



MAKING CONFIGURATION  
MANAGEMENT WORK FOR YOU

---

ELECTRONIC SYSTEMS LABORATORY  
Georgia Tech Research Institute  
Georgia Institute of Technology

Authors: Jeanne Balsam, Mark Pellegrini,  
and Jean Swank

*This page intentionally left blank.*

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

## TABLE OF CONTENTS

<b>OVERVIEW .....</b>	<b>1</b>
<b>OBJECTIVES.....</b>	<b>1</b>
<b>CONFIGURATION MANAGEMENT OVERVIEW .....</b>	<b>1</b>
CONFIGURATION MANAGEMENT ORGANIZATION .....	2
PHASES OF CONFIGURATION MANAGEMENT.....	2
CONFIGURATION ITEM IDENTIFICATION.....	3
GENERAL BENEFITS .....	3
<b>WORK PRODUCT ORGANIZATION .....</b>	<b>4</b>
<b>BASELINE TYPES .....</b>	<b>5</b>
LEVELS OF CONFIGURATION ITEM CONTROL.....	5
<i>Informal Control</i> .....	5
<i>Formal Control</i> .....	6
CM BASELINE CONTROL PROCESS .....	6
<i>Configuration Item States and Transitions</i> .....	7
State Definitions.....	7
State Transitions.....	8
<b>BASELINE DOCUMENTATION .....</b>	<b>9</b>
BASELINE VERSION DESCRIPTION.....	9
PRODUCT VERSION DESCRIPTION .....	11
VERIFYING BASELINES .....	12
<b>CMS SUPPORT FOR AUDITS AND PEER REVIEWS .....</b>	<b>13</b>
<b>CONTROLLING REQUIREMENTS WITH THE CMS.....</b>	<b>14</b>
<b>CONCLUSION .....</b>	<b>15</b>

## LIST OF FIGURES

FIGURE 1: PHASES OF CONFIGURATION MANAGEMENT .....	2
FIGURE 2: DIRECTORY STRUCTURE TEMPLAEE .....	4
FIGURE 3 CM BASELINE CONTROL PROCESS .....	6
FIGURE 4 STATE TRANSITION DIAGRAM .....	8
FIGURE 5 EXAMPLE BASELINE VERSION DESCRIPTION .....	10

## LIST OF TABLES

TABLE 1: CONFIGURATION ITEM EXAMPLE AND GUIDELINES .....	3
TABLE 2: PRODUCT VERSION DESCRIPTION CONTENTS .....	11

*This page intentionally left blank.*

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

---

## OVERVIEW

---

The concepts of this paper relate to an established Configuration Management System (CMS), with automated tools that provide baseline capabilities, file revision retention, history logging, and ideally an integrated change control tool. Once a CMS is established, practices to support the efficient use of that system, including ease in establishing usable baselines and use of the CMS to support other process areas emerge as added value items.

Baselines, documentation of baselines, auditing of baselines, and change control for baselines are covered in the following sections at a high-level, detailing value added practices in these areas. Additionally, use of the CMS to support document audits and peer reviews, and control of requirements is discussed.

---

## OBJECTIVES

---

The above concepts are discussed with the purpose of passing on lessons learned about the importance of:

- 1) Well-defined baselines
- 2) Well-defined development and execution environments
- 3) Building, executing, and testing software on a “clean machine”
- 4) Change control throughout the product lifecycle
- 5) Configuration management in the audit and peer review process
- 6) Use of configuration management techniques to control requirements.

After reading this paper, the reader will have insight into how the practices described in this paper could benefit their current process or how they may be helpful in defining a new process.

---

## CONFIGURATION MANAGEMENT OVERVIEW

---

Configuration Management (CM) activities are a critical element in ensuring that the customer receives the required product(s). These activities are initiated early in the product lifecycle and continue throughout the product lifecycle. CM is not simply version control of work products (e.g., code, schematics, documentation).

CM involves identifying the configuration of the product (i.e., selected work products and their descriptions) at given times, systematically controlling changes to the configurations, and maintaining the integrity and traceability of the configuration throughout the product life cycle. The work products placed under CM include the products that are delivered to the customer and the items that are identified with or required to create these products (e.g., the compiler, drawing package, support utilities, non-deliverable documentation).

Within the context of this paper it is assumed that informal control of work products, using the CMS, begins at work product inception. Formal control begins after the work product has been baselined or as defined in the Configuration Management Plan (CMP). Criteria for transitioning from informal to formal control must be defined in the CMP.

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

## CONFIGURATION MANAGEMENT ORGANIZATION

A “Configuration Management Organization” must be defined whether for the entire company or for a product specific development. It is important that senior management or their authorized designee:

- 1) Establishes tools for CM use.
- 2) Reviews activities to verify that adequate resources, funding, and process adherence is in place.
- 3) Appoints a Configuration Manager.
- 4) Designates a Configuration Control Board (CCB).
- 5) Identifies CM training requirements.

The Configuration Manager documents and maintains the CMP, creates and maintains the CMS, assigns document numbers, establishes and updates baselines as directed by the CCB, creates product and associated version release control documents for each baseline from the CMS as directed by the CCB, manages access to the CMS, maintains CM activity records, reports on CM activities, periodically audits product baselines, and supports external (QA or assessment) audits of CM activities.

The CCB represents the interests of the stakeholder for the product. The CCB authorizes the establishment of baselines, identifies configuration items (CI), and levels of control, reviews and authorizes changes to baselines, and authorizes the creation of products from the CMS.

