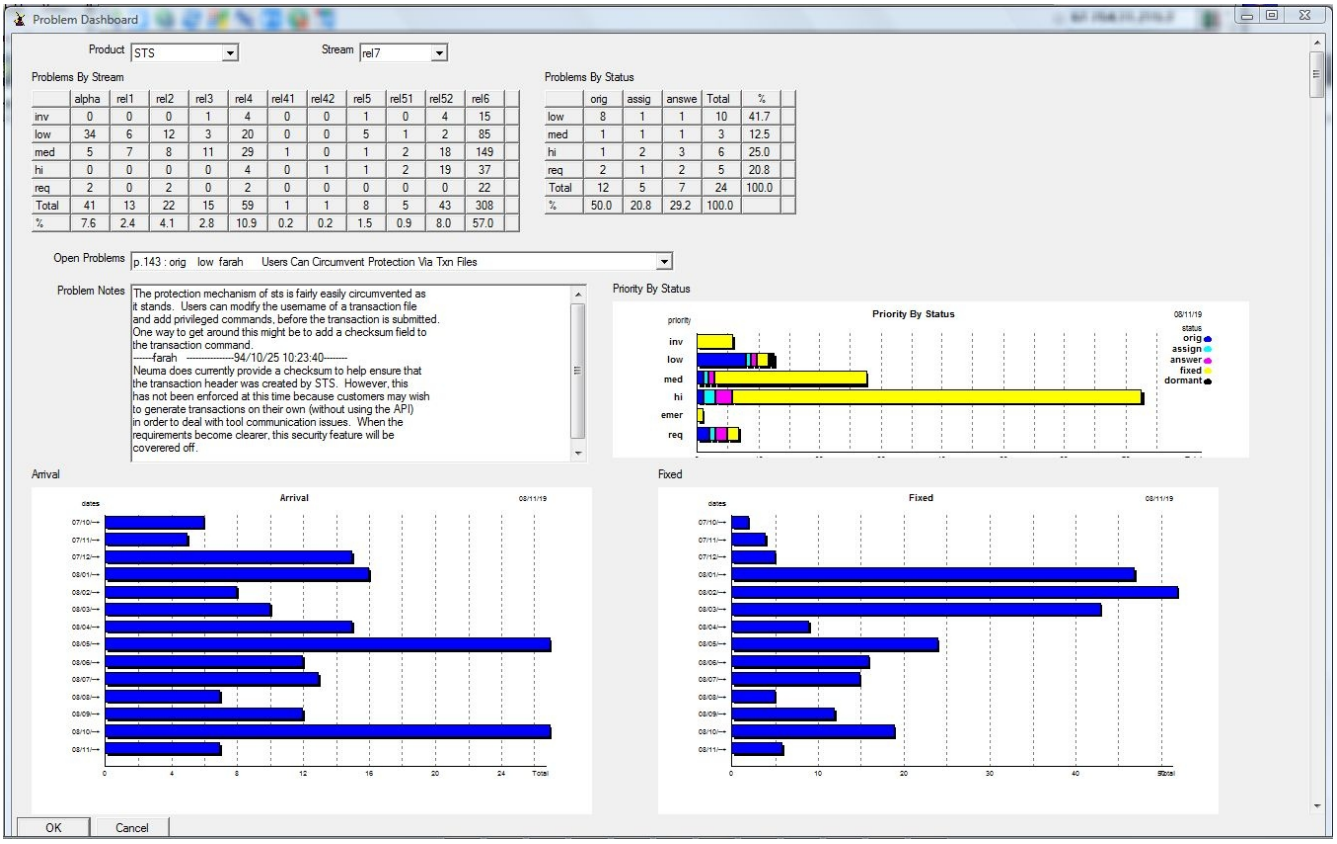
problem report (a.k.a. issue, defect, bug report) or feature activity, accordingly. Some configurations of CM+ may have an "ECR" (Engineering Change Request) object in place of problem reports or feature activities. Typically, this is an ECR activity for which planning and tracking is performed.

Information associated with the problem report or feature activity is private to that object, that is, not generally visible to customers. However, as the status of the object moves forward, a trigger will automatically promote the status of the original CCR so that the progress is reflected there. Once the completed functionality is available for delivery, the status of the CCR reaches its penultimate state, awaiting only customer closure, based on receipt and acceptance of the request implementation.

Problem Reports

Problem reports (also known as bugs, defects and issues) identify a non-conformance between a requirement and the design, implementation or documentation for a product. CM+ provides a comprehensive problem reporting and tracking system. This problem tracking application may be used as a stand-alone system, or it may be fully integrated with your complete ALM application suite.

A problem can be defined as a deficiency in a product or process that represents a deviation from the product requirement or specification. For any problem entered into CM+, the system tracks critical information such as who originated the problem, the status and priority of the problem, the build in which the problem was found, the release stream in which the problem will be fixed, as well as the problem description. As with all CM+ applications, the set of data tracked against a problem report can be fully customized.



A problem report must clearly identify the deficiency, and how to reproduce it. Ideally, the originator can trace the problem back to the requirement which is responsible for identifying the non-conformance. In any case, the problem report is not a request to change a requirement, but a description of how the product does not conform to an existing requirement. As such, a change which fixes a reported problem should not need a modification of the requirements, of the user documentation, nor of the training material. It may require the

specification of a temporary work-around until the problem is resolved.

Problems are generally, though not always, easier to fix than features are to implement. The impact of the fix is often already included in the original design effort. The primary task for addressing problems is repeatability in reproducing the problem, after which the fix is generally rapidly forthcoming.

CM+ Problem Tracking automates problem tracking to:

- permanently record all problem reports in the repository
- avoid duplication of effort
- record work-arounds for problems that won't be fixed right away
- track the current status of each problem and who is responsible for moving the problem to the next state
- measure the quality of products objectively, so the program manager will know when a product is ready to release
- identify and categorize the root causes of problems
- determine the volume of problems and the average time taken to fix them, permitting accurate resource requirements planning for future products and releases

Neuma recommends the use of the following guidelines to help ensure successful problem management.

- Use CM+ to track all problems. There must be a single, reliable source for all problem data.
- Give the entire product team (read) access to all problem information.
- Screen all problems to ensure proper routing of the problem report and proper interpretation of the data fields, especially priority. It is recommended that a Problem Review Board review problem data. The role of a Problem Review Board becomes most important when new releases are in field trial or are being shipped to the initial set of customers.
- Integrate problem tracking with other parts of the development process to maximize its benefits and to help ensure the timeliness and accuracy of the data.
- Establish a clear flow of responsibility for each problem from the point of its origination until it is closed.

Problem priority is often used to reflect a combination of factors including:

- the customer's view
- potential damage that may result
- required response time
- the risk, when fixed, to existing functionality

Neuma recommends that priority be tracked so that the priority simply reflects the time by which a response is required, and that the original (i.e. originator's) priority is also recorded.

The default set of problem priorities for CM+ are:

- none - priority not yet established
- emer - problem needs immediate attention and resolution
- hi - problem must be resolved in next minor or major release
- med - problem should be resolved by next major release
- low - problem can be resoved in product team`s own time line

In CM+, a problem report belongs in the engineering domain and indicates an engineering deficiency. If there is a need to capture customer change requests due to customer-originated problems, it is recommended that a Customer Change Request application be used, which is separate from the problem reporting function. The reasons are many:

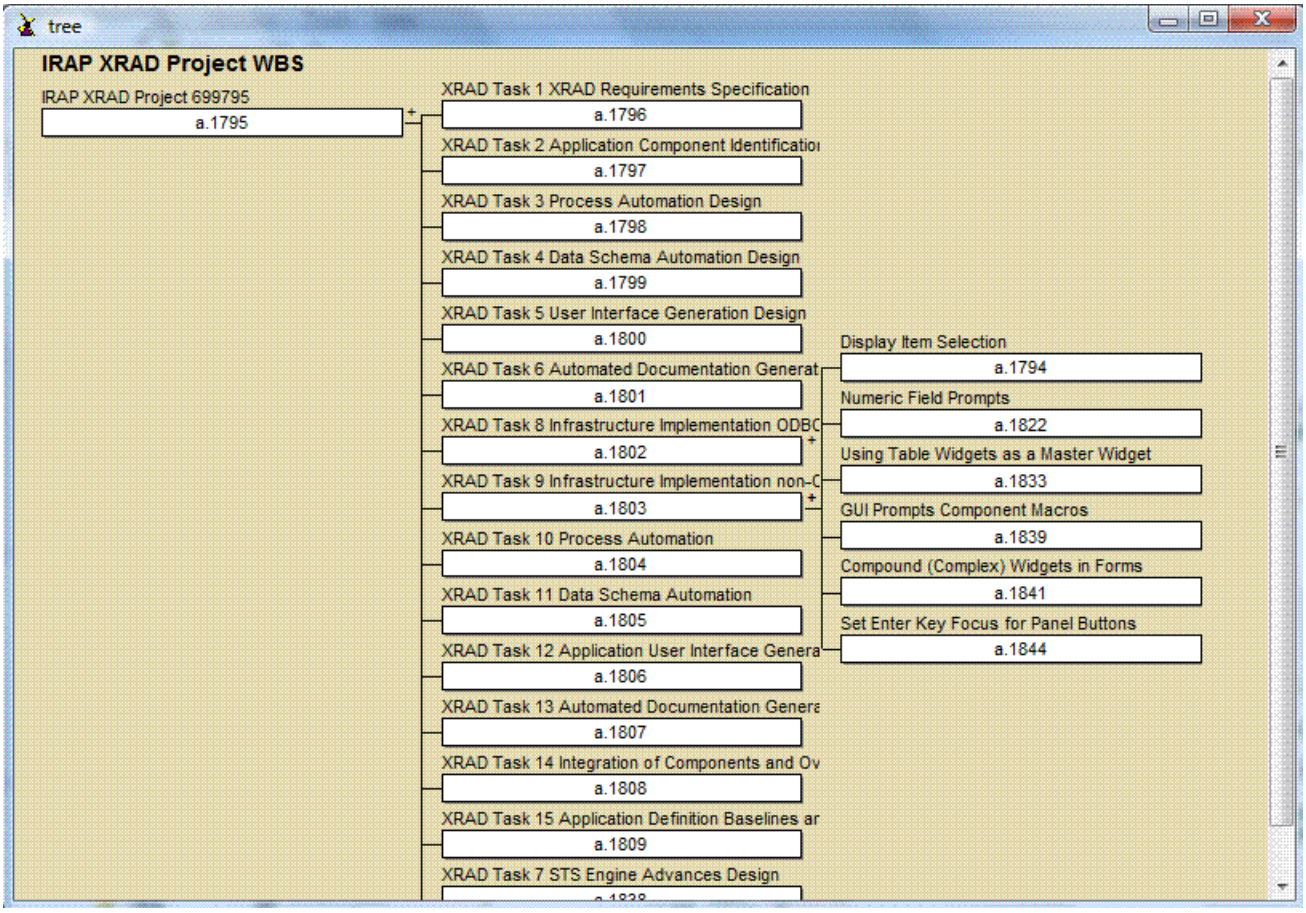
- Multiple customer may report the same problem
- Customers are not always aware of the difference between a problem and a feature request
- Many customer reports can be addressed without changing the product (e.g. plug it in)
- Requests must be tracked against each customer so that feedback can be automate
- The content of the problem report may contain proprietary information, not suitable for the customer

Activities/Tasks/Feature Requests

An activity is a task or a collection of tasks used to help manage project implementation. Activities are generally created for feature requests and for new requirements (which may result from a feature request). Unlike customer requirements, activities are in the domain of the product team and are used to plan implementation and track progress.

Activities may be of various types. There are Feature Activities, which are analogous to "Product Requirements". There are "Design Activities" which are analogous to "Software Requirements". There are also Test Plan activities, Build and Integration Activities, Test Activities, Documentation Activities and any number of other activity types specific to your project. Typically, different types of tasks will have different work flows associated with them.

Activities are sometimes arranged into a tree structure known as a Work Breakdown Structure (or WBS), with the top level activity known as the "Project" activity. The word "task" is somewhat of a synonym for "activity", although, the term task is generally only used for a bottom level activity of a WBS, while activity may be used at any level.

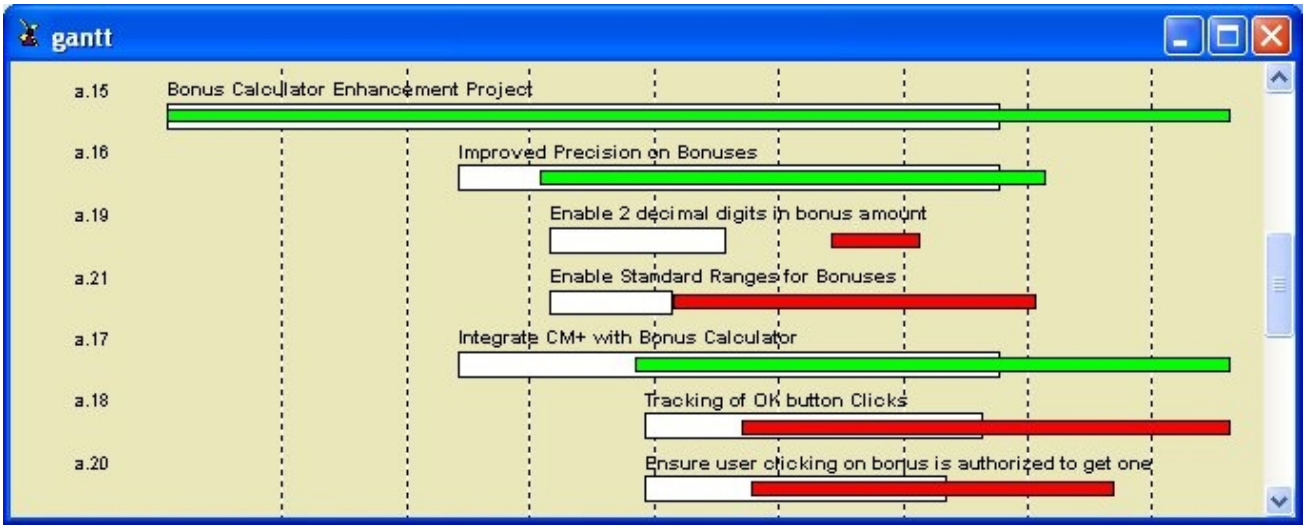


Activities are generally used in one of two ways: (1) a traditional scheduled task, or (2) a prioritized task. Traditional project management assigns tasks and schedules them for implementation. Agile project management typically prioritize tasks, optionally assigning them for implementation, or else leaving them in a priority queue for selection by the design team members. In the former case, resource planning is more essential.

Activities are decomposed into smaller tasks, both to ensure reasonable sizing and to ensure task assignment can ultimately reside with a single individual. In a traditional model, an activity might have just a start and end

date planned and tracked, or might have multiple checkpoints, such as start, designed, coded, reviewed, tested. In the traditional model, there are many CM+ capabilities that may be used to help plan and track progress. These include progress charts, Gantt charts and tabular summaries, typically arranged in a single dashboard with product and stream selectors.

The out-of-the-box configuration supports an Agile model, whereby activities, or tasks, are prioritized and assigned, with the assumption that they are sized for implementation within a single Agile cycle, or perhaps two or three in some cases. This is a feature-driven, priority model. Tasks are generally assigned to developers who can most effectively complete them in the required time frame. At the end of each cycle, the project team ensures that, at a minimum, there are sufficient tasks for each team member to last throughout the next cycle.



Generally, activities are handled distinctly from problem reports. One reason is that the appearance of a problem report can be quite unpredictable. Another reason is simply that the number of problem reports is generally much larger than the number of activities, so that, at least in a traditional model, it is a significant burden to try to plan the implementation of each fix. Instead, a general metric is used to allocate sufficient resources for addressing the appropriate portion of the problem backlog. However, feature activities differ in a much more fundamental way from problem reports. A feature requires a specification, modification to the requirements baseline, a new set of test cases, and potentially a change to documentation and training material. A problem, on the other hand, is a defect which when fixed will allow the existing specification, requirements, training material and documentation to conform to the desired operation of the product.

Software Tree/System Structure: Directories and Files

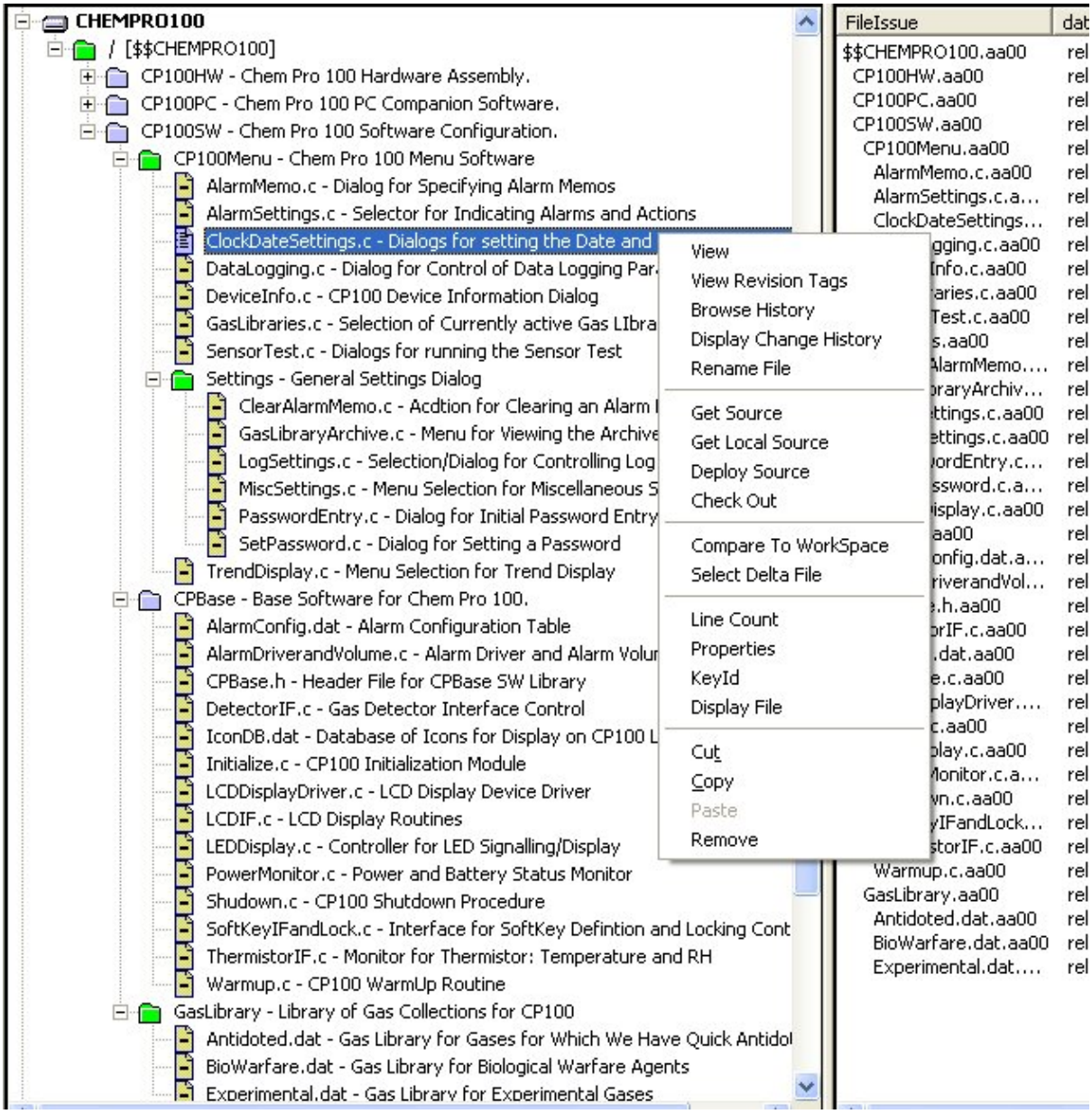
CM+ allows you to organize source code, requirements and other objects into object hierarchies, or trees. The tree structure allows you to organize objects functionally, by design or otherwise. The most common tree structure in a CM tool, including CM+, is the source tree. It is a directory/file hierarchy in which source code revisions are organized, much like in any modern file system.

Although both the contents of a file system and of a source tree change over time, the difference with CM tools, including CM+, is that it is necessary to remember how a source tree has changed over time. In fact the source tree may be at many different states at the same time: release 1 version of the source tree, the release 2 version and so forth. When multiple releases are under construction or otherwise still being supported, the source tree may change in each of those releases, typically with some level of isolation of changes between releases.

In some cases, such as fixing problems, a change to one release may be desirable in another release. In CM+, by default, when a change affects a revision object in one release that is shared in a derived release (as

indicated by the Product's Stream History), the change is automatically propagated to the derived release. There are options to prevent this from happening, but generally, it is a good choice to have this as the default behavior.

CM+ allows you, initially, to build your release structure by bulk loading in a series of "baselines". By specifying the appropriate release information on each baseline bulk load, CM+ can determine the stream history of the product, and the branch history for the files within the product. Where it is unable to do so, it will ask the user for assistance so that it can accurately trace a later revision of a file (that perhaps has been moved to a different directory) to its earlier revisions. Otherwise, although the baseline can be accurately captured, it might not be possible to trace all objects from one baseline to another. So, for a file that has moved from one directory to another, without user input it might appear to have been deleted from the one directory with a new file (by the same name) added to another directory.



Context views are used to view source trees, and other context-sensitive data, within the context of a specific release. A context view provides a file-system like view, but with the understanding that each item in the view actually represents a specific revision of that item, as determined by the context parameters.

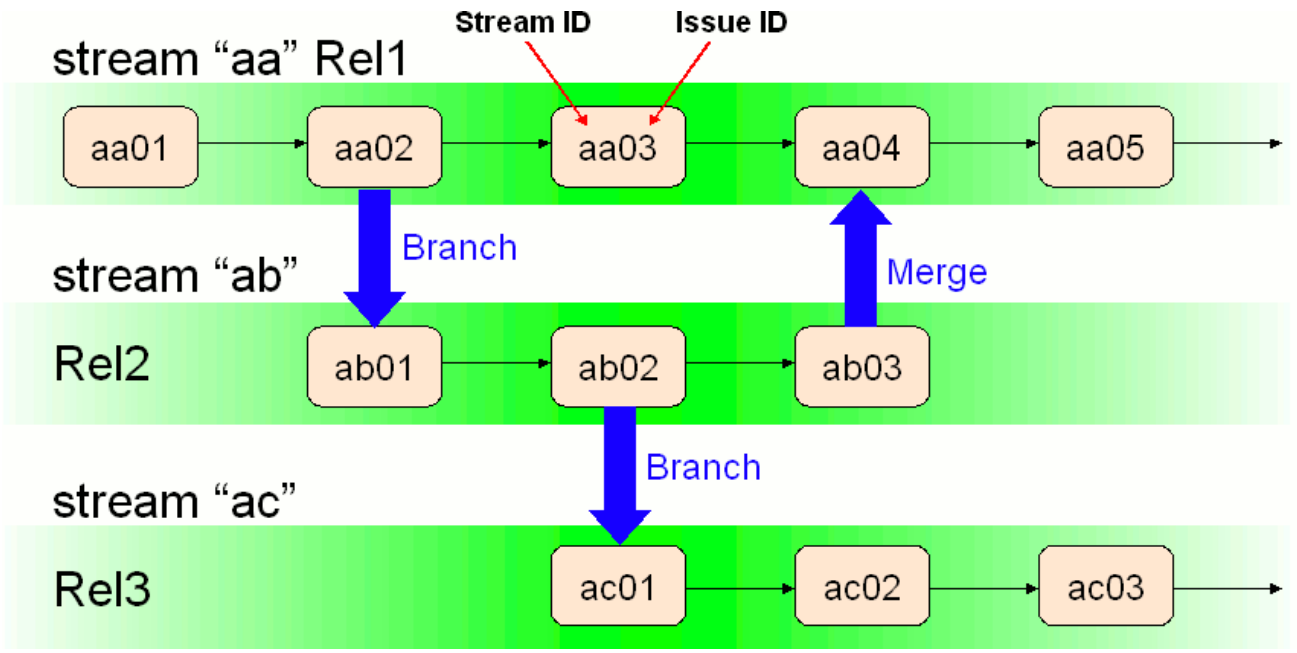
All changes to a source tree structure must be performed as part of a CM+ Update (a.k.a. change package). The update records all changes to the tree structure so that as it is applied to a baseline, a build or a context, those changes appear.

Version Control, Streams and Source History

Any SCM tool must allow revisions to source code and must be able to track such revisions. This capability is generically known as version control. CM+ avoids the use of the term “version” because it is an overloaded term, preferring instead the term “revision”, one used in hardware revision control. CM+ allows revisions to be made along an existing branch, or by creating a new branch. When a new branch is created, its "created from" point is recorded as part of the branch data. Such information is never derived from the branch naming.

In the default and highly recommended configuration, CM+ uses branches for parallel support of an object. In most cases, this corresponds to major and possibly minor releases. The branch is named sequentially in its creation order, and is labelled with the release to which it belongs. Revisions are numbered within a branch. The branch/revision identification scheme is one that is easily customized in CM+. By default, branches are identified with a double letter (aa, ab, ac,...az, ba, bb, ...) and revisions within a branch with a double (or triple if necessary) number (00, 01,..., 99, 100, 101,...). This gives a revision identification suffix of the form: aa00.

A branch derived from another branch is given a name independent of it's ancestry, from the next available branch name. When "branch tracking" is enabled, the branch name used is the same name used for branching of the product root. In this way, all release 5 objects of a product hierarchy would typically share the same branch name. In general, branch names are very unimportant to the user. Instead, it is the stream label that matters. Streams are often identified by their ultimate release names: rel1, rel2, rel3, rel3_2, rel4, etc. Stream names are chosen by the project office, and, unlike branch names, generally reflect the marketing designation. Stream names are attached not only to source code, but to requirements, engineering change requests/activities, problems, builds, etc. They are a key identifier and an important designation for establishing a working context.

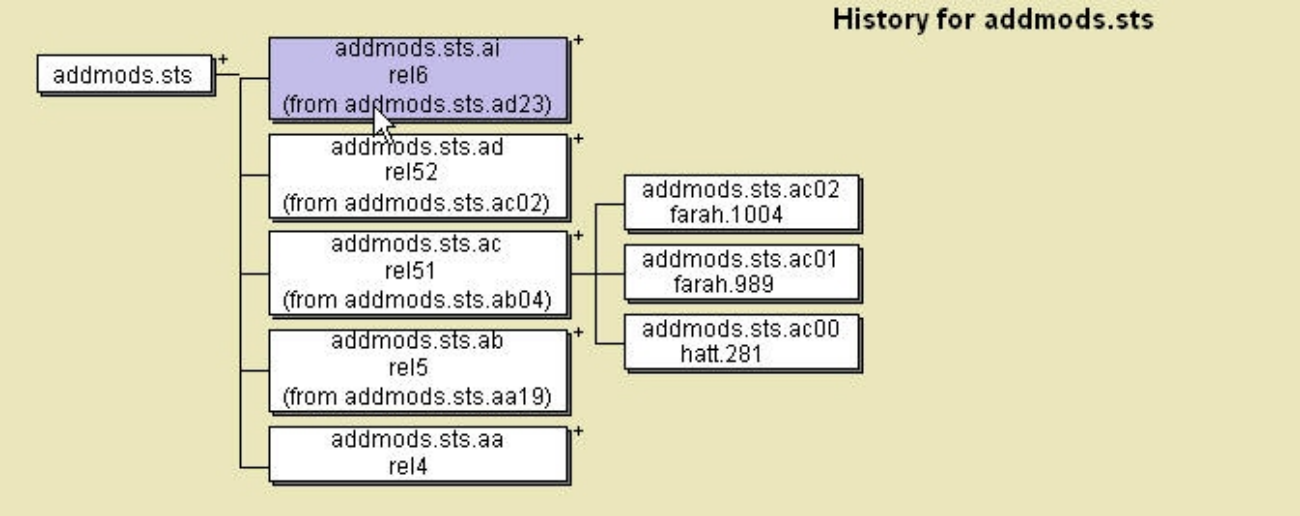


Because branch naming is independent of ancestry, branching in CM+ forms a two-dimensional (2D) branch/revision (also referred to as branch/issue, as in “issue” within a branch) structure. This is a

simplification over many older processes and tools. However, the real simplification comes because, in CM+, branching is not required as often as in other tools. The reasons for this trace back to a combination of how branching is used in other tools - its use is generally overloaded, primarily because there are not sufficient first order objects and other algorithms to support the wide range of change management requirements – and the amount of time the tools/processes require checkouts to persist. So the default approach in some of these tools is to branch and label.

CM+ on the other hand uses first order objects for change packages (Updates), build records (Builds), and baselines (Issues). Traceability to features is done through Updates rather than directly. Algorithms are used to allow promotion levels to be tracked without branching, by using an optimistic, rather than a pessimistic approach. Additional algorithms are used with Updates to track short term parallel checkouts without the need to create branches.

Because CM+ does not overload its branching there is a three-fold effect. The first is that 2D branching is simple and generally maps onto the Product Road Map, a listing of what releases the product has been through and is planning on going through. This eliminates the need for a complex branching strategy and the associated learning curve for it. Secondly, with fewer branches there are fewer merges required, and branch labelling is virtually eliminated. Thirdly, because of the regular branching pattern in CM+, which reflects product history, CM+ can infer much more accurately a user's intent and provide assistance in a more automated fashion.



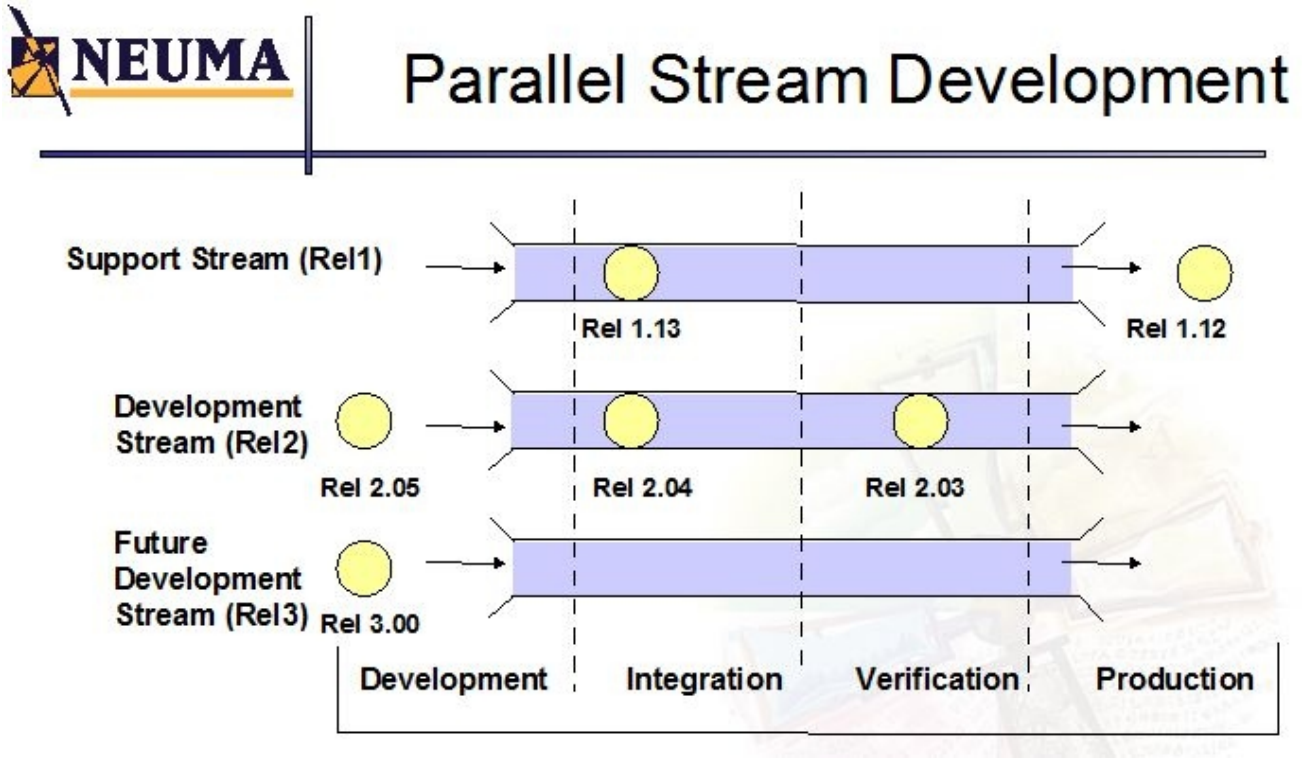
For example, because the user has a context set, which is captured by an Update, a checkout operation to an Update can immediately tell the user if a branch operation is required. Similarly context views can be established without the need to write a "configuration view specification", significantly reducing potential error. Merge ancestors can be easily identified. Problems fixed in one stream but not another can be easily identified. And Build comparisons are easy and fast, showing Updates performed, File Revisions created, Problems fixed, Activity Features implemented, Requirements addressed, and so forth.

2D branching, along with the capabilities that support it, is a unique feature of CM+, and is responsible for a dramatic reduction in both developer and CM manager overhead in using the tool. CM+ history browsers can be used to view source and directory history and to easily identify the revisions (and branches) being referenced by the current user context view.