## PHASES OF CONFIGURATION MANAGEMENT

The major phases of CM include: Initiation, Monitor and Control, Baselining, Baseline Control, and Release. Figure 1 presents a very simplified view of that process.

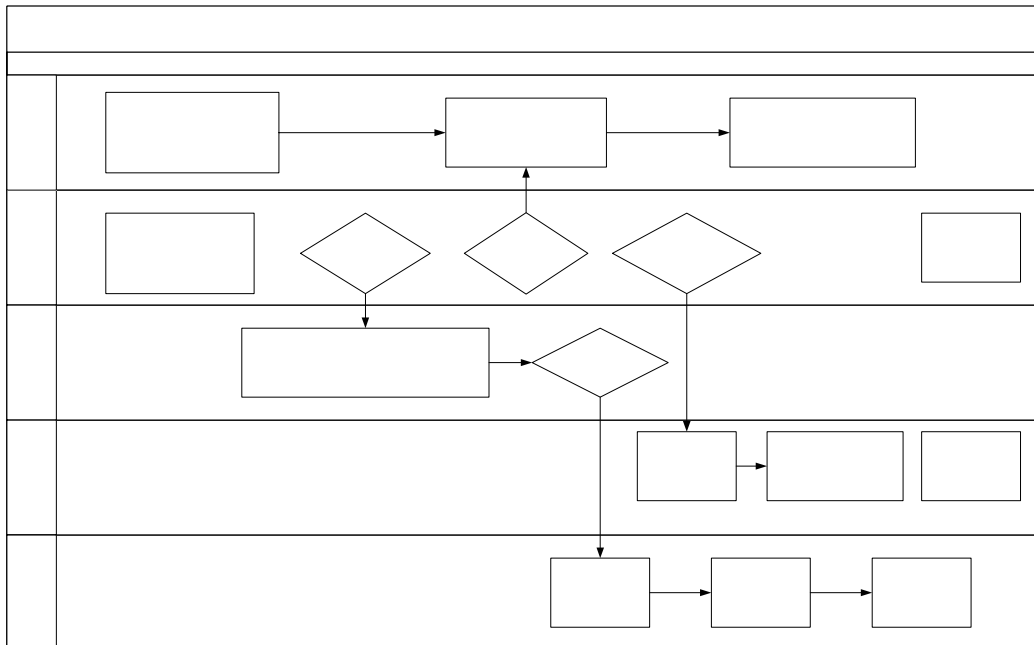


Figure 1: Phases of Configuration Management

During the initiation phase the directory structure or “work product organization” must be defined and documented along with the CCB, Configuration Manager, and known CIs. That documentation becomes the

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

CMP for the product and as such is a living document that must be updated in the “Monitor and Control Phase.” During this phase the CMP is referenced to determine when CIs are added, transitioned to formal control and baselined. The “Baselining Phase” is essentially a state at which a baseline is declared. Establishment of a baseline impacts the control required for items in that baseline as defined in the CMP. Changes to baselines are documented in ECRs, reviewed by the CCB, implemented, tested, reviewed, and assigned to a release or subsequent baseline. All baselines should be audited by the Configuration Manager. Baselines to be released to a customer (internal or external) should also be audited by Quality Assurance (QA) to independently verify that the product to be released is complete and can be rebuilt. Each of these topics will be discussed in more detail in this paper.

## CONFIGURATION ITEM IDENTIFICATION

The CCB must uniquely identify each CI for the product in accordance with established organization identification procedures. The CIs and product specific methods for unique identification are documented in the CMP. Guidelines for, and examples of, CIs which should be identified are shown in Table 1.

*Table 1: Configuration Item Example and Guidelines*

CI Categories	Examples/Guidelines
Plans	Project Plan(PP), Software Development Plan (SDP), Quality Assurance Plan (QAP), and Configuration Management Plan (CMP)
Specifications	System/Subsystem Specification (SSS), Software Requirements Specification (SRS), and Interface Requirements Specification (IRS)
Design documents	System/Subsystem Design Description (SSDD), Software Design Description (SDD), and Interface Design Description (IDD), and Database Design Description (DBDD)
Test documents	Acceptance Test Plan (ATP), Software Test Plan (STP), Software Test Description (STD), and Software Test Report (STR)
Test data	Input and output files.
Other miscellaneous documents	Coding Standards, Product Version Description (PVD), Software Product Specification (SPS), meeting minutes, and peer review data.
Code	The baseline contains all project code including source files, assembler and linker/compiler directive files, assembler libraries, utilities, and test programs/scripts.
Hardware data	All data files necessary for reproduction of the unit shall be put under configuration control. This includes schematics, mechanical drawings, printed wiring board Gerber files, and programmable device binary files.
Commercial tools	A text file listing the name and version of the tools in use by the product must be configuration controlled. This includes assembler, linker, compilers, utilities, and operating systems.
Other tools	The executable version of all tools used for a product along with any associated libraries and data files will be kept under configuration control. Otherwise, a text file listing the name and version of the tools in use by the product must be configuration controlled.

## GENERAL BENEFITS

In the short-term, a CMS provides several immediate benefits even before the first formal baseline is declared. First, in the initial stages of development it provides a safe way to share files, as CIs, among project team members. Second, it provides the informal history of changes, which allows developers to easily communicate modifications to anyone reviewing or modifying the item. Finally, revisions in the CMS may be compared and informally baselined at any time.