Version control in CM+ applies both to text and non-text files. For text files, a Source Manager (SM) is used to perform delta compression on the multiple revisions of the source file, in a manner that makes it quick to re-create a revision, regardless of the number of revisions. Binary (non-text) files are classified as well-behaved source manageable binary files (smbin) or arbitrary binary files. The former, such as Office documents, can also be delta compressed. The latter, such as executables, cannot be delta compressed. Storage within the repository may still, however, be in compressed format with expansion on retrieval. This is controlled on a file class by file class basis. [A file class is essentially a set of properties associated with one (or more than one)

file suffix.]

Version control applies not only to files, but to higher level objects as well. Products are organized into Releases, which one or more release in each Stream of a product. Baselines are defined, not only for releases, but also for intermediate states of the implementation components, whether they be requirements, source code or otherwise.



This branching, at a high level, permits Parallel Development for a project. Again, Streams are used to identify the parallel branches of product development, and these Stream identifiers are attached to all information associated with the product, including source code branches, requirement branches, activities/tasks, problem reports, test runs, and product builds. In this way, Streams are used to support Parallel Development.

Configuration and Change Management

Although Configuration Management (CM) is an essential component of any SCM tool, it does not have to be a central component of a user's experience. In CM+, CM operations such as revision identification, branching, configuration alignment and status accounting are automated. A user may invoke an operation which leads to such CM operations, but the operation itself is generally automated. Furthermore, users generally do not have to be aware of which revisions they are working with. Instead, a user context is used to infer the correct revisions. This applies both to developers and configuration managers.

CM+ is designed to allow its users to focus on Change Management. Change requests appear on to-do lists (typically in the form of separate Problem Report and Feature Activity to-do lists). Changes to objects are tracked through Updates which are automatically linked to the to-do list item when the process to begin the change is started. All objects (e.g. source files) are checked out against an Update, including newly created objects, and all structural changes are recorded as part of the Update.

Updates are change packages which track the files modified, the reasons for the change, the product and stream to which the change is targeted, as well as additional integration and unit testing information. Other

information can be tracked as required by your own processes (e.g. peer review comments). Targeting Updates, and related information (e.g. Problems, Requirements, etc.) to specific development streams of specific products is known as Stream-based Change Management. Stream-based change management ensures proper process and data flow through the entire application life-cycle.

Change-based CM is the process of using change packages (i.e. updates) throughout the engineering portion of the ALM cycle. Rather than having to check-in individual files, perform delta reports on individual files, merge files from one release to another, promote files, etc.., CM+ allows change-based operations. A single check-in operation will check in all files belonging to an Update. Similarly, a single click is used to review the entire Update. An Update can be propagated from one stream to another, if necessary, in a single operation which will create a separate Update for the target stream. Updates, rather than files, are promoted - fewer operations, fewer finger problems. And with any promotion, CM+ verifies that dependencies are not being violated and informs the user of the specifics of any such violation.

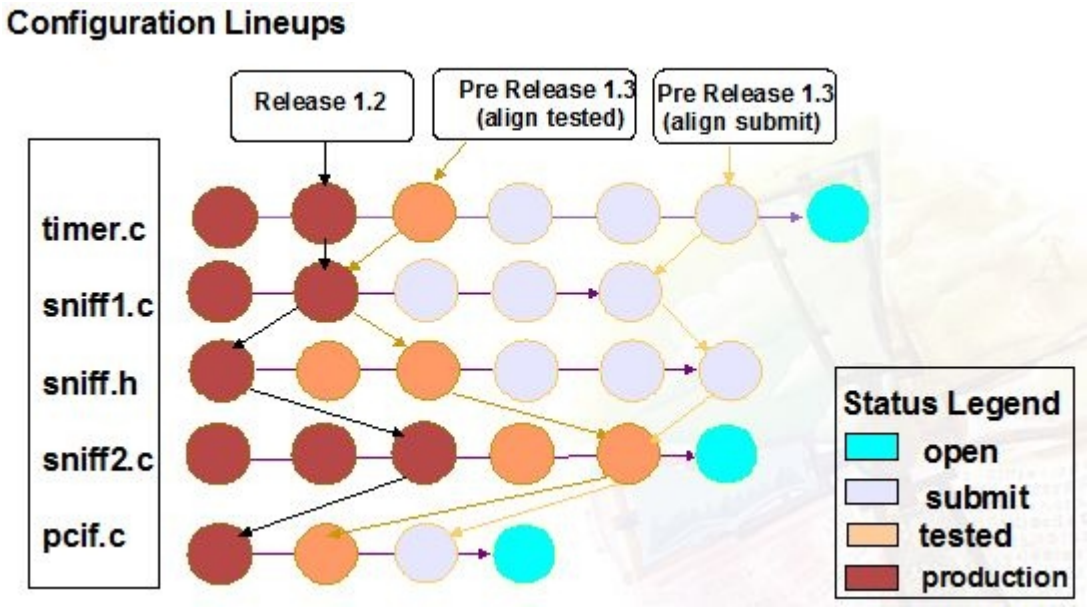
From a CM Manager's perspective, Updates are promoted into the nightly build, and the nightly build is recorded in terms of a baseline plus a set of Updates on top of that baseline, making it easy to eyeball the changes. The build definition can be modified in the case of an unsuccessful build, simply by removing offending updates, or adding in new ones, until it is finally frozen, typically on a successful build. As well, baselines are generated based on Update promotion levels. A configuration is automatically defined based on Update promotion and can be redefined by promoting additional Updates or rolling back Updates, numerous times until the configuration is frozen into a baseline.

Although most modern CM tools have change packages, many have these as an add-on feature, in some cases requiring separate databases and even separate user interfaces. Neuma built CM+ from day 1, not just with Updates as the central core item, but with over a decade of prior experience in using Updates in large telecommunications projects. As a result, you'll see a lot of change-based capabilities in CM+ that are not present in other tools. You'll also see a lot of traditional CM operations (such as labelling, complex branch resolution, and manual configuration definition) missing - either because it is automated, or simply not required.

Baselines, Builds and Release Management

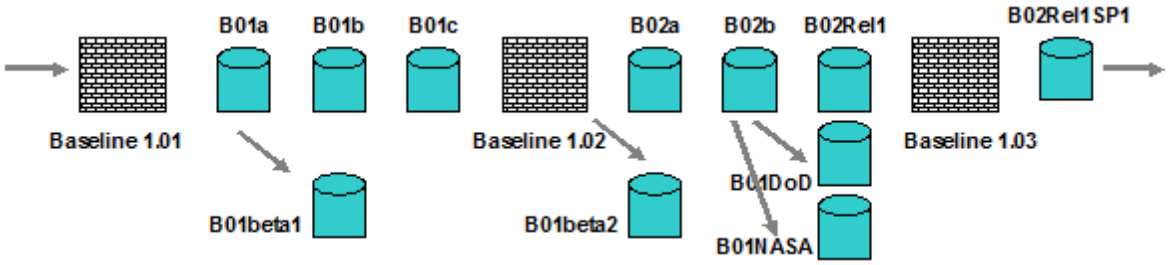
A Baseline is a reference configuration, typically reflecting a significant, or at least coherent, set of revisions which can be used to generate a software deliverable. In CM+ a baseline is specified as a set of file revisions, but this specification is hierarchical. The baseline is a set of directory and/or file revisions, and the directory revisions themselves are baselines for the directory. This hierarchical definition allows the creation of baselines at any level of the source tree. In addition, the creation of a subsequent baseline can re-use sub-directory baselines if there are no applicable changes to those sub-directories. In the end, the expanded revision hierarchy forms the base-lined set of revisions.

A Configuration is either Open, that is, being configured, or Closed, that is, frozen for all time (at which time it becomes a Baseline). The "align" operation of CM+ allows a configuration to be created, or an open configuration to be modified, based on a set of rules. Typically these rules indicate the product, the development stream, the promotion level and options indicating whether fixes from earlier streams should be automatically included in this stream, whether sub-products found in the tree should also be aligned (as opposed to restricting alignment to the product proper), and whether or not to override stream history. The promotion level instructs CM+ to align into the configuration only those revisions whose Updates have attained the specified promotion level. Alignment creates or modifies a configuration, but does not freeze it by default. A "freeze" operation, or option on the alignment, causes the configuration alignment to be forever frozen in time as a Baseline.



The baseline name is identified by the top level directory of the alignment, along with a revision identifier, established in the same manner as for source code, that is, whenever the content changes. Directories are very similar to source files in that they may or may not need to branch, and otherwise need new revisions created in order to change them (unless they are in the process of being changed already – i.e. not yet frozen). The difference is that the content of a Directory, rather than being source code text, is the set of file/directory revisions it contains.

A Build record, sometimes referred to as a Build Notice or Engineering Change Notice (ECN), especially when referencing it in the future, is a bill-of-materials (BOM) for creating a deliverable (or set of deliverables). A Build record identifies the BOM in terms of a Baseline, along with a set of Updates that are added on top of the Baseline. A Build record may also specify a set of options which cause only a subset of the specified (Baseline + Updates) to be used in creating the deliverable. For example, there might be separate French and English options, either or both of which could be selected. In this way, multiple Build configurations can be derrived from a single Baseline. A typical evolution of Baselines and Builds is shown in the accompanying diagram.



Unlike Baselines, which are Closed configurations, a Build can go through a set of promotion levels. Typically the promotion levels might be: Open, Defined, SITested, VerTested, Field, Beta, Production. That is the build can progress all the way to a "Production" status. Most builds tend to go as far as "System Integration Tested", with some being used for verification testing, and others progressing through to field trials.

In CM+, a Build is defined usually by referencing a previous Build or Baseline, and specifying the set of Updates to be added to that Build or Baseline. The set of Updates, in turn, are selected in a number of ways. Sometimes, all “ready” (or even “checked in”) Updates are automatically folded into a nightly build. Sometimes the Build Manager will specifically promote individual updates from a list that are marked “ready”. And typically, as the end of a release approaches, updates might be selected based on the Problems and Features they address. The Build Manager has control over how to define the Build, and can drill down into details as

necessary during the definition process.

Builds are tested and results may be captured in the CM+ Test Suite Management function so that a record of successes and failures of test cases against specific builds is tracked. Eventually, a Build is considered ready for release.

For all releases, it is necessary to generate release notes which identify the set of changes made to the product since the previous release. In some cases, a Version Description Document is required with file revision details. More often though, the customer wants to know what new features there are, what problems have been fixed, and what problems are outstanding.

CM+ allows you to compare any two builds to determine this information and to present it interactively or in text format, ready for a technical writer to create releases notes. However, the information given to the tech writer is only as good as that entered into the repository. As such, part of release management deals with educating users as to how to properly raise a problem report, reply to a problem report or document a feature abstract (i.e. a product requirement). When done properly, the tech writer's job is simpler.

Although not part of the default CM+ configuration, it is possible to track build deliverables against customers so that at all times you are aware of which builds your customer has received, and which builds your customer has installed.

Test Suite Management

Test Suite Management in CM+ deals with the storage, organization and retrieval of test cases, as well as the organization of test results when test cases are run against their intended applications.

CM+ allows the management of both manual and automated test cases in a manner based on extensive experience. Unlike source code, which is bulky and continually changes, test cases tend to be short snippets of instructions which tend to remain largely unchanged over time, and even across generations of products, depending upon the application. Product (i.e. black box) test cases tend to have a much longer life time than the products themselves. In fact, some test cases are designed to support a standard which is to be enforced across many different products from many different vendors. For these reasons, Neuma believes that it is insufficient to take a source code management application and use it for test case management.

Test cases tend to be quite numerous and are best organized functionally, according to the functionality they are designed to test. Because of the large volume, CM+ is configured, by default, to automatically number test cases, and to allow them to be organized hierarchically. Each test case is tied to a requirement or problem report and through this linkage, a traceability matrix can be formed. In CM+ this matrix is queried to identify, for any sub-tree of requirements, which test cases are used to cover it off, which requirements are missing test case coverage, and for any sub-tree of test cases, which requirements they address.

As well as the test cases themselves, there are the results from running the test cases. In CM+ the running of a series of test cases are tracked as a test session, which is part of a test run. Each test session identifies the Build against which the tests were run, the tester, and the set of tests that were run, as well as those not yet run. Finally a test session identifies those test cases that failed during the session.

A test run is a hierarchical collection of test sessions. This allows test sessions by different testers, or on different dates, to be tracked as a single test run. In some cases, such as initial verification testing, test sessions may even span builds, as corrections are made to permit completion of a test run, where otherwise the functionality might preclude it. In addition to tracking test case success/failure, failed test cases can be documented as part of the test run, or may be documented against individual test cases. However, well defined test case structure will usually preclude the need for additional documentation.

Because of the traceability of test cases to requirements, it is also possible to trace requirements to both successful and failed test case attempts. CM+ will let you identify failed test cases for a given set of requirements, as well as the affected requirements of a set of failed test case attempts. More complex

queries, such as "When did this test case last pass/fail?" are also possible.

Document Management

Document management encompasses a wide variety of management techniques and support of a variety document types. There are several classifications of documents and each classification requires its own process. Examples of document classes include:

- Product User and Reference Guides
- Design Documentation
- Status Reports
- Technical Notes
- Attachments

Each of these would typically have different management requirements. In CM+, document management is sometimes a process on its own, and sometimes part of another object process. Looking at these 5 examples, we'll explore this in more detail.

Product Guides are typically part of a product and can typically be treated in a manner similar to source code. They have a Product to which they belong and typically evolve in each release of a product. As such, they are best managed in the same way as software. The input for changes to product documents should normally be the product requirement documents themselves, as these detail how the product is to operate.

Design Documentation has two components, the overall product design, and incremental design documents used to address particular features as the product evolves. The overall product design documents should be part of the software source tree and updated in a similar manner. Often, product design documents are updated in a major step near the end of a release based on a set of incremental design documents. Incremental design documents are typically generated by the developers as part of a feature design. In CM+ it is possible to associate the document directly with the feature activity, so that the activity number itself can be used to identify and to access the document. Incremental design documents do not require 2D version control because they are targeted to a single release. Instead, they usually evolve in a linear fashion, typically with reviewer feedback attached, until they are signed off by the reviewers. At that point, implementation is started. Attaching such incremental documentation directly to the activity has the added benefit that it can be readily pulled out as part of a build comparison to more completely identify the operation of new features.

Status reports tend to be a serial type of document. Each week a status report might be generated and saved. Identification of such documents is best automated within an umbrella identifier for the entire series. At most, very rudimentary version control is necessary so that corrections to a status report may be tracked. However, a status report does not otherwise evolve. It should be easy, however, to navigate status reports in the sense of the next and/or previous reports.

Technical notes are a type of document that might be fairly frequent for a product. They generally convey techniques or other useful information regarding the use of a product, perhaps for a specific application. Such notes may need some linear version control so that information may be clarified and expanded over time. The names of technical note documents might be as simple as TN.### where the ### is a number assigned by the CM+ application. Typically, there is no relationship between one technical note and another. Other document types such as PN (process notes), CV (customer visit) or similar types can be defined so that the organization of this type of document may be simplified.

Attachments are typically arbitrary files that are attached to another object to provide additional information. For example, a problem report might have a screen capture or a core dump attached. A requirement might have an equipment drawing or a screen sketch to indicate expected behavior. Attachments are identified arbitrarily (usually by number, such as att.999) and accessed through the object to which they are attached. They otherwise have no organization.

Additional document classifications may be specific to your organization. CM+ can be customized to deal with

a variety of classifications and requirements.

Extending your ALM Suite

The CM+ ALM Suite can be easily extended with additional applications or with additions to existing applications. Some applications are available from Neuma (e.g. Meetings, Time Sheets, Customer Tracking). Others can be readily generated in a matter of hours to a few days. The intent is that CM+ can capture all of the necessary data to manage your Application Life-cycle.

Generally, application design involves a few steps: identification of the data schema for the application objects, identification of the process to go with the objects, creation of pull-down and pop-up menus for the applications, creation of to-do lists appropriate to various user roles, and creation of one or more dashboards to simplify the user interface. Documentation of the application is another matter. But for the most part, if the design is in line with the other ALM functions, the application should be fairly intuitive, requiring only overview documentation and a bit of process guidance.

It is generally far easier to extend the CM+ ALM suite than to acquire and integrate a separate tool. By extending CM+, you continue to use a single repository, with the same administration, the same CM+MultiSite capabilities, the same process engine and the same user interface. At the same time, your technology transfer capability remains simplified.

Capability Overview

A 4th Generation CM/ALM tool goes well beyond a 2nd or 3rd Generation tool. Much less support is required, and there is an abundance of additional capabilities. This section describes some important framework capabilities of CM+ from a development perspective, allowing the user to understand, not so much how to do something, but simply that CM+ can meet the requirements needed. Areas covered include:

- CM+ MultiSite
- Process State Flows and Promotion Levels
- Context Views
- Workspace Management
- Delta and Merge Tools
- User Interfaces
- Roles and Permissions
- Integrating with IDEs
- Data Export and Import

Each subject is covered briefly to introduce the concepts to the reader.

CM+ MultiSite

CM+ MultiSite is a component of CM+ Enterprise which allows a development team which is globally distributed to share the same CM/ALM repository. Typically, all repository updates are queued, as transactions, to each site for execution in the exact same order. The result is that each site contains a full set of project data. This also makes each site a potential disaster recovery point for all of the other sites. In some cases, CM+ MultiSite is used solely for disaster recovery.

It is also possible to customize CM+ MultiSite so that certain sensitive data (e.g. documents, source code) is restricted from being made available at specific sites. In such cases, restricted sites do not provide a full disaster recovery capability.

CM+ MultiSite allows users to travel from one site to another without, in the general case, any differences in their use of CM+. They have access to the same data, regardless of their location. And data is replicated in "real-time" (i.e. transaction by transaction) so that no site has an advantage over the other sites.

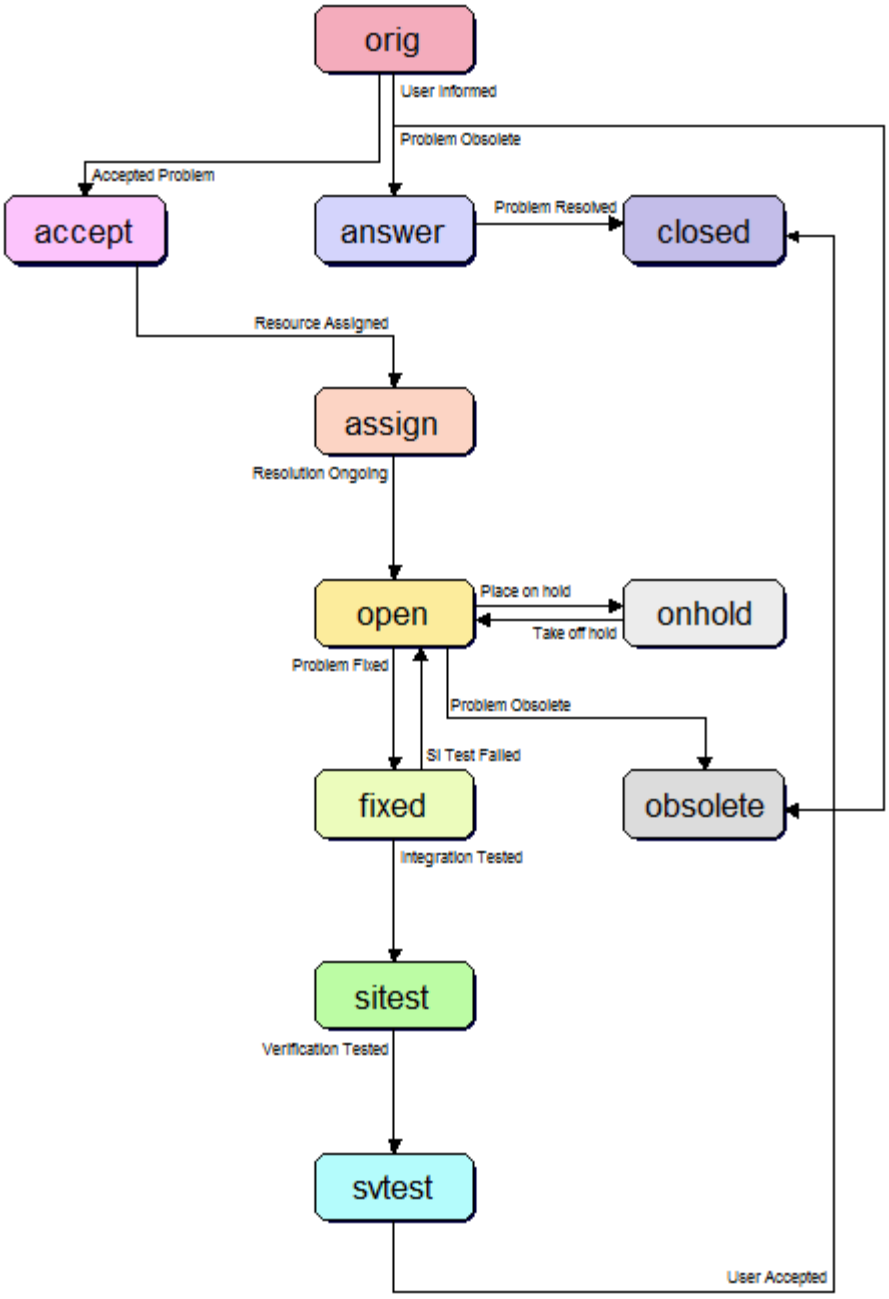
If a site is off the network for a few moments or for a few weeks, CM+ MultiSite will automatically restore the site to its proper state when it is reconnected. This allows, for example, a CM+ library to be disconnected from the network and brought on-board an aircraft, a sea vessel or even the space station. As a site node, full access to all of the data at the time of disconnection, is available to any clients (i.e. in its temporary sub-net) connected to the disconnected node.

Process State Flows and Promotion Levels

Most objects tracked in a CM+ repository have a specific set of states in which they may reside. For example, a Problem Report might go from Originated to Open to Fixed to Tested. The same goes for Updates, Build records, File Revisions, Documents, etc.

Along with the set of states, a set of permitted transitions (from one state to another) determine the State Flow for an object. The State Flow is often presented visually as a State Flow Diagram. Each transition is labelled with the action which causes the transition to occur between any two named states.

Problem Process Flow

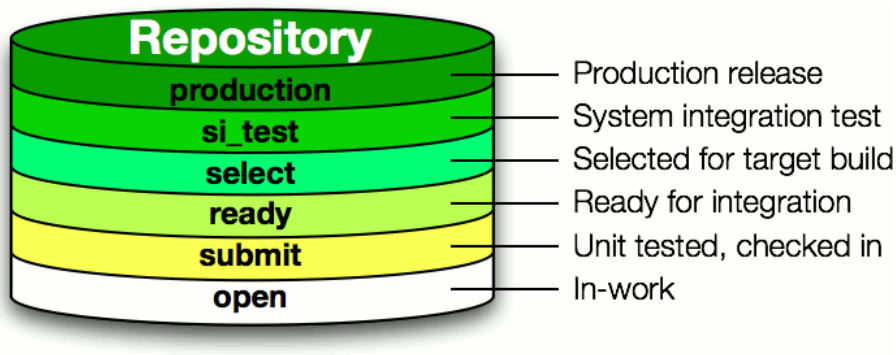


For process-driven applications, each transition has a set of permissions associated with it: which role(s) are necessary to perform the action causing the state transition. Together with owner and assignee fields, this information is sufficient to automatically build and present to-do lists to each user, based on the user's roles.

Revisioned records, such as source code or requirements, tend to have a very small number of states, typically: Open, Submitted, Closed and Obsolete. In effect, a revision is either Open (i.e. in progress, and possibly already checked in), and Closed (checked in with no further changes allowed). An intermittent state allows "publishing" of the content before it is determined if further changes will be made.

The set of states in a normal successful flow of an object are stacked in an ordered set to indicate promotion levels. The term promotion level is used when there is a, generally linear, set of states through which an

object traverses, and the application is interested in whether or not an object has reached either a specific state, or a higher state, according to the stacking order. For example, this is typically the case with Updates, where Updates at a Selected state or higher are used to define a "nightly build" operation.



States are typically color-coded so that data can be highlighted with the color to show the state more visually. The color coding is also used for state-based graphs, gantt charts and state flow diagrams.

Context Views

A CM+ library (i.e. repository of information) contains a lot of information, for multiple products, multiple releases of each product, for various functions within each release, etc. The data is overwhelming. A context establishes a view of a subset of the data that enables a user to work within a specific environment.

Generally, a context is established by specifying a product and a development stream. When a user looks at his or her to-do lists, they generally present information within the user's context view. So if a developer has problems assigned to him for fixing, the context view would normally cause only those problems for the specified product and development stream to appear within his to-do list. Similarly, when creating an Update, the update would, by default, be targeted to the product stream indicated by the context.

A context also affects the view of source code in a source tree. The structure and content of the tree reflect the given product as it appears for the specific stream given by the context. By default, this is the latest revision of code in a specific stream. However, by setting the promotion level context, it can instead show the latest revision which has attained, at a minimum, the promotion level specified (i.e. the Update which produced the revision has been promoted to or beyond that promotion level).

Context views may also be specified by selecting a specific Build record or Baseline. This allows viewing and retrieval of precisely that code which went into creating the Build or Baseline. A context view can also be modified by adding a set of Updates to the existing view. For example, to identify a patched Build being used for development, a user might select a specific Build for the initial context, and then add in the Update(s) which constitute the "patches" to that build.

Context views may be dynamic or static, or even dynamic with static backup. A dynamic view changes content as Updates are promoted. The rules indicate what is part of the context view, and as the data changes, the content of that view may change. A static view identifies specific revisions that are part of the context view. Builds and Baselines are examples of static views. When a dynamic view does not resolve the context for a particular file, the static context, if specified, may be used to resolve it. Otherwise, the latest revision of the latest branch is used to resolve it. Applying Updates to a context causes the content of the Updates to override the dynamic context. This is an example of a static view component of the context overriding the dynamic view.

Context views are used to pre-populate context-sensitive fields on forms, to intelligently select default values for form/dialog fields, to restrict the set of possible selections in a pick list, to restrict the set of records shown

in a query or report, and to select the correct revision of a context sensitive file, requirement or other item. This makes CM+ easier to use, requiring fewer keystrokes while improving the quality of data.

Data Schema

The CM+ application resides on the STS Engine, which contains a Hybrid Repository. The repository provides for both relational and other types of data, including revisioned data, large objects, hierarchical data, files, delta-compressed files, and various other data types. In addition, the user may create custom data types using either the DAD language or the STS command language. Schema is normally defined in two parts:

- a .dad file which contains the field/type definitions and which can introduce new data types
- a .sts file which identifies special properties, including permissions, on the fields, and specifies work flow.

Schema is organized in modules, with a module dealing with a specific functional area, such as Problem Reporting. Modules are defined by the .dad and .sts files initially, but can be modified as part of a normal database transaction, to add/remove/rename fields, change field types, change properties, etc. Modified schema may be exported into revised .dad and .sts files. This is generally useful only if the data schema will be used for creating additional libraries.

Schema definitions include reference value and reference list fields. Such fields will automatically result in navigation links in the various mechanisms of the user interface. In addition, a field may be specified as a "sub-table" field, meaning that the field points to a set of records which belong to (i.e. are owned by) that data field. For example, the "revs" field of a "file" points to the "branches" (i.e. branch records) of that file.

The various properties that can be assigned to fields affect the behavior of the application. For example, a "statdate" field is updated whenever the "status" field is modified. There are dozens of properties that may be assigned to the various fields of an application. Below is a typical display of the most interesting aspects of the "Problem Report" data schema.

TABLE: probs	KEY: ProblemId	RECORD: Problem						
Field Name	Type	Size	Property	Class	Table.Field	Options	Width	Title
type	ProbType	1	subclass	data				Type of object with problem (doc, sw, etc)
status	ProbStatus	1	status	data		M X		Indicates action to be taken and by whom
date	Date	2	statdate	data		A R		Date of last status change
owner	UserId	2	owner	ref	#users	A		Person responsible for problem correction
assignee	UserId	2	subowner	ref	#users	A		Person assigned to fix the problem
priority	Priority	1		data				Urgency indicates timeframe required for fix
product	ProductId	2	product	ref	#products	A		Product under which problem was discovered
stream	Stream	1	stream	data		A		Product stream to which problem relates
request	RequestId	4		ref	#requests			"Request that Spawned this Problem"
title	Textline	4	title	textline				Brief, Unique description of the problem
reviewers	List	4		list	#users			Users responsible for reviewing work
originator	UserId	2		ref	#users	A		Person who initiated the problem report
from	UserId	2		ref	#users	H M		Previous owner of the problem
location	Textline	4		textline			32	Brief description of problem location
category	ProbCategory	1		data				Categorization of problem resolution
effort	Shortnum	1		data				Effort taken to resolve problem
forecast	Date	2		data				Latest forecast fix date
dates	StatusDates	16	statdates	appltab		M R		Record of dates each status was achieved
attachment	List	4		list	#attachments	HPM		Attachments to this problem
notes	Note	4	reply	note				Problem description and all (datestamped) replies

Workspace Management

When working with files under version control in CM+, your local files are stored in your workspace. Your workspace may be used for any number of updates and you may have more than one workspace to work on one or many projects. Your current workspace is reflected in the status bar under the label "Dir". If you start to make changes or if you are otherwise in a situation where your workspace has not been explicitly specified, your current workspace is used as your workspace. Once you associate your workspace with a specific source code update, that workspace definition is used for operations associated with that update (e.g. check-in, check-out, delta, etc.).

Typically, source code files are arranged in a directory tree. Your workspace directory specifies the root of such a tree. Whenever you retrieve a file, it is placed in the appropriate directory under the tree root, with directories being created if necessary in order to place the file there.

It is also possible to work in a "pooled" workspace where all files reside in the same directory rather than in a directory tree. In this case, your workspace directory specifies the directory containing the pool of files. In this latter case, all files which might appear in your workspace must have unique names.

The active workspace feature of CM+ is an extension of the "checked-out" indicators found in many IDEs and CM tools. CM+ actively updates the source tree browser to reflect the current state of the workspace any your context view. The key points of identification are:

1. Is a file checked out by the current user
2. Is the file checked out by another user
3. Is the file present in the workspace
4. Does the file in the workspace have read/write access
5. Do the contents of the workspace file differ from that of the context view
6. Is a directory frozen or open (i.e. can be modified)

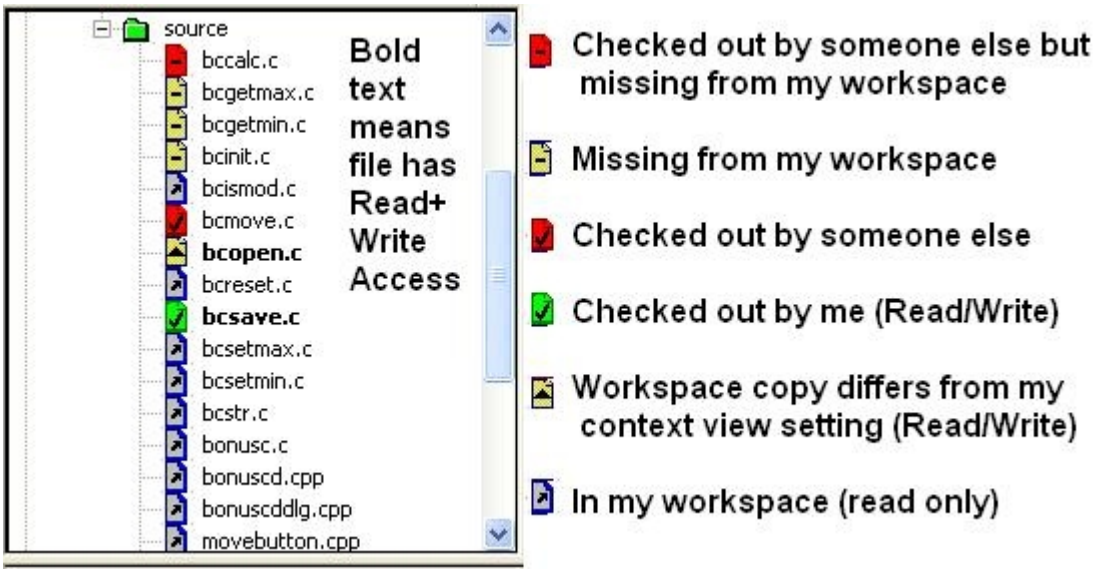
CM+ uses color, symbols and bold text to indicate the various conditions.

Color is an important indicator of the state of a file. A check-out operation is indicated by a green (by user) or red (by another user) color. The check mark is added if another symbol does not override it. Yellow is generally used as an alert (missing or different). A directory appears in green unless it is frozen, in which case it appears in grey.

The symbols used for a file icon are a dash (missing file), a delta (file differs), a check mark (file checked out) or an arrow (file matches repository)

Bold text indicates that the file in the workspace is writable (i.e. can be edited).