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

In the long-term a CMS provides:

- Control over changes to CIs – CM systems normally allow level of access or privileges to be defined for each class of user.
- Maintenance – long-term maintainability of the product and control of the maintenance to the product.
- History – knowledge of who made modifications and why. Some CM tools allow Engineering Change Requests (ECRs) to be linked to the CIs which were changed.
- Reproducibility – baselines allow the product to be reproduced.

---

## WORK PRODUCT ORGANIZATION

---

It is important to put work products into a CMS in an organized manner, using a documented structure. If developers simply place work products anywhere they choose, it becomes difficult for someone else to find them at a later time. Personnel whose duties span multiple products, such as senior managers and QA, must familiarize themselves with numerous different organizational structures if ones derived from a standardized set of templates are not used.

Normally, a small set of structure templates needs to be developed, documented, approved, and disseminated to the organization. At the beginning of the product lifecycle, the project team must select the template that is most appropriate for their product, define the product architecture, and develop their directory structure to comply with the architecture and selected template. This information must be documented in the CMP along with specific locations of work products. The decision of where to place work products should be made with the establishment of baselines in mind. The CMP is then approved by QA and any other impacted group.

The CM tool should support baselining at multiple levels in the directory structure. The CM tool will record the current revisions of all of the files and provide a means by which this same collection of files can be reproduced at a future time. Source files, documents, test data, etc. that are logically grouped together should be physically stored together within the directory structure. Mixing CIs together that are not intended to be baselined together is counterproductive, making the documentation and reproduction of baselines less efficient.

Figure 2 shows a template of a directory structure where all CIs for a single product are contained within a single folder (e.g., “Product A”). The CIs for each product are further subdivided in separate folders for software, plans, other documents, etc. There are also sub-system folders which would be used if the product can be logically divided into separate sub-systems. Notice that many of the same sub-directories that are present at the system level are repeated at the sub-system level. A

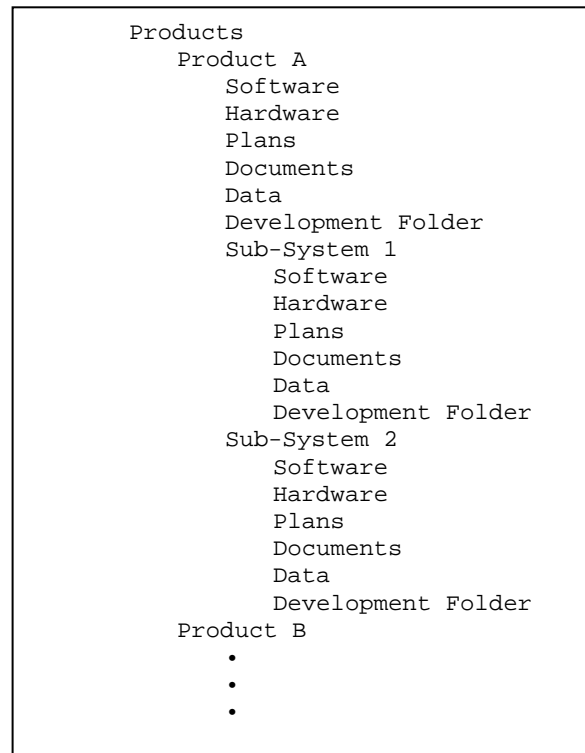


Figure 2: Directory Structure Template

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

CI that is only relevant to a sub-system should be stored in the appropriate sub-directory for that sub-system, whereas a work product that spans multiple sub-systems (or the entire product) should be stored in the appropriate sub-directory at the product level. This organizational structure facilitates finding work products. It also allows independent baselining at the sub-directory, sub-systems, and/or product level.

Directory structures are updated in an iterative manner as the phases of the product lifecycle are executed and the product becomes well defined. Updates to the initial directory structure should adhere to the principle of keeping associated work products together at the lowest possible level in the structure. Conscientious modification which support baselines goals will ensure that work products can be easily located and that baselines can be easily established.

---

## BASELINE TYPES

---

The CCB must define appropriate baselines for the product. There are four types of baselines: Functional, Allocated, Developmental, and Product. A brief description of each type of baseline follows.

FUNCTIONAL BASELINE is prepared once the product requirements and system specifications have been documented.

ALLOCATED BASELINE is prepared once the requirements specified in the Functional Baseline have been allocated to respective subsystems.

DEVELOPMENTAL BASELINE(S) is a working baseline that is normally under a combination of informal (author) and formal control. This type of baseline would not normally be released to the customer, but is maintained for internal purposes. As development progresses it is desirable to create a baseline at incremental stages in development as deemed appropriate by the CCB. The Developmental Baseline should contain design documentation, tested units, test data, code under development, etc. in addition to the Allocated Baseline.

PRODUCT BASELINE is prepared when requirements for the specified baseline have been implemented. The Product Baseline is an official release either for integration or to the customer. For large systems that have multiple components which must be integrated, software CIs should have passed unit tests, internal integration tests, and be prepared for system integration before this baseline is declared. This baseline will be under formal control. As integration progresses and revisions are made to CIs, the baseline will be incremented. Once acceptance testing has been passed another Product Baseline will be declared. The product baseline may contain user documentation, test reports, version description, etc. in addition to the Developmental Baseline.

The difference between the Developmental Baseline and Product Baseline is the degree of formal testing and readiness for release to the customer.