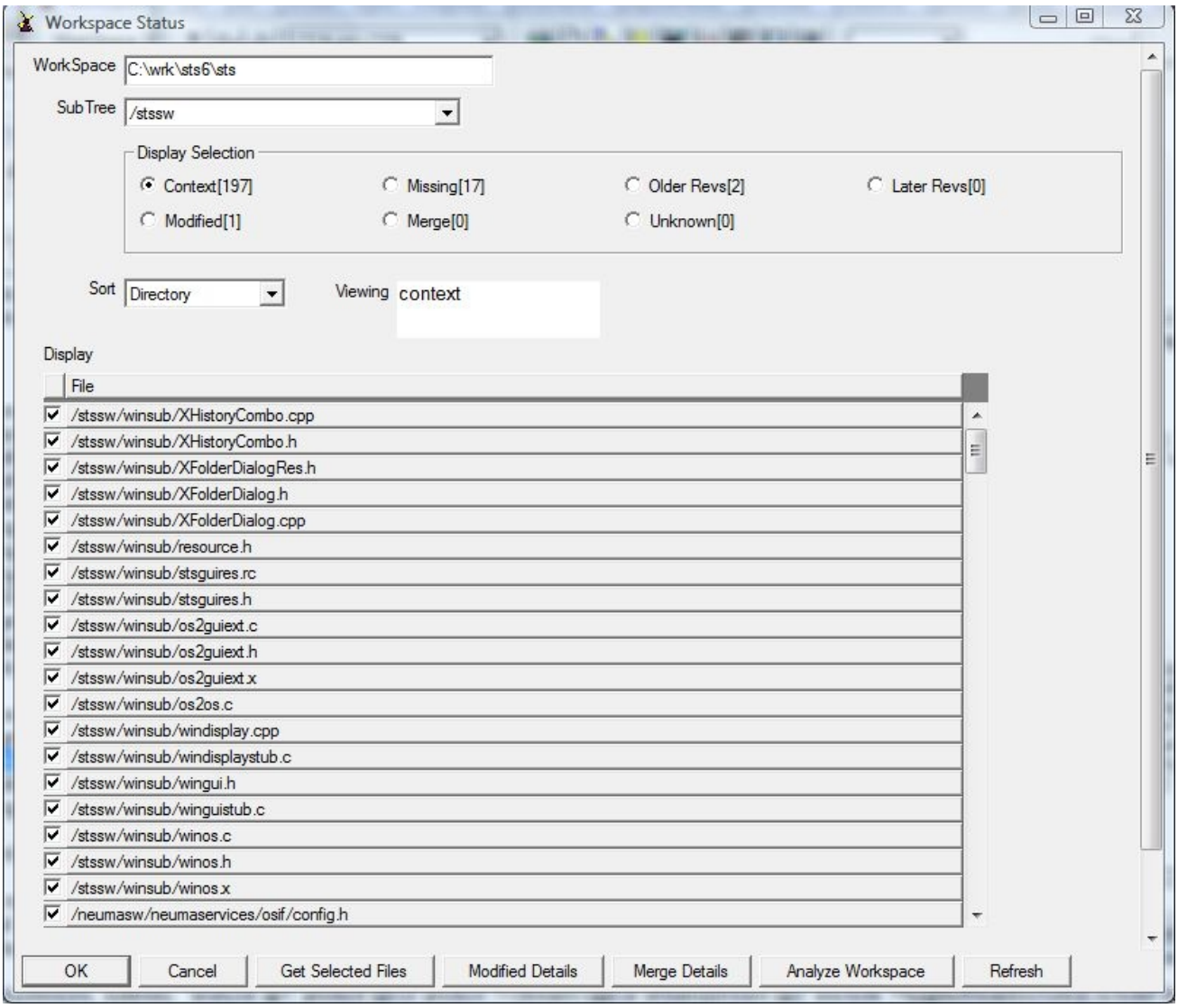
ACTIVE WORKSPACE EXAMPLE



When a file in your workspace is modified, or its read/write attribute is changed, the active workspace will reflect these changes. Similarly, if you retrieve a file from the repository into your workspace, the active workspace will reflect the change. As well, if your current context view is changed (e.g. you Set Context to another view), or the status of a file in your context changes (e.g. it is checked out or a new revision is checked in), the active workspace will reflect the change. When you scroll through your source tree, it may take a few seconds for the active workspace indicators to be adjusted correctly.

Note that folder colors will vary based on the state of baseline definitions in progress within the user view. A "gray" folder indicates that the folder in view has had no changes applied to it, within the current context, since it was last frozen, typically as part of a baseline definition operation. Otherwise, the color (green, red or manila) indicates that there are changes since the last freeze operation. Manila is the normal coloring, while green or red indicate that you or another developer actually have the folder checked out – this is a rare occurrence in CM+ as folders do not need to be checked out to make changes to them.

CM+ supports simple workspace population by clicking on a source tree, or any sub-tree and selecting a "Get" operation. CM+ manages, by default, the entire tree structure of the product for the workspace, whether you are retrieving the entire tree or just a few files. "Compare to Workspace" and "Synchronize" operations allow you to identify differences between your context and workspace, and, if desirable, to synchronize some or all of these.



In general, at any time, your workspace will have a set of files which can be collected into various “workspace” states. Some may match (your view of) the repository, others may be older (or in some cases newer) revisions from the repository but not matching your view. Some files may be missing from your workspace, or perhaps you have files that are in the workspace but not the repository. In some cases, you may have modified the files in your workspace, while in others, the originally matching repository view has changed because others have modified and submitted (i.e. checked in) the files into your view of the repository.

In these cases, a set of workspace actions are needed to reconcile the workspace to your view. In some cases you may want to capture the changes in your workspace and check these in. In some cases you'll need to re-base (i.e. merge) changes made to the repository into your workspace, or perhaps simply retrieve the later revisions. A Workspace dashboard is available to support these operations. It may be customized to your specific development requirements.

Check-out and Check-in

The concept of check-out, in the SCM world, is taken from the library book model. When you check out a book from the library, you have it and nobody else can have it. When you check it back in, somebody else can check it out. In some cases a library will maintain a queue of those waiting to check out a book and will check

it out for the next person in the queue as soon as it is checked in.

In CM+, the concept of check-out is similar to this. A check-out operation on a file means that you have control of the file and others must wait their turn. This check-out control applies to a specific stream (i.e. branch) of the file, not to all branches, and applies to what is known as "exclusive" check-out operations. A check-out operation is always performed against an update. The stream of the update determines which branch of the file is being checked out.

The check-out operation refers to the reservation of the file (branch), not to the operation of getting the file to your workspace. However, when you do check out a file, you are given the option of retrieving it to your workspace. The workspace associated with the update is the one used for all check-out and check-in operations against that update. The "Get Source" operation can also be use to retrieve source code to your workspace, whether or not the file has been checked out.

A checkout operation is usually an exclusive checkout, meaning nobody else can check out that branch of the file until the exclusive checkout is completed (either by a checkin or a cancel checkout operation). If a file is exclusively checked-out, it may be possible to "queue" for check-out of the file. If your site is configured to allow queueing, this option will be available on the check-out panel. When the current checkout is completed, the first checkout operation in the queue is processed automatically.

It may also be possible to check out a file in "parallel" mode, if it is not already checked out exclusively. In this mode multiple parallel check-out operations may be performed on the same file branch against various updates. The file revision associated with your check-out will change whenever a peer check-in on that file branch occurs - you may or may not notice this. In the case of parallel check outs, the first update checked in claims the current revision number, and the outstanding checkouts are up-issued. CM+ may be configured to notify users involved whenever a parallel checkout is started or completed.

Source code files in your workspace will normally contain a commented id line as the first line of the file which indicates where that file was retrieved from. The information in this line should not be edited so that CM+ can more accurately suggest appropriate action. For example, if parallel checkouts are permitted by your administration, CM+ will use the id line information to detect, upon check-in, if you require a reconcile operation. A "reconcile" operation is a type of merge due to parallel checkouts or other operation sequences which effectively cause the same effect (such as starting your change from an older revision of the file). Reconcile detects that a file has been modified and checked in since the starting point of your change. When you go to check in, CM+ will ask you to reconcile the change, which you may, at your own risk, ignore. There are valid reasons for ignoring this request. For example, perhaps your change is trying to perform an alternate fix to the fix that was applied as the last check-in.

Parallel checkouts were once considered a fairly necessary way of life in SCM. However, with improved processes, it is possible to minimize the length of time a file, which is in frequent contention for change, is checked out. Many shops take advantage of such processes and eliminate the need to support parallel checkouts at all. This simplifies work flow, as testing only has be done on the original change, and not on merges. Reconcile operations become much less frequent as well. This is a recommended mode of operation, providing your design architecture is not at odds with it. It has been validated in both small and very large environments.

Regardless of whether or not your administration permits parallel check outs, it is always possible to perform parallel changes. You retrieve the file from the repository (without checking it out), make your changes, test it, and at that point you check it out against an Update, and then check in the Update. CM+ will still identify the need to reconcile if it exists, as long as id lines are used for that class of file (and this is normally the case for source code files). The difference in this scenario is that the CM repository has no knowledge of your parallel "check out". Some organizations discourage this behavior and some force exclusive checkouts (with or without queueing), but allow this behavior as an emergency back door. Many CM tools use this as the primary (and perhaps only) checkout model.

With any type of check-out operation, the option always exists to Cancel the Check-out. This means that you've changed your mind. Perhaps you thought a change to a file was necessary, but it is not. Cancellation of a check-out can also be used as another "back-door" capability when an organization permits only exclusive check-outs – if a more urgent need arises, the initial checkout can be cancelled and repeated later on.

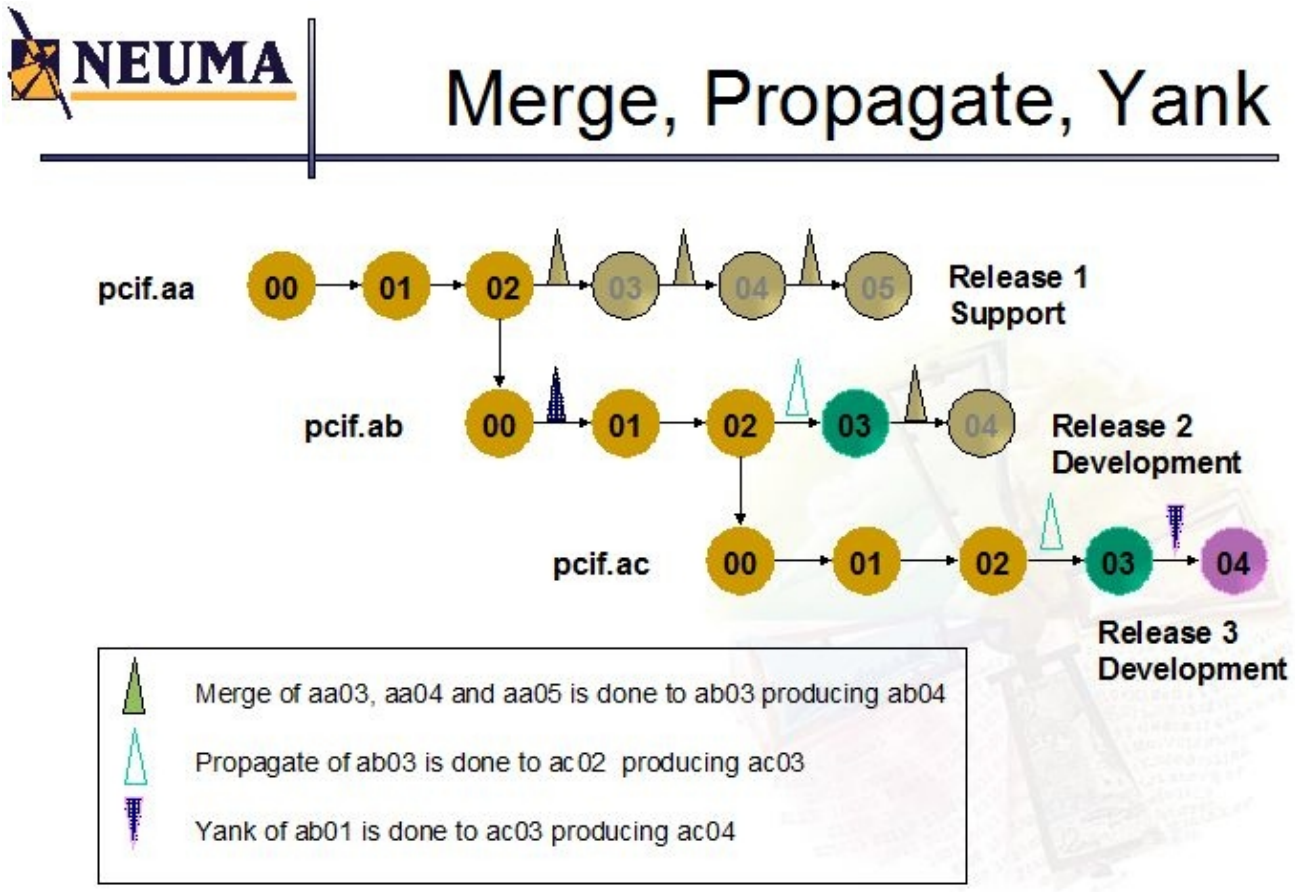
Finally, when you perform a check-out against an update for which the stream of the update does not match any branch of the file, CM+ will advise you that a branch operation is required. If you proceed, CM+ will create a new branch of the file as part of the check-out operation. If you subsequently cancel the check-out, the branch will disappear as well.

Delta and Merge Tools

Any modern CM tool supports delta and merge capabilities allowing you to identify and to merge parallel changes made to a common ancestor so that a single result may be used going forward.

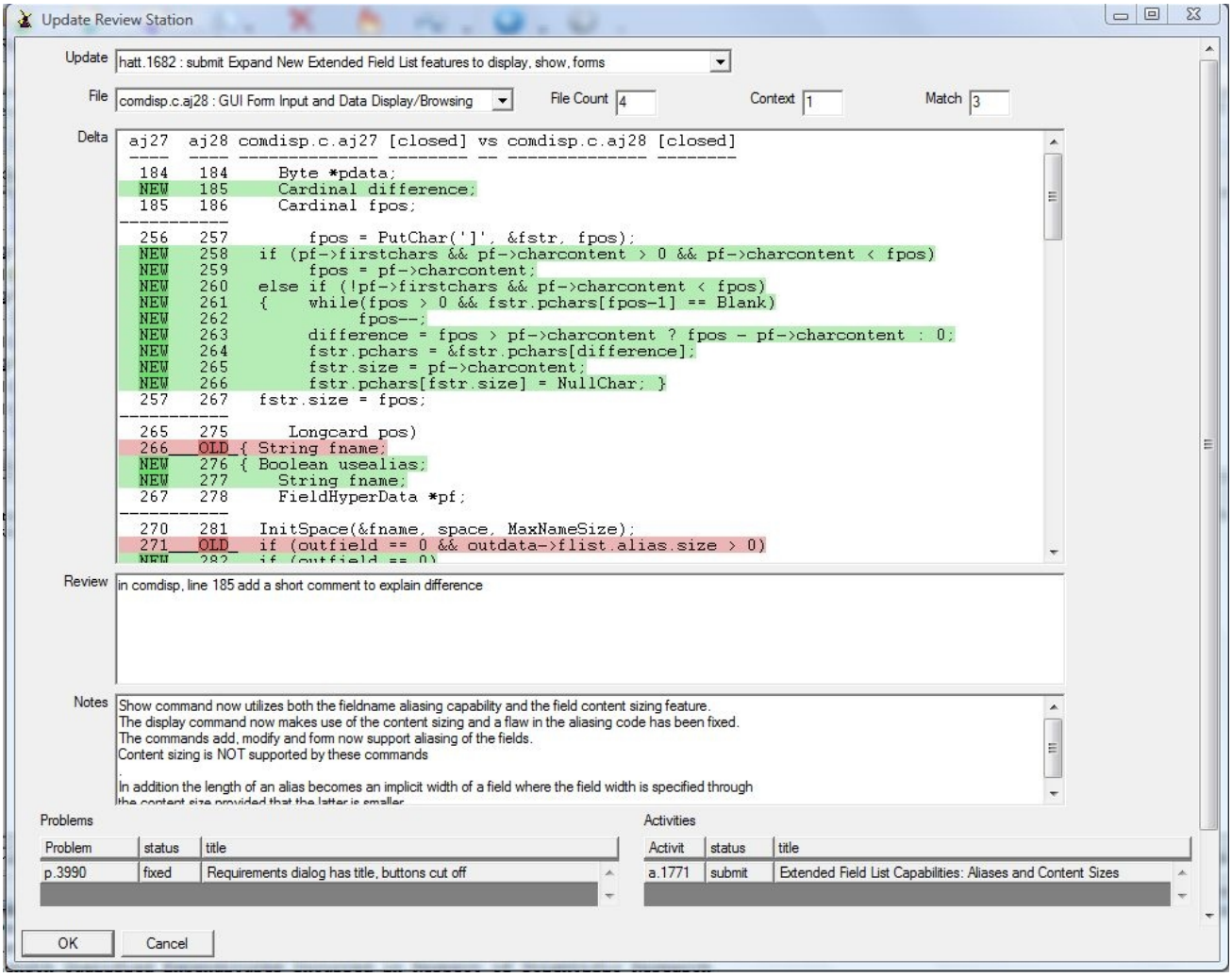
CM+ supports multiple tools to provide these functions, including both Neuma proprietary and third party (including Open Source) tools. The dashboard capability of CM+ provides an easy way to look at code deltas (i.e. differences) on a file-by-file basis, an update-by-update basis, or in various other ways. It also provides an easy way to navigate changes, by user, file, update or otherwise.