## LEVELS OF CONFIGURATION ITEM CONTROL

There are two levels of configuration control: formal and informal. These levels are described in the following subsections.

### INFORMAL CONTROL

Informal control is also referred to as author control. Informal control is used during the initial development of the work product. It requires version control and informal audits of the CMS. The informal audits ensure that appropriate work products are being checked into the CMS in appropriate locations.

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

Additional audits may be conducted to verify that the product may be built at different stages in development. These audits should be conducted in conjunction with developmental baselines.

## FORMAL CONTROL

Formal control requires version control, formal change control, baselining, and baseline audits. The CCB in conjunction with the CMP identifies when items will transition to formal control. Formally control items are modified only after authorization from the CCB, which follows written ECR procedures. All deliverables must be placed under formal control.

### CM BASELINE CONTROL PROCESS

The Baseline Control Phase includes procedures to ensure that changes to a baseline are reviewed, approved, controlled, and tested. Any product for which a baseline has been established must use this phase. Baseline control includes documenting requested changes, approving changes, scheduling approved changes for design and implementation, and verifying that changes have been properly implemented. The steps required to accomplish these activities are depicted in Figure 3 and described in the list that follows.

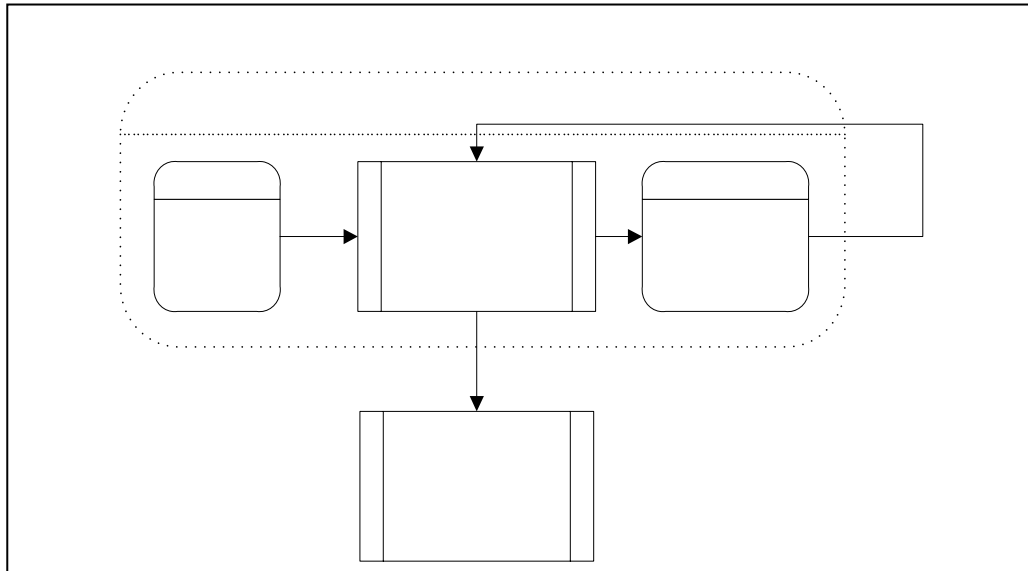


Figure 3 CM Baseline Control Process

1. A project team member (or other authorized person) enters an ECR into the CMS, according to the product's CMP, for the following reasons: 1) a request for new/modified requirement or enhancement is received from the customer or internal sources, or 2) a problem is identified either internally or externally.
2. The CCB convenes to review the status of the ECRs. The CCB is responsible for assessing change requests to determine impact to the product and approving and prioritizing changes. The CCB reviews the ECRs and determines which shall be acted upon for the next baseline.
3. For each approved ECR, the project team members perform their assigned development/modification.
  - The project team member that is assigned to implement the ECR:

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

- a. plans the work to be done including actual modification, documentation updates, and test plan for this specific change.
  - b. does the work.
  - c. performs regression testing in accordance with the ECR test plan to verify correct implementation and overall operation.
  - d. updates, as appropriate, any documents affected by the change (e.g. design documents, user manuals, etc.).
  - e. updates all necessary ECRs describing the change or changes made for the work product(s).
  - f. notifies the CCB of the completion of the ECR.
- If, during the course of planning and executing the work, it is discovered that other CIs may be affected by the implementation of the solution, the CCB is notified immediately.
  - If, during the course of planning and executing the work, it is discovered that estimated time/cost to complete is larger than originally estimated, the CCB is notified immediately. The project team member that is assigned to implement the ECR:

## CONFIGURATION ITEM STATES AND TRANSITIONS

A feature that is useful in a CMS is the ability to identify the state of an individual CI. There are four defined states that should be applied to items that are controlled under the CMS. These are:

### *State Definitions*

**Exp** (“Experimental”) – This is the initial state that should be assigned when a new CI is added. This state indicates that the CI is currently not suitable for the use for which it is ultimately intended. It has not been reviewed or approved, nor is it formally controlled.

**App** (“Approved”) – This state indicates that the CI has received the required level of review as defined by the appropriate CMP and is approved for integration or release. Once a CI has achieved this state it may only be modified under the authority of an ECR.

**Rel** (“Released”) – This state indicates that the CI has been released, as defined by the appropriate CMP. Once a file has achieved this state it may only be modified under the authority of an ECR. Typically, this state is applied to all CIs at the time a product is baseline is declared.

**Dev** (“Development”) – This state indicates that a CI is no longer suitable for integration or release, but is in a state of transition. The difference between this and the Exp state is that work is being done under the authority of an ECR. A CI that is in the Dev state may only be modified under the authority of an ECR.