CM+ also supports a number of merge operations. These include:
- Merge – merge two parallel branches of changes into a single result
- Propagate – merge a single change from one stream into a parallel stream
- Yank – remove (i.e. un-merge) a change from the source code history.
- Reconcile – merge changes related to a parallel checkout
- Synchronize – merge changes made to the repository into a workspace



Neuma ships CM+ with two delta tools and one merge tool active, and an additional delta/merge tool available. Neuma's built-in Delta reporting tool is a text based tool which provides color-coded difference displays and

metric summaries. Any number of files may be compared in a single delta operation - typically, the files of an Update, or the files modified between two builds. The delta reporting tool can be integrated with dashboards, and a dashboard is provided which allows dynamic modification of options such as number of context lines, and number of matches to end a difference.



The built-in delta allows for text capturing of delta reports across multiple files, useful for peer review annotation, version description documents, and quick assessments of changes that have been made. The KDIFF3 tool (3rd party Open Source) is also active and allows for side-by-side (visual) views of a delta report, in the context of the entire source code file. The KDIFF3 tool is also the default merge tool used by CM+. Other tools may be substituted for the KDIFF3 tool, for either or both visual deltas and merging. The KDIFF3 is selected as the default merge tool because it allows multiple merge operations in a single invocation, and because it is available across platforms. The built-in delta tool is always available, regardless of whatever “visual” tool has been configured. This color-coded text delta is the one used in dashboards which show delta reports..

CM+ coordinates all delta and merge operations by calculating and accessing the files appropriate to the operation. For merge operations, CM+ calculates the common ancestor and the two differing files. CM+ allows for merging of two branches, propagation of a change from one branch to another, and yanking of a change from the history of the current context.

User Interfaces

CM+ Enterprise supports a number of interfaces:

1. CM+ Enterprise native interface (Windows, Linux, Unix, OpenVMS)
2. CM+ Command Line Interface
3. CM+ Web Interface
4. IDE Plugins for SCC API and Eclipse platforms
5. File Explorer Interface (Windows)
6. CM+ VFS (Virtual File System) Interface (Beta Windows)

By far, the first of these is the most used interface. It may be used remotely through a thin client. It may also be used in a remote site through the CM+ Remote Client facility, although this is rarely used because of thin client technology and because of CM+ MultiSite capability which supports multiple geographic locations for a library.

The command line interface is generally used for scripting purposes, especially for defining and customizing the native interface.

The CM+ Web Interface is recommended for giving access to non-project team personnel (e.g. the customer, auditors, etc.) or for giving information access and data entry capabilities to mobile off-site personnel (e.g. on-site support personnel).

The IDE plugins support a variety of IDEs including Visual Studio/.Net, Eclipse, etc. These support basic change control operations, and support the capability for extension to other aspects of CM+, including Problem Tracking. This requires a Fixed User license.

The File Explorer interface provides a means for less technical personnel to access CM+ from the file system as portrayed through the Windows File Explorer (i.e. Shell). This requires a Fixed User license.

Finally, the CM+ VFS interface allows a user to specify a context (i.e. view of the CM repository) which is mapped, in real time, to the Windows file system. All operating system or third party tool commands may be used on the files in the specified view, as if they were real read-only files on the user's file system.

Roles and Permissions

CM+ may be customized to define roles and the set of permitted operations allowed with those roles. Users may be assigned a single role or multiple roles. In some cases a user might have no roles assigned, limiting the use of the tool.

Roles are supported in a number of ways. In both the the native interface and the CM+ Web interface, specific capabilities are configured by role. These generally indicate the set of menus and items visible on the CM+ Web user interface, along with appropriate To-Do lists or other information Containers to be displayed.

Roles also dictate the layout for the native client user interface. This includes the menus and items visible (both for pull-down and popup menus), the set of information trees, containers and to-do lists shown in the "Tree" pane, and the set of tabbed reports pre-generated.

Roles are the primary driver both in configuring different views of the user interface, and in establishing Dashboards for efficient navigation of data and task completion.

Besides the user interface, there are a number of other items controlled by role. This includes state flow actions (i.e. Transitions, or who can do what when an object is in a given state), command execution, and data access (by Table, Field or Record).

Permissions are usually specified by roles, but a number of special Access Permissions may be used to

modify a user's permissions as well. These include things such as whether or not the user is allowed to update the repository, or whether or not a user's permissions propagate up to his/her manager.

Integrating with IDEs

CM+ supports integration with several IDEs, including Visual Studio, Visual .net, Rational Rose, Eclipse and others. Any IDE which supports Microsoft's SCC API can be used with CM+. As well, Neuma supports integration with the Windows File Explorer, allowing those with less technical backgrounds to have some configuration management capability without learning how to use a new tool.

Unlike most IDE integrations, Neuma's IDE integration is change-based. Checkouts are associated with Updates, and Updates may be checked in as a single operation, rather than having to check in file by file.

Data Export and Import

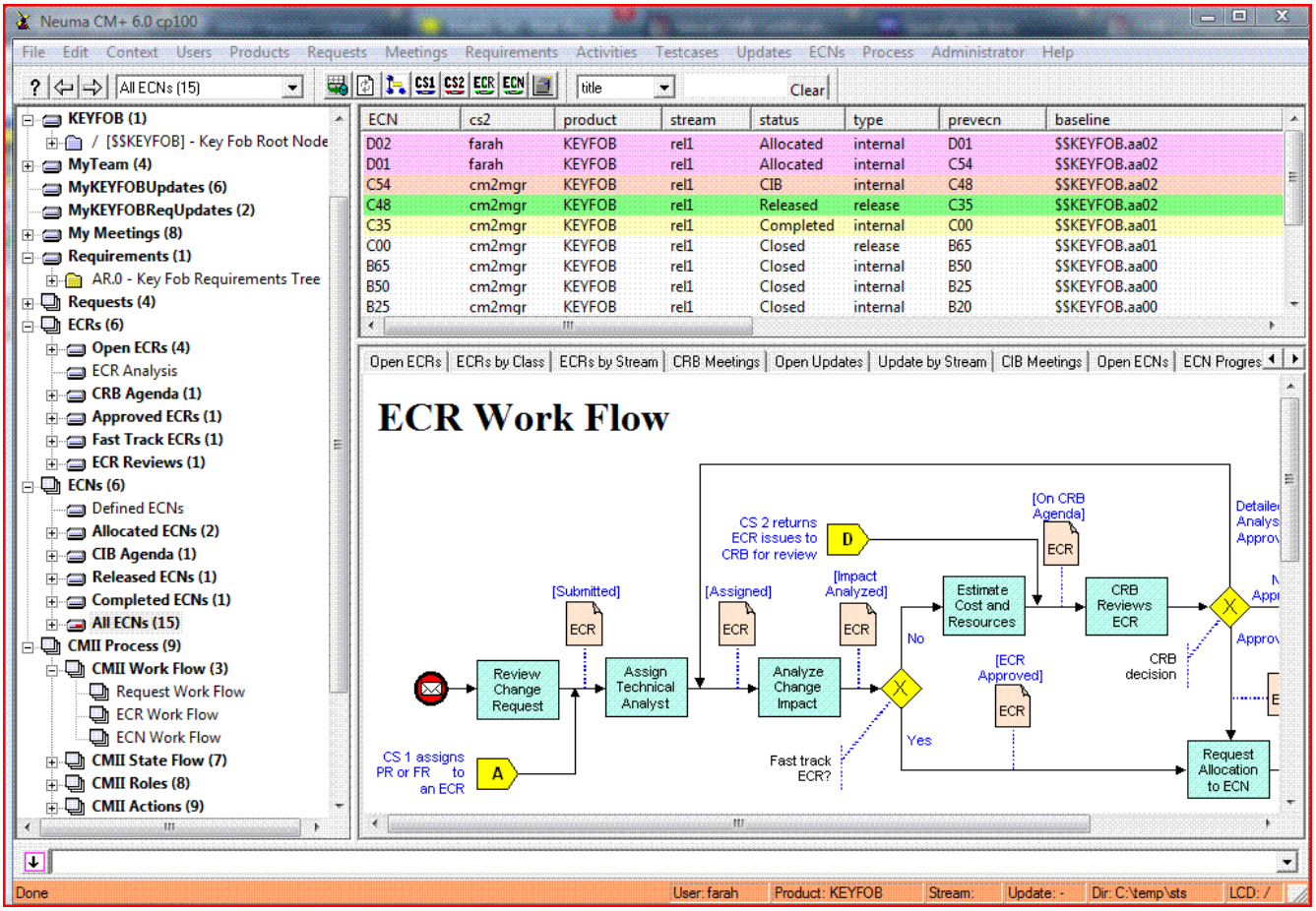
CM+ allows a variety of export formats for its data. Data may be exported in text format, spreadsheet format, HTML, or XML. Similarly, CM+ can import data in a variety of formats including text and XML formats, and some spreadsheet formats (others requiring an intermediate step). As well, CM+ allows the export of diagrams in a number of formats, including Postscript and HTML5.

Imported data may be processed in any manner (i.e. it does not have to be saved in the repository), and may include the use of 3rd party tools to process the information.

User Interface Overview

The CM+ User Interface is a combination of pre-defined, semi-custom and fully customized panels which are generated to support your ALM processes.

The main screen provides a familiar tri-pane appearance, with a browser panel on the left, a data panel on the top right, and a reporting panel on the bottom. The reporting panel is a tabbed panel on Windows platforms, and may or may not appear as a tabbed panel on other platforms. The main screen also includes a Status Bar, a Tool Bar, a Data Filter, a Browser Pick List, and a number of other features.



The browser panel contains a number of tree browsers, including the source tree, product tree, and perhaps an organization chart. It also contains a number of To-Do list items which are determined based on assigned data and your roles. Additional items might include risk lists, newly created items and process guidance.

The data panel displays the information selected in the browser panel. The information may be color-coded to reflect status, priority, etc. The data filter above this panel is used to restrict the amount of information shown. Clicking on a data panel item will cause additional details about the item to be shown in the "Notes" tab.

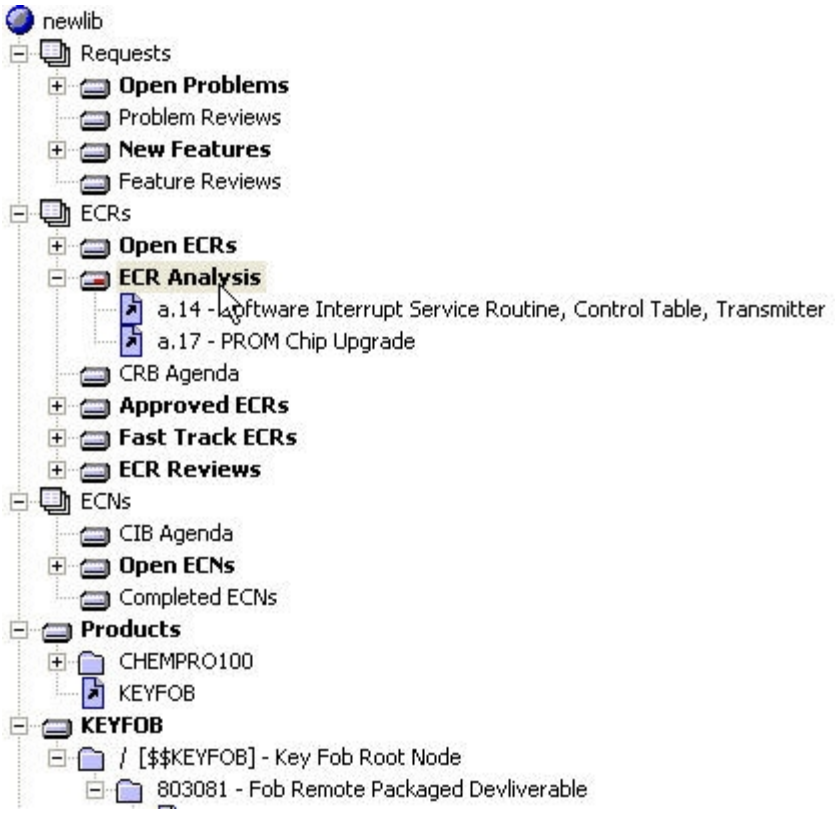
The tabbed panel provides a number of pre-populated reports, interactive graphs and summaries, and areas for process help, search results, system messages, and a number of other customized areas.

CM+ has an object oriented interface, right-clicking on an object, whether in a chart, a tree, the data panel or on a form, will cause a pop-up menu to appear with a list of actions that may be performed. Often, left clicking on an object will cause a drill down into more details, such as with a cell of a table summary, or a key or reference field in a display panel.

Across the top of the display, the pick list can be used to bring focus to a particular browser from the browser pane, with the corresponding data shown in the data pane. The iconic tool bar has Commit and Refresh buttons, along with a number of other buttons, normally customized to launch dashboards.

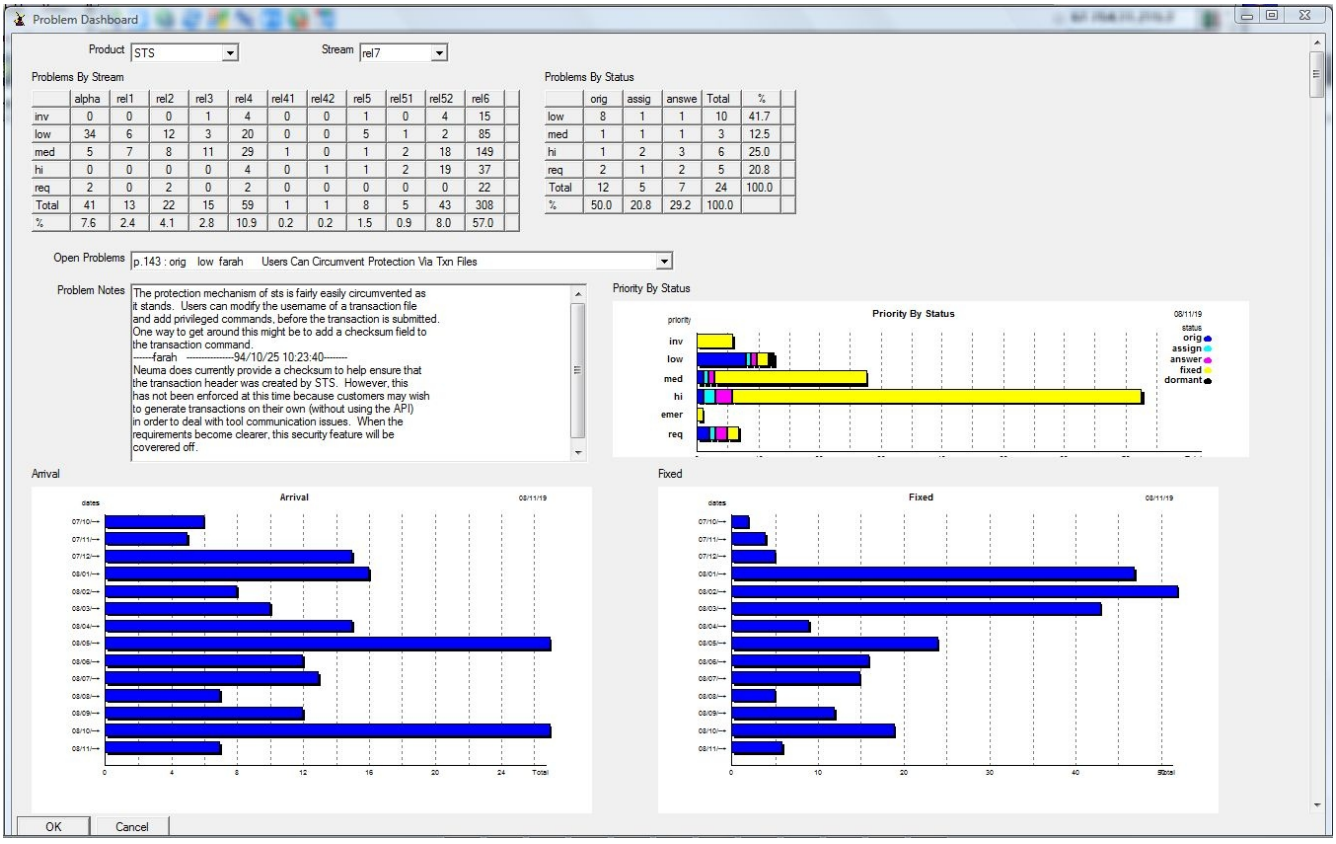
The status bar, along the bottom of the main panel, is used to identify key parameters of your context setting, including your working directory (Dir:), your Product and Stream context, the current User, and any particular focus items such as a Build or Update. The status bar is color-coded so that events dealing with the status of the CM+ client/server relationship may be easily highlighted.