Note that all work done in the App, Rel, or Dev state must be linked to or referenced to the ECR under which the work was authorized.

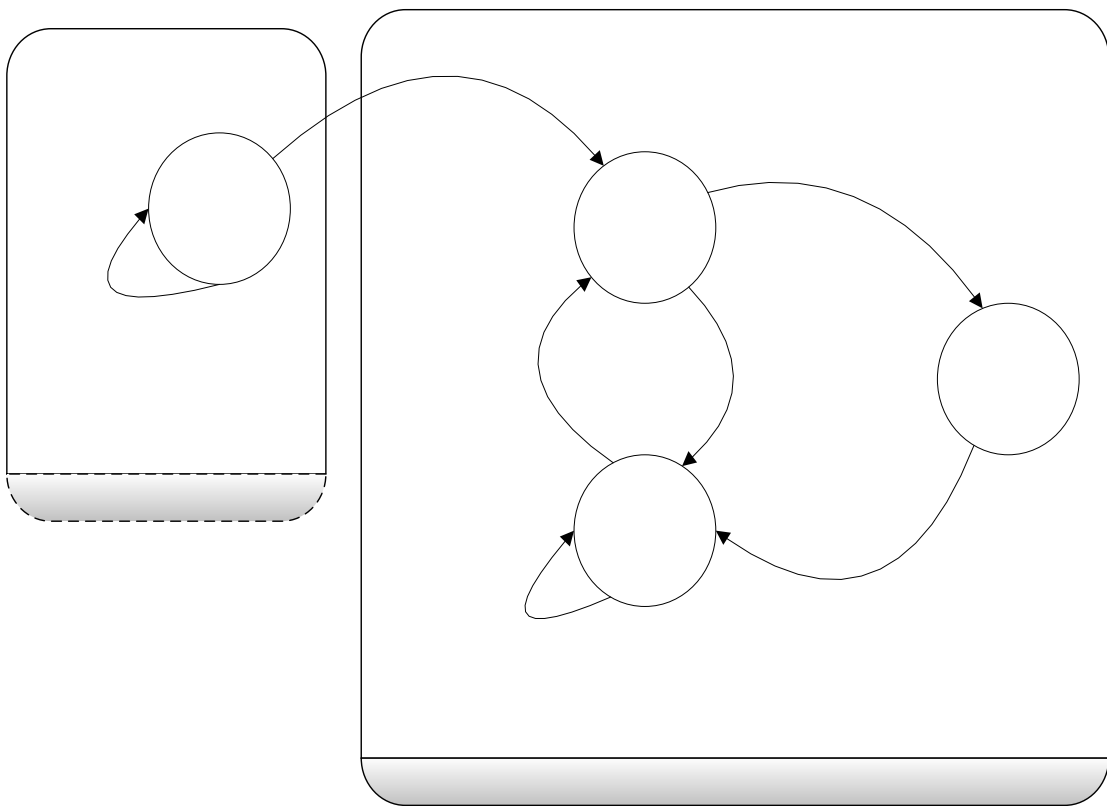
# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

## *State Transitions*

Figure 4 describes the normal sequence in which CIs transition from state to state. For the purposes of this discussion “check-in” means to enter an updated revision into the CMS, and a transition indicates one of the following possible changes:

- A modification to the CI (i.e. a new revision is checked-in).
- A change of state to an existing CI (i.e. no new check-in).
- A check-in of a new CI and a change of state at the time of check-in.



*Figure 4 State Transition Diagram*

**Exp** – A CI in the “Experimental” state may transition back into the “Experimental” state as many times as is necessary until it reaches a point where the author decides it is ready for review. Each transition corresponds to the check-in of a new revision. Once it is reviewed and approved it moves into the App state (which is done by changing the state without checking in a new revision).

**App** – A CI that is currently in the App state is validated as an integrated CI of some larger entity according to whatever procedure is defined in the CMP, and then the state is manually changed to the Rel state to indicate this. If changes need to be made to the CI after it is initially approved, but before the entire entity it is a part of is validated for release (e.g. a unit needs to be re-worked after failing an integration test), it transitions to the Dev state under the authority of an ECR.

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

**Rel** – A released CI returns to the Dev state when additional work is performed upon it under the authority of an ECR. This transition corresponds to the check-in of a new revision that includes a state change to “Dev” at the time it is checked-in.

**Dev** –A CI in the Dev state may transition back into the Dev state as many times as is necessary until it reaches a point where the author decides it is ready for review. Each transition corresponds to the check-in of a new revision. Once it is reviewed and approved it moves into the App state (which is done by changing the state without checking in a new revision).

---

## BASELINE DOCUMENTATION

---

Each type of baseline will have associated documentation which varies in formality based on the type of baseline being performed.

### BASELINE VERSION DESCRIPTION

The CMS supports the capture of baselines, but sometimes reproducing the baseline may not be as simple as one might expect. If the structure of the CMS has not been well established or segregation of sub-system work products has not been enforced, the Baseline Version Description (BVD) will be more complex to develop.

To supplement the documentation of a baseline that is provided by the CMS, a BVD is created. A BVD is a document that records the essential information needed to not only reproduce a baseline, but to understand why it was created. It answers the following questions:

- Who created this baseline?
- What type of baseline is it?
- Why was it created?
- When was it created?
- What is the level of control for the CIs?
- Which, if any, ECRs have been incorporated into it?
- When was its predecessor baseline created?
- How is this baseline reproduced?

The BVD does not contain the development environment, build instructions, or operating instructions. It should be as simple and streamlined as possible, but still answer the questions posed above. Figure 5 shows an example BVD. The details of how to document the components of the baseline will be tool-specific. There may be other organization-specific information that should be added as well, but the goal is to make the document as simple as possible and still do the job it was intended to do. This will encourage appropriate baselines to be declared and implemented.

Depending upon the tool used, and how judicious the project team was in establishing their directory structure, it can be a challenge to document how to reproduce a baseline. However, once a BVD has been created, it is usually a simple process to copy the previous one and make the appropriate adjustments to it so that it describes the next declared baseline. All of these BVDs can be stored together in a single folder,

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

incorporating the name of the baseline into the file name to facilitate identification. This makes it easy to browse through the documents to locate a previously documented baseline.

BASELINE VERSION DESCRIPTION					
<b>Project #:</b> A-1234	<b>Work Product Name:</b> Football simulator				
<b>Version :</b> 22143	<b>Originator:</b> George P. Burdell				
<b>Date:</b> November 27, 2001	<b>Date of Previous Baseline:</b> November 13, 2001				
<b>Baseline Type:</b> <input type="checkbox"/> Functional <input type="checkbox"/> Allocated <input checked="" type="checkbox"/> Developmental <input type="checkbox"/> Product					
<b>Control Type:</b> <input type="checkbox"/> Formal <input type="checkbox"/> Informal <input checked="" type="checkbox"/> Mixed					
<b>Reason for Baseline:</b>					
This is an unplanned baseline. Although the splash screen graphic has not been completed, the part where the bulldog whimpers as it runs under the bleachers to hide is done. If the customers decide they want a demo we would like to be able to easily reconstruct this version of the product.					
<b>Comments:</b>					
<b>Summary of Formal Changes from Previous Baseline:</b>					
ECR #	TITLE				
12	Football trajectory calculator shows all passes as going straight up in the air.				
13	Tackle calculator never shows a 300 lb. tackle able to bring down a 180 lb. running back.				
<b>Summary of Author Changes from Previous Baseline:</b>					
The files Quarterback.cpp, and Tight_end.cpp were added to the project T:\product\football_simulator\software\players.pj					
<b>Baseline Identification:</b>					
<b>Project</b>	<b>File</b>	<b>Label</b>	<b>Rev.</b>	<b>State</b>	<b>Formal Control</b>
T:\product\football_simulator\software\main.pj	All Files	b5D	1.2	Exp	
T:\product\football_simulator\software\graphics.pj	All Files	b5D	1.2	App	X
T:\product\football_simulator\software\players.pj	All Files	b5D	1.7	Exp	
T:\product\football_simulator\software\scores.pj	All Files	b5D	1.3	App	X

*Figure 5 Example Baseline Version Description*

One system for naming baselines is to use a label of the format “b”<baseline number><baseline type>, where the baseline number is a continuous number starting at 1, and the baseline type is “F” for functional, “A” for allocated, “D” for developmental, and “P” for product. An example of a series of baselines labeled using this system is as follows: “b1A,” “b2F,” “b3D,” “b4D,” “b5D,” “b6P.” In this example there are six declared baselines, including three developmental ones.

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

The baseline identification information contained in the BVD is used as input into the Product Version Description (PVD). The summary of formal changes from each BVD developed since the previous PVD is also used as input into the new PVD.

## PRODUCT VERSION DESCRIPTION

The Product Version Description is the document that provides complete information about a delivered product and how to construct it. The PVD describes the physical materials that constitute a released product (CDs, manuals, etc.), the contents of the release, such as executables and dynamically linked libraries (DLLs), instructions for how to build the released product from its source files, and many other items of crucial information. Documenting all of this information is critical if the product will ever be reproduced and its development continued. This information is especially important if project personnel leave the organization or development environments change over time. If a product is put back into development after a long hiatus, much of the institutional knowledge of the product may be lost. The PVD should provide the information that the developer needs to retrieve the documentation, source files, and development tools from the CMS that are needed to rebuild the specified product baseline.

Some products are relatively simple and easy to reproduce, but often times they are quite complex. They may contain multiple executables and DLLs, numerous source files, as well as installation scripts. The product may incorporate components that are shared with other independent products, or “borrow” components directly from other products. All of this information must be documented in order to have a complete understanding of what constitutes the product. The major sections of the PVD are shown in Table 2.

*Table 2: Product Version Description Contents*

Section	Content
1. Scope	Identifies the product and its purpose.
2. References	Any referenced documents should be fully identified here. Product information that is contained in other documents may be incorporated in the PVD via references.
3. Release Description	Contains information detailing what the product is and how to use it. It has the following subsections:
a. Physical Inventory of Materials Released	A description of the physical contents of the delivered product (e.g. identifies the CDs, manuals, or other components that together constitute the delivered product).
b. Inventory of Released Product Contents	A list of all of the items that make up the delivered products (e.g. executables, DLLs, documents). All of these items should be stored in the CMS, and their location and revision number should be specified. It does not include source files, unless these are included as a part of the release.
c. Installation Instructions	Instructions for installing the product.
d. Changes from Previous Release	A summary of the changes to the product since its previous release, if applicable.
e. Operating Instructions	How to use the product. Normally a reference to a separate user’s manual is placed here.
4. Build Description	Contains the details of the components used to create the product and how it is built. It has the following subsections:
a. Configuration Items Built	This is a list of all of the delivered items that were built from other CIs developed for this product. This section will normally be a subset of the Inventory of Released Product Contents section. For example, the