The user interface is role-based. Depending on your roles and the customization of your site, your menus, tabs, to-do lists, browsers, tool bars, and status bar may have differing content.



Dashboards and Work Stations provide areas where interactive status summaries can be comprehensively presented, and where specific tasks can be accomplished. A project dashboard might allow you to select a product, and stream, and show you the status of activities, problems, changes, builds, etc. for that product stream, allowing you to change product and/or stream at will, to monitor a different development area. A typical project dashboard might have items such as problem arrival and fix rate graphs, activity summaries by priority and status, gantt charts showing Build/ECN progress, and a display panel of changes which can be used to zoom into details or to chase down integration links. All graphs, charts, display panels, etc. are interactive allowing drill-down and other data navigation actions.

A typical work station might present delta reports for a set of updates, selected by user, product and stream. Delta report options would likely appear on the work station, and an area for review comments would typically also appear. There may be additional buttons for electronic approval or for other specific actions.



Some text windows will allow popup menus to drill down to the details of a line of text. For example, a search results window might have a popup action that allows you to open a file on the line where the search match occured. Some, such as delta reports, may use color to highlight changes or significant terms.

There are hundreds of panels that might pop-up in your application suite. Each is customized using the GUI Customization capabilities of the STS Engine. Where a form has extraneous, confusing or insufficient information, make your administrator aware so that changes may be quickly applied.

CM+ Usage Model

General Process Flow and Traceability - Request and Test Perspective

The CM+ usage model supports assignment and role-based to-do lists and approvals, in the context of end-to-end traceability. To support this, the following basic work flow is in place.

(1) Raising a change request. Requests may be raised by, or on behalf of, the customer. These are tracked against the customer. If the request can be handled by the help desk, it is. Otherwise, it is forwarded on to product management.

(2) Requirements and Non-Conformance. Product management manages a set of requirements and a set of problem reports (non-conformance items) against those requirements. Requirements may be renegotiated on a per-release basis, so that significant evolution can occur between releases. In an Agile environment, these "release" cycles may be kept very small (down to a few weeks), to allow for the customer to provide feedback on interim releases.

(3) Change Request Approval. When Product management receives Customer Requests, they are each considered in turn. If a request is a problem report that the help desk could not handle, an engineering problem report is raised, and assigned an appropriate priority which helps to determine when it will be addressed. If a request is a feature request, it may be folded into the current release, requiring a modification to the existing requirements baseline, or it may be channelled to a future release for which the requirements are still being negotiated.

In the former case, a change request activity (or CR activity, or feature activity) is generated in the current release with a link to the requirement which is being modified. The activity clearly identifies the scope of the change in the current release under development, and is used to update the requirements. In the latter case, the request may be handled the same way (i.e. generating a feature activity for a future release), or it may be handled directly as a new requirement from which a Work Breakdown Structure (WBS) of release activities will be compiled. Typically, this depends on where the requirements phase for the next release stands at the time the request is reviewed. In any event, the activity or problem report is always linked back to the request object, for future traceability.

(4) Problem and Activity Approval. Problems and Activities are approved by the product management team and targeted to a release stream for implementation. Only when approved, does the problem or activity go to the engineering team. When problems or activities are approved, the corresponding requests are updated to reflect the approval.

(5) Updates (Change Packages). Approved problems and activities are assigned by the software management team to individual designers for implementation. Each problem or activity is addressed by one or more software updates, linked to the problem or activity, with each update performed by a single developer. It is quite plausible to have multiple developers work on an activity in separate updates. The developers are expected to follow appropriate guidelines to ensure that the work performed in each update is, as far as possible, independent of other updates, and where it is not independent, the dependency of one update on another is recorded as a prerequisite. Updates which have been successfully integrated into the release no longer require explicit prerequisite tracking. As well, updates which have an implicit prerequisite, by virtue of the fact that the same file is being changed again, do not require an explicit prerequisite.

Updates can only be created by referencing an approved activity or problem report. When an update is created, the activity or problem report state is updated to reflect the fact that the implementation phase has begun. At this point, various files may be checked out of the repository against the update, creating new revisions of each file which are linked to the update. A checkout operation should be performed as close as possible to the point in time that the file is actually going to be modified by the developer. This helps to minimize parallel checkout requirements. Even if a single developer is performing all of the work, it is wise practice to, when necessary to minimize checkout duration, split one planned update into two or more updates

which can be individually tested, reviewed and checked in.

If the development changes are being performed by a third party, an alternate path may be followed from this point forward through to step 9.

(6) Reviews and Update Checkin. Developers perform unit testing, document such testing as part of the update, and have a peer review performed on the changes made to the software as part of the update. When testing and peer reviews are successfully completed, and documented as such, the Update may be checked into the repository. At this point, the problem reports and/or activities associated with the update should be modified to reflect the fact that implementation is complete, but only if there are not additional outstanding updates referencing those problems and/or activities. Typically, the requests referenced by these problems and/or activities are also updated to reflect an "implemented" state, although sometimes this action is held off until a successful integration has been completed.

(7) Ready for Integration. When all of the updates required for a problem fix or activity implementation are completed, or if the updates can be integrated piecemeal and an appropriate subset is completed (typically one update), these updates are marked ready by the developer. This indicates that the updates, already in the repository, may be selected for integration into a system build.

(8) Creating a Build. Typically a new build is created nightly and sanity/integration testing is performed on the build prior to the start of the next day. This is done by selecting either all of the ready updates, or some specific subset as determined by the build team in conjunction with product management directives, The selected updates are promoted to the "select" state which is used to define the build record, which identifies exactly what is going to be in the build - that is what baseline, and what additional updates, are used to create the build.

Once the build is successfully built and sanity/integration tested, it is promoted to that state, typically a "system integration tested" state, and the newly selected updates are also promoted to a similar state, with the build number attached to such newly selected updates, indicating the first build to integrate these updates successfully. However, if the build fails, it must be fixed, by some combination of additional updates (perhaps reopening a problem report or else creating a new one) or rollback of selected updates back to the ready state (and perhaps reopening the associated problem report or feature, or creating a new problem report). Any rolled back update must be quickly remedied or else rolled back to the "checked in" status until such time that a fix can be applied.

(9) Build Verification. Occasionally a build will be delivered to the verification team to ensure it passes preliminary feature testing, performance testing, and perhaps regression testing. In cases where development is performed by a third party, the product engineering team is receiving a set of binaries (i.e. the "build") from the third party, and must ensure that the following have been performed:

* A list of updates identified, each linked to the problem report(s) and activity(s) addressed.
* For each update, proof of unit testing and peer code reviews, both showing successful completion.
* Documentation of a successful integration/sanity test of the build being delivered.

Verification must record the set of tests to be run, that have completed successfully and that have failed, as part of each test run. Each test run will also identify the build against which the tests were run. In some cases, where failure rates are high, verification may be abandoned for the build. In other cases, it may still run to completion to help identify all issues. When the verification team puts the build through its test suite successfully, the original requests, along with the problems and activities addressed, and the software updates used to address them, should be promoted to reflect the tested status.

(**10) Traceability and Promotion.** Testing may run through multiple levels, including initial system testing, in-situ testing (e.g. flight worthiness test), and perhaps field trials. According to each product's requirements, additional promotions should be performed on the build records to reflect such testing. Such promotion may or may not be reflected at each level on the earlier artifacts. In some cases, promotion of these artifacts to the tested status may be delayed until all such test levels have completed satisfactorily.

(11) Build Promotion. It is quite normal to see a number of builds cycle through testing before one actually reaches a production (or release candidate) status. Typically, at this point, a beta release is performed to

ensure that there is full acceptance in production use prior to a wider distribution. And typically, changes are made and a new build is produced prior to wider distribution. For this reason, it is important to distinguish between test levels, beta testing, and production promotion levels in a build record's state flow. When a production state is reached, the original request should be updated to reflect that fact that the request in now available for distribution. Normally, it is not important to identify this state in the interim artifacts (i.e. between, but not including, the request and the build).

(12) Deployment Tracking. Deployment tracking may be done in the CM repository, or as part of an asset/deployment repository. In either case, it is important that the deployed build can be readily identified on the deployed target, whether it be a computer, a helicopter or otherwise. The build process, along with the appropriate target design, must ensure that the deliverables are tagged with the build record id and that the tags are accessible on the fielded units.

General Process Flow – Project & Configuration Management Perspective

(1) Requirements Baseline. Configuration management starts from a requirements baseline. Changes to the requirements baseline must be tracked and used to create new baselines. Typically, a requirements baseline is used to give marching orders to a development engineering team.

A requirements baseline evolves within each product release stream. Each new release effort starts out with the delivery of a requirements baseline for that planned release. During the course of the release effort, requirement changes may occur. These are identified by updates tagged with the appropriate product release stream (stream for short). Typically, a number of updates for a stream are collected over time, and a new baseline is created to reflect the collection. This baseline is usually delivered to he development team as a requirements "delta" showing the changes to the requirements that have to be factored into the current stream.

Typically a baseline of requirements is frozen every time it is to be delivered to the development team, and after it has been approved by the customer or product management team (representing the customer). However, the process of customer approval may necessitate intermediate baselines.

(2) Product Requirements. The requirements, known as the customer requirements, may be specific to a customer, or generic to a market. In either case, it is necessary that the requirements are allocated from a "must, should, want" perspective, to a product specification ("this is how the product will work"). This specification should be primarily functional, with a full decomposition down to the last user-affecting detail.

Often, the original requirements are specified as a product specification and no allocation between customer and product is necessary. If the requirements are presented as Customer Requirements, the allocation to Product requirements must be completed in a specification phase. Prior to the start of the specification phase, an initial effort sizing is performed on the Product requirements to get an estimate of the resources/cost required to complete the requirements. This helps to ensure that the customer and development team have the correct scope in mind. Over time, these estimates will become more accurate, allowing budget allocation to be completed early in the cycle. Initial attempts may require completion of a specification phase prior to realistic budgets.

(3) Product Specification and Planning. The specification appears as a list of features, typically organized hierarchically by function. It is a best practice to make these features act as both a specification and an activity/task which can be prioritized, scheduled and assigned. In some cases, the scheduling may be broken into phases: specification date, design date, implementation date. More frequently, the specification tasks are scheduled and a new set of tasks are created for the design phase and possibly again for the implementation phase, though these two are frequently combined as separate milestones on a given task.

Once the specification tasks are completed, detailed sizing is performed for each feature. At this point in time it should be possible to specify accurate completion dates with a given set of resources, and the specification can then be used as the basis for a contract. Note that in some organizations, both specification task descriptions and design task descriptions are referred to as System and Software Requirements respectively. In effect, this is accurate. However, unlike Customer Requirements, the development team has a large

amount of control over these, and is responsible for the project management of these tasks, including prioritization, assignment, reviews, etc. For this reason, it is more effective to treat them as tasks with the associated documentation attached to such tasks.

(4) From Specification to Design and Implementation. Allocation of specification tasks into design tasks is sometimes a complex procedure, depending on the system architecture. In some cases, the specification tasks map 1 to many onto the design tasks. But in some cases, especially middleware intensive systems, the specification mapping is a many to many mapping process. In such cases, there should be a "mapping" exercise performed at the end of the specification phase (or at the end of each specification phase in a more agile environment). The net result is the same in either case - a set of design tasks to be performed. These may be done collectively in a design phase, or iteratively across design tasks in a design and implementation phase. In most environments, each design task will include a "design spec" and an "implementation" milestone tracked with a single task object. However, they can be split up into separate tasks if desired. The design/implementation tasks form a design WBS.

(5) Implementation Using Updates. Design tasks start with the creation of a design spec (or, more frequently, an incremental design spec) and then spawn a number of Updates for implementing the design. The design spec author is generally considered the "lead" on the task, but may employ a team to complete the implementation. Updates are created with a number of key guidelines.

1. An update is a unit of change performed by a single developer.
2. Generally, an Update will address only one design activity, to maximize traceability.
3. More than one Update per design activity is encouraged when this helps reduce risk and bottlenecks.

For example, if one of the files of the update has significant checkout contention, an initial update with the changes to that file, and perhaps stubs if required to support that change, can be performed as an initial part of the update. Or if a design has a risky element to it, the risky portion can be packaged in an update separate from the non-risky portion. For a complex design task it is sometimes better to split the work across a sequence of updates that can successfully be integrated using only the previous updates of the sequence.

(6) Quality: Unit Testing and Code Reviews. There should be a level of quality assurance to make certain that code reviews and unit tests (where the “unit” is the Update) are being performed correctly. Guidance should be provided by the design team, and the CM repository should hold the contents of such tests and the results of the reviews. These two steps are probably the most effective means of reducing field problems, when performed properly. Good quality will simplify verification structure and reduce the amount of branching and merging required, allowing updates to be made against the stream's main branch.

(7) Build Cycles The Build team will normally schedule an ongoing, regular compile cycle, such as a Nightly Compile, to verify that checked in updates will integrate cleanly. In a typical shop, the process is set up to automatically include all updates which have been marked "ready" for integration by the developer. These are selected for the Nightly Compile, and any errors are addressed in a priority fashion. The resulting build is sanity and integration tested to ensure that the run-time environment will be adequate for continued development support.

As a release approaches, the set of approved problems and feature activities for a release stream are restricted to ensure stability of the build that results from slower change rates. Typically, at this point, work on the next major release cycle is already in progress, possibly in the specification phase. The build team may also have builds at other levels (other than the "ready" level), but this generally is the case only if poor quality results in a significant number of updates being rolled back from nightly builds. Otherwise nightly builds can be promoted to serve the higher levels of verification testing and field trials by more carefully controlling the work approvals for the release stream.

(8) Builds, Baselines and Variants. Typically, the Build/CM team will establish a number of milestones along the course of the development streams. At these milestones, baseline configurations will be established against which future change can be measured and future builds built. A Build is defined roughly as a baseline plus a set of changes. The Build may often be a subset of a baseline, or a particular variant established by including the appropriate components and/or variant updates. Multiple variant builds may result from a given baseline, depending on the level of run-time configuration capability and customer requirements.

From a CM perspective, it is imperative for product design to allow for a generous portion of run-time configuration so that numerous build variants are not required. However, in some cases this cannot be avoided (e.g. different run-time platforms). To define a baseline, a product is selected, a stream is selected, and a promotion level is selected. CM+ will create a configuration reflecting these variables. As well, a number of options may be considered, such as whether or not bug fixes/enhancements to previous streams are to be automatically rolled into the stream, and whether or not configuration alignment is specific to the product proper, or also to its sub-products (i.e. are the sub-products aligned separately or at the same time).

(9) Production Build. The Build process clearly tracks those builds which are candidates for verification, those that have successfully been verified, and those which are production ready. A build that is production ready is said to sit on the "product shelf", ready for distribution.

(10) Distribution and Deployment. Distribution may or may not be part of the CM team role. If so, the CM library should maintain a list of deployment sites/targets and the current build level at the site/target. Ideally, the deployment history is also maintained.

CM+ Run-Time Overview

CM+ can be run with as few as one server, and as many as all five. Typically, all servers are run on the same computer (per site). Multiple libraries, and hence multiple sets of servers, may be run on a single server computer.

Server licensing is by machine. To run the CM+ Library server on multiple machines requires multiple server licenses. However, other servers may be spread across machines where appropriate.

There are 5 types of server processes for CM+:

1. Neuma License Server
2. CM+ Library Server
3. CM+ Transaction Server
4. CM+ Query Server
5. CM+ Web Server

Here is a brief description of the server processes.

1. Neuma License Server

The License Server is used to support Floating Network Clients. The license server, on a given network or network partition, will support floating licenses which determine the number of concurrent users that may be using CM+. This is in addition to any Fixed User Clients, which do not require the license server for use. The license server handles checkout and checkin of licenses, and automatically reclaims licenses which have gone missing (e.g. someone with a license checked out disconnects from the network). In the case of CM+ MultiSite, there is a License Server and a separate license key file for each site.