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

Section	Content
	Inventory of Released Product Contents section might list two executables, one of which was developed for another product but is included as a companion for the executable developed for this product. Only the executable that was created for this product would be listed in this section.
b. Applications/Tools Required to Build	All software applications and tools that are needed to fully reproduce the delivered product should be listed here. Note that all software development tools should be archived so that they are available for use if the product is put back into development after a long period of inactivity.
c. Build Procedures	The sequence of steps needed to build the product must be documented. It is not sufficient to simply control the source files and applications needed to build the product. Sometimes there are compiler settings or important steps in a build process that are not obvious to someone who is not familiar with the product's development environment (or might be forgotten by the original developer after an extended period of time). The detail to which the build procedure needs to be documented is subject to differences of opinion, but more detail is always better than less.
d. Inventory of Configuration Items Needed to Build Released Product Contents	A complete list of the CIs (locations and revision numbers within the configuration management tool) needed to build the items listed in Configuration Items Built should be listed here. Depending on the tools used, this could consist simply of one or more folder-based baselines, or could be much more complex. A reference to a BVD document can be used here (normally the last developmental baseline that was successfully submitted for qualification test).
5. Installation Build Description	This section should contain whatever information is needed for someone to recreate the installation disk for the product, if it uses one.
6. Known Problems/Upgrades	If the product has been through its qualification test and some problems were noted, a decision may be made to release the product with the known problems, and to correct them in a future release. It is also possible that the product may have some recognized limitations for which plans have already been made to correct them. This type of information is recorded here along with open ECRs.
7. Notes	This section contains any additional information that is of value to the builder or user of the software that is not covered in any of the previous sections.

## VERIFYING BASELINES

When developing software it is extremely important to be able to verify that all of the components needed to build the final product are identified and controlled. This is the purpose of baselining. The ability to rebuild the delivered product from its components is critical when it is a product that will be maintained and extended in the future. Performing a CM audit of the files that constitute the product against the product's documentation is important, but verifying a baseline is more than that.

Consider this scenario for a product that is brought back into production six months after its original developer has left the company and a new engineer is assigned to rebuild the last delivered baseline. All of the files that are documented in the baseline description are retrieved from the CMS and compiled, but there are files missing. No executable can be built. The problem is that the files in the baseline are not a

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

comprehensive list of the files needed to build the product. When the original developer “verified” his baseline using a new, clean, folder on his computer, the compiler was pulling files from a long forgotten folder that was created early in the project. That machine has since been reformatted and the files are now gone.

Building the software from scratch on a “clean machine” is the best way to verify that the software can be recreated at will. A “clean machine” in a perfect world is a machine that contains only the software components (operating system, compiler, revision control software, etc.) that are required to build the software product. A build on a clean machine would catch the problem in the scenario given above.

There are three methods for achieving a clean machine:

- 1) Wiping the machine and installing all the software needed for a build as described in the PVD.
- 2) Creating “ghost” images of standard development and operating environments. This is done by creating a clean system that has all of the software that constitutes each defined development and operating environment. Each of these environments is ghosted on to one or more CDs.
- 3) Using a commercial product which allows the creation of a “virtual” machine which may be configured and saved. These “virtual” machines may be re-instantiated as required to provide the clean environment needed for verifying a baseline.

Method number 2 above is described in more detail in the following paragraphs.

Once ghost images have been created on CDs, the machine can be restored to its “clean” state in minutes. If there is a need in the organization for multiple clean machines, rather than preparing ghost images for each machine separately, it is possible to use the same set of CDs created from one machine to ghost all of the others. To be certain the CDs will install correctly on all machines, they must be absolutely identical in their hardware configurations, down to the motherboards. The machines must be purchased together to assure this level of compatibility, as manufacturers will sometimes substitute internal parts during the production of machines that might otherwise appear to have the same specifications. Of course, adequate software licenses must be obtained for each machine that will be ghosted. If a software license for rarely used, expensive development tools is an issue, a machine can be ghosted with the basic environment, and then the additional software installed after ghosting. This is still a faster method of creating a clean machine than installing all of the software from scratch.

Sometimes a standardized, ghosted image that is a perfect match for the specified development environment for a product may not exist. There may only be available an image that is a superset of the software needed to verify the baseline of a particular product. The auditor must decide if the irrelevant software that is on the ghosted image might actually contribute to the successful build of the software (i.e. if it were not present, the clean build would fail). A decision may be made to prepare a clean machine from scratch, rather than from a ghosted image. But if the decision is made to use a ghosted image that contains software applications and/or tools beyond what is specified by the developers as needed for the build, these should be documented in the audit report. If a future attempt to reproduce the product fails, the audit report can be examined to determine what, if any, additional software was available during the previous audit of the product baseline.

---

## CMS SUPPORT FOR AUDITS AND PEER REVIEWS

---

A CMS can enhance the process for auditing documents and help in the control of a work product that is being peer reviewed. When a document needs to be audited or peer reviewed, it is imperative that the auditor

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

and/or reviewers know what version of the work product is to be examined. The use of a CMS allows the author to specify the location as well as the specific CMS revision of the subject document.