2. CM+ Library Server

The Library Server's sole purpose is to update the repository based on transactions found in the "txn" (transaction) directory. The library server does not directly communicate with clients. (All client communication with the repository is either through the file services of the OS, or through the transaction or query servers.) The Library Server processes transactions from the txn directory until it has processed the latest transaction, indicated by the "txnid" file within the txn directory. The Library Server should be the ONLY process updating the repository files: <library>, *.TEXT, *.NOTE, src directory, notes directory. In the case of CM+ MultiSite, there is a Library Server at each site processing the same transactions in the same order.

3. CM+ Transaction Server

The Transaction Server, or txnserver, is the primary means of communication between the server machine and the client. Because the client is a smart client, it may perform all queries without the need for a txnserver. However, in normal practice, all updates to the repository occur through the txnserver. [In "nfs" mode, the client itself can take the place of the txnserver, but this requires write permissions to the txn directory for the clients, and this is generally considered a bad idea. Likely the only time you will see a client running in "nfs" mode, without a txnserver, is for the demo library, which by default permits write access to the txn directory because it is generally run as a single-user demo.]

The transaction server is responsible for placing transactions into the txn directory and for assigning a unique sequence number to each transaction, the transaction id. It is also responsible for handling client queries regarding the state and results of a transaction.

The txnserver may be run in either "net" mode (single site) or in "multi" mode, for CM+ MultiSite. In "net" mode, CM+ will support the use of multiple transactions servers running concurrently. Normally, a single txnserver is used for simplicity. In "multi" mode, there is effectively one txnserver running at each site. The txnserver spawns a process to handle each connection. On Windows CM+ MultiSite platforms this can cause congestion problems. So, for CM+ MultiSite on Windows, there are actually a number of different txnservers running: one to handle site client requests, and one to handle messages from any other sites with which the site needs to communicate. [This happens on Unix as well, but through an automatic spawning of the txnserver process.]

For CM+ MultiSite, there is a difference between the "Master" (or Main) site and "Slave" sites. The Master site is the only one responsible for assignment of transaction ids. This is then communicated to the other sites. This ensures that all transactions will run in the same sequence at all sites. Transaction id assignment is completed and communicated before the transaction is processed. If, for some reason, the Library Server is not running at the Master site, it is quite likely that transactions will be processed at other sites first. Once the Library Server is started at the Master site, it will process the pre-assigned transactions in order.

The txnserver at the Master site is also responsible for distributing transactions to all slave sites. [A future configuration will permit a hierarchy of Slave sites from the Master, so that intermediate nodes, rather than the Master, can communicate with the "leaf" nodes of the hierarchy.]

Any communication between any two sites is queued up for delivery in case of low bandwidth, network congestion or network outages. In situations where one of the sites is on a laptop, it is quite common for queues which are several days or weeks long to occur. The txnserver ensure that the queues are properly distributed when a reconnection occurs.

There are separate processes for handling client requests and for de-queuing requests from other sites. On Unix, the separate processes are spawned as needed. On Windows, the txnserver is invoked once for client requests, and once for each site with which the local site will communicate directly. Generally this means that the Slave sites have a single dequeue process, to communicate with the Master site, while Master site has a dequeue process started up for each Slave site.

4. CM+ Query Server

The CM+ Query Server is similar to the transaction server, but handles queries only, instead of updates. This includes the transfer of all files from the server to the client. The query server eliminates the need for direct file system access to the repository. However, it is generally slower than the OS file server, and implements its own caching strategy for each client. Generally, the Query Server is used only to support Remote Clients which have TCP/IP access to the network, but not file system access. This is generally, though not always, the case when there is a low bandwidth connection between the client and the Library Server machine.

In more recent years, the use of VPN (Virtual Private Networks) have significantly reduced the requirement for use of the Query Server. The VPN allows, what would be otherwise remote clients, to operate as normal clients with full file access.

5. CM+ Web Server

The CM+ Web Server is a Web Server along with a set of Perl scripts which allow communication with the repository through a local client running (in the background) on the Web Server machine. You may have multiple Web Servers on any number of machines running for a given CM+ configuration, although generally, a single CM+ Web Server is configured, if any.

Because of the use of Thin Client technology (Remote Desktop and VNC, as examples), the use of the CM+ Web Server to support users who are sometimes away from their local networks is rare. Such users prefer the use of the richer Fat client through Remote Desktop, VNC or other remote computer access technology.

However, CM+ Web Server is still a useful option for presenting a subset of functionality to others who reside outside of the network, such as customers or sub-contractors who need only partial functional access to the repository.

The CM+ Web Server defines the user interface for each user or each group of users. This is done through a set of customization files which generally indicate which To-Do lists and what Menus/Menu Items appear on the web client interface.

As of this writing, the CM+ Web Server is supported only with the Apache Web server on Linux platforms, though Neuma is working at wider platform availability. The CM+ Web Server may also run in a Virtual Machine (e.g. on a Windows client or server machine).

Deployment Strategy for CM+

Once initial evaluation of CM+ has been completed and a decision has been made to acquire CM+ for your project, there are a number of steps that must be taken to deploy CM+. It is expected, though not absolutely necessary, that a Neuma support person will spend 2 or 3 days on site, helping you with your initial deployment and on-site training. Unless you have complex requirements, such as transferring all existing revisions from your existing CM tool, or extensive modifications to the out-of-the-box process that need to be completed before user training, you should not require extensive consulting and implementation services from Neuma.

Deployment consists of seven basic steps:

1. Understand the scope of CM+ functionality you wish to use initially
2. Identify initial projects that will use CM+, including, if applicable, projects across multiple sites.
3. Install CM+ on the target server(s)
4. Load in your existing data (apart from files)
5. Load in your existing software product baselines
6. Adjust the Pre-defined CM+ processes and user interface
7. User Training

Steps 1 to 3 should be done prior to visits by Neuma. As well, preparation for steps 4, 5 and 6 should be done prior to a Neuma visit. If you wish, you could attempt these steps on your own, but you might want a bit of training before trying to adjust your processes and make any associated user interface changes. Step 7 includes both end user training and CM manager training.

These steps should be used as a guideline for deployment. Once you have made a decision to purchase, you should work with Neuma to ensure that deployment will be both low risk and swift.

Scope of CM+ Functionality

The first step in deploying CM+ is to decide which parts of CM+ are going to be used initially. Neuma recommends the use, at a minimum, of Problem Tracking, Change Management, Version Control and Build Management. We also recommend at least very basic Activity Management. This could be in much the same form as for Problem Tracking (i.e. activity title, objectives, priority, [target development] stream, owner and assignee), or it can be more elaborate if you prefer.

The fact that you use Problem Tracking or Activity Management initially will not increase the learning curve significantly. Users will most likely just see their "To-Do" lists and click on them and select operations from the objects displayed (e.g. Fix Problem, Reply to Problem, Implement Activity). These will lead them through to the rest of the Change Management functions while ensuring traceability.

You may want to delay extending your processes into new territory initially (other than the above) unless your processes are already well understood by the development team. For example, if you're already using Requirements Management, but doing so by publishing a spread sheet, it might be good to move this function into CM+ sooner than later. You may even wish to delay Change Management and just start out with a simple Problem Tracking to get your users used to the tool. However, if they are not doing a lot of Problem management, that might not help.

When considering the scope of CM+ functionality, it is important to identify any functionality that you have not had a chance to evaluate. If this is key functionality for your site, it's important that you understand how it works. Most CM+ functionality is highly configurable. But some is much less so. For example plug-ins or interfaces to other tools, which are restricted to working with 3rd party interfaces, will naturally have a lower level of configurability. In any event, make sure key functionality meets or exceeds your expectations before committing to it. If there are areas that need improvement, discuss this with Neuma as Neuma is usually in a

position to address your requirements in a priorty fashion.

In any event, remember that CM+ has been designed so that both the process configuration and the user interface can easily be changed over time. This allows you to select an appropriate starting point and move forward from there over time.

Identify initial projects to use CM+

One of the more difficult decisions is to decide which projects are going to be the first to use CM+. We recommend selecting projects where the participants are well suited for helping with the migration of CM+ to other projects. You should expect very little disruption in your migration from your current CM methods to the use of CM+ (unless you're making sweeping process changes at the same time, in which case there will be some process learning curve). However, we would also recommend that you select projects that are at reasonable cut-over points (e.g. end-of-release, new project, lull in development, etc.).

Although CM+ has been designed to match most projects well, not all projects will be a perfect match for CM+. For example, a combined hardware/software project may raise more issues than a pure software project. When considering your initial projects, select an initial one that can be deployed easily. Leave more difficult projects to subsequent deployments. This allows you both to improve your training in CM+ before having to address more difficult issues, and provides a successful model for use in other projects. And remember that the difficulty level is not often a technology factor, but is often a personnel factor, whether on the management side or with the team member buy-in (e.g. resisting change or familiarity with older less functional tools).

Install CM+ on the target server

Neuma recommends a Windows, Linux or Solaris server for CM+, although any Unix server will do (Windows, Linux and Solaris have the most advanced user interfaces at this point in time). The power of the server is not really important until the number of users rises above a couple hundred, provided that the server is not already nearly fully saturated with other applications running.

Neuma also recommends that the server is one to which your clients have read access through the file system (e.g. NFS or SAMBA, or native Windows networking). This eliminate the need to have to maintain multiple CM+ environments (including customizations and upgrades), and provides for a single machine install operation (instead of an install operation for each user workstation).. Instead, a single environment per site should be sufficient.

Finally, Neuma recommends installing CM+ using the default settings to simply any support communications. However, this is not necessary and will not affect operation or performance. Installation should be a 5 minute process.

Loading in your data

There are a number of steps to loading in your existing data, but they are all straightforward. Remember, when you do bulk loading of data, it is a good idea to first disable your Automatic Save ("autocommit") feature, using Edit | Preferences | Options (and disable the "Automatically Save" option), and to view the results before you explicitly commit them to the repository. Generally, it's easier to fix up your data entry file than to have to re-do or correct parts of it. If you have errors or are not satisfied with your results, use the File | Cancel menu item to cancel your transaction and to try again.

You may have data in other repositories that you wish to continue to maintain in the other repositories for any

number of reasons. We first point out that having the data integrated into a single library will give you much more capability than having it separate. This includes both data navigation and handling cross-application process issues (e.g. problem promoted when update submitted). So in these cases, you may want to consider one of a number of options including:

- Migrating the data to CM+
- Replicating part of the data in CM+ (e.g. title, priority, assignee, status), and maintaining that replication.
- Exploring the level of integration between CM+ and your other tool. This will likely be more expensive and will not result in seamless integration (and hence will be less user friendly) but may be your hard requirement.

Data Loading

There are a number of steps involved in loading in your library data. These steps are generally quick and easy to do. However, this really depends on how easy it is to get the data out of your existing repositories. If you can export data in an XML, CSV or Text format, it should be relatively easy to load it into CM+.

The loading process involves the following:

1) Load in your Users
2) Load in your Activities/Tasks
3) Load in your Problem Reports
4) Load in your Source Code
5) Load in your Requirements
6) Load in your Test Cases
7) Load in any Other Data

Software and Requirements may be loaded for multiple baselines. Generally to do this, you load in your older baselines first, with the most recent last. For Software, there are automated procedures for loading in the baselines and there are multiple query screens for helping to identify changes across baselines (e.g. Is this a new directory or has it been renamed?). All other data is easily loaded through simple (generally one line) scripts involving the “runfile” command of CM+. Large objects are generally dealt with as separate files durnig the loading process. Care may be needed to properly export traceability links.

Loading is generally performed product by product. This may be done all at once or over a span of time. Neuma recommends some support in this area if you have not been trained on writing STS Engine scripts.

Adjust the Pre-defined CM+ Processes and User Interface

CM+ comes pre-customized for a fairly reasonable set of processes. However, you are likely going to want to tweak these processes. We recommend that you first view the process flow diagrams (Process | Process Flow | <Object> Process | View Process Flow) and identify the changes you might like to see. Keep in mind that there are a few states that are not so easily configured as they are referenced in various triggers, rules, scripts, etc. But these are few and generally will not need to be renamed. Other state flow changes can be identified and Neuma will review these changes and either apply them or help you to apply them. Once you have identified your candidate changes, Neuma can likely make the changes in a few minutes to a few hours. Process may be modified at any time, it does not have to be perfect to start. What we do recommend is that the process that affects most of the users be near perfect so that there is not the need for significant incremental process training for your users.

You may wish to use the process flow diagrams to inspect and possibly adjust the process flow in CM+. There are two ways to do this. One is to adjust the process flow in the "schema" files (see the bottom of the .sts files) where they are initially defined. This will allow you to define the schema for any new libraries you may create in the future.

A second way is to adjust the process directly in the library using the process flow diagrams for each type of

object. In this way, you are affecting only the processes for the specific library that you are adjusting. When you make your adjustments, you can export the changes for one or all modules using the Process | Schema | Dump... buttons. These can then be placed in the schema directory for use with the creation of other libraries.

The process definitions also involve things such as high-lighting (criteria and colors) and documentation of symbolic range elements (e.g. "hi" means that "it must/should be addressed before the next release"). These you may deal with at a later time when you're fine-tuning your CM+ library.

You may also wish to tweak the CM+ user interface. This will likely involve adding and/or removing a few menu items, to-do lists or other items from each role. CM+ default roles are most easily modified through graphical user interface (GUI) configuration. The GUI configuration affects who sees what menus, what's in each menu for each role, what menus are defined, what tree browsers and to-do lists appear in the top left window, and what the contents of each of the system's dialog boxes are.

Normally there are few changes. For example, one customer may prefer to retrieve files to the workspace by default when checked-out, while another may wish the default does not retrieve them. You may want to pre-can a couple of reports or metrics that aren't already available, or you may wish to simplify the menus for certain roles. These are all fairly straight forward operations that normally (unless you require extensive changes) will not require additional consulting costs from Neuma. The GUI can be customized by role, or even at a finer level (e.g. user) where it might be necessary in some cases.

Some customizations are easy to perform without affecting existing Neuma-supplied GUI configuration files. This makes it easy to incorporate future advances in Neuma's GUI configuration (although you're always free to ignore future changes to the GUI configuration). Other changes may need to be re-applied with each release of CM+. The CM+ merge facility can be used to automate most of this work.

Yet in other cases, you may wish to add additional fields to your data schema as part of your initial customization. These might be necessary to define your process or user interface more specifically. As will all other changes, these may be done prior to deployment or at any time subsequent. Some customers perform continuous improvements, while others like to save them up and hold a user session when a significant number of changes are being made.

User Training

The last part of the deploy procedure deals with training your users. Neuma offers training courses on the basics of using CM+. There is a 1/2 day user course which covers Problem Tracking, Change Management/Version Control and Context/Workspace Management (plus a touch of Activity Tracking). This is intended to be customized to each project as necessary. It can be delivered by Neuma, or can, through the Training Kit, be licensed for delivery by the customer. The course should also be customized to work with a copy of your populated repository, although it can run on Neuma's default training library.

Training of Power Users is a full day course and covers more of the power features of CM+. It allows users greater flexibility in exploring CM+ and expanding their knowledge of CM+ capabilities. The goal here is for power users to gradually, over time, pass on the features they find most useful to their peers who have had minimal training.

The other aspect of training that should be completed up front is the Configuration Management and Administration course. This is a two-day course and it is recommended that at least two personnel take the course so that they can spare off one another (vacations, sickness, departure). That being said, once the basic set up of the CM process is completed (e.g. setting up nightly compiles to run automatically, etc.), the CM and Administrator jobs will be a part time effort, and might focus initially on bringing new users on board with CM+. It will also involve interaction with Neuma for specific support queries and customization requests.

Up and Running

At this point in time, your CM+ library(s) for the trained user base should be up and running. If you need to immediately provide CM+MultiSite capability, a few simple steps are required to replicate an initial checkpoint of your library for use at other sites. Once the replication is complete, your initial site can continue running and your other sites can be brought on-line as required. It is recommended that you run at least one other site to provide a warm stand-by disaster recovery. Neuma provides a specially priced server that can be used only for disaster recovery (i.e. no users are allowed at the site until it is switched to become an active user site).

Neuma recommends having a Neuma consultant on site for 2 or 3 days for your initial start-up and training, but this is not essential. Keep in mind, however, that switching to a new tool is a significant change for your team, and anything that can be done to make the transition easier is well worth it. Neuma's consulting rates are priced to make this affordable. If you require support at any time, Neuma can be reached most effectively by email at cmsupport@neuma.com, or through direct telephone support.

Prior to moving a library to live production, Neuma recommends that you spend time working through the CM+ Demo Tutorial Guide and that you run through the set of tasks you expect each role to be performing. It is best to get experts on each role to assist you so that you may receive initial feedback and fine tune the user interfaces to your specific requirements.