It is important to refer to the location of a file in the CMS rather than attach it to an email; this reduces the possibility of examining an incorrect version of the work product. When a printed copy of the document is required, it is recommended that the CMS revision number be written on the printed copy. The CMS may provide this service, but if it does not, manually record the CMS revision number on the printed document. When the revision number of a work product to be reviewed is known it is easy to tell if the work product has been updated (work product has a later revision number) since notification of audit or review. Always access a work product with the CMS as opposed to going behind the scenes and accessing the file directly (e.g., by using Windows Explorer). Obtaining a file via the CMS will ensure that the correct version is being accessed and will indicate if another user has locked the work product and may be updating the file.

When an audit or peer review has been completed and modification or changes to the work product are identified, the CMS allows for ease of verification of changes because the new revision and location can be specified. The examiner of the file can access the new revision of the work product and can see the work product history of all the revisions between the one that was originally examined and the new revision of the work product. The CMS should also allow for differencing of files to quickly determine what changes have been made.

Peer reviews can be facilitated through integration with the organization's CMS. The first step is to control peer review data within the CMS if this is not already being done. This brings all of the benefits of revision history and sharing of data that a version control system provides for other work products to the peer review process. If peer reviews are currently being conducted using printed forms, these can easily be converted into electronic forms (e.g. using a word processor) so that they can be archived using the configuration management tool. Once all peer review data is being managed within the CMS it can be utilized more effectively, particularly during peer review meetings.

For a relatively small investment, a conference room can be equipped with a networked PC that communicates with the CMS. This allows access to the peer review forms and reviewed work products during the meeting. The room can also be equipped with a projector that displays the screen of the networked PC to all of the meeting participants. This provides several benefits. First, the work product being peer reviewed can be displayed to all participants in the meeting, which makes it easy to refer to specific sections of the work product. Second, work products that were not included in the review packages, but are determined to be needed during the review, may be accessed. Third, it permits the findings (i.e. defects) to be recorded in real time, and to be agreed upon as being accurate by all of the participants before the conclusion of the meeting.

---

## CONTROLLING REQUIREMENTS WITH THE CMS

---

Requirements Management is a crucial element of the product management process. This activity begins at product initiation and continues through the life of the product. Its purpose is to establish and maintain a common understanding between the customer and the project team of the product requirements. The clarity and completeness of the initial requirements determine the type of product to be developed; as such, they are critical to the planning of the product's development. Thus, it is critical that requirements be reviewed and updated in a controlled manner.

Although there are many tools on the market for the specific purpose of requirements management these tools may not be cost effective for all organizations. Managing requirements through a CMS is feasible, and a method for doing this is described in the following paragraphs.

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

The CMS can be used by the project team to control requirements as they are being defined and reviewed. Requirements must be identified with a unique identifier. This identifier should be composed of a number and system or subsystem identifier to facilitate traceability throughout all phases of the lifecycle (i.e., requirements, design, implementation, integration, and test). As the requirements are being defined, the CMS allows controlled update of the requirements file(s). The CMS also provides a history of the updates by recording when a file is checked in, who checked it in, and a comment by the person checking the file back into the CMS. Therefore, the comment should be a well-defined statement of the modification including, where appropriate, the identifier for each requirement that was directly impacted by the change. The comment may be used in the future to determine which revision should be examined for a specific addition or change.

As previously described for audits and peer reviews, the CMS allows the project director to identify the revisions of all documents to be peer reviewed or audited and provides support for the control of changes to the documentation. This allows the project team to review the requirements, and assists in buy-in from the project team because they have the opportunity to review what is being promised to the customer and are allowed to provide input into the definition of the requirements. When the project director, the project team, and the customer approve the requirements, the Configuration Manager should use the CMS to declare a Functional Baseline or Allocated Baseline and place the requirements document(s) under formal control.

A critical element of requirements management is control of the requirements to ensure a stable target for the product development. To provide this control it is necessary to identify and declare baselines. Requirements must be baselined at critical points in the product development. At a minimum, the requirements will be baselined every time they are delivered to the customer. This ensures that the exact document delivered to the customer may be reproduced at a future date. Additional baselines may be desired for internal purposes. Baselines and level of control must be documented in the product CMP.

When the requirements are under formal control and a modification to requirements has been identified, it will be necessary to write an ECR. The proposed requirements change will undergo the ECR process to define the change, the impact on the product development schedule and cost, and the effect on the entire system (e.g., design, implementation, and test). The relevant person makes a determination of the potential impact of the requested change to the schedule and budget. If there is a significant impact, re-estimation should be conducted. When the ECR is completely defined, it will be evaluated by the CCB for acceptance. If the ECR is accepted, it will be assigned to a project team member to implement. When the implementation is complete, the modifications will be reviewed to make sure that the modifications have been addressed and appropriately propagated through all the product development phases before the ECR can be closed. When the ECR has been completed, the CCB will determine if a new baseline is necessary.

The project team should review requirements activities on a regular basis, at least monthly, to assess if requirements are still well defined and if any design or implementation issues have arisen that may require a modification to requirements. This review includes review of traceability to existing design and test documents. In other words, work products that are affected by a change in requirements, design, or implementation must all be updated to reflect the impact of the change.

---

## CONCLUSION

---

Simply controlling files using a CM tool is not configuration management; it is only a part of the CM process. Baselines must be clearly documented and the development and operational environments must be defined and reproducible. The accuracy of a product baseline must be verified by repeating the build process as documented in the PVD on a clean machine. Changes to the product must be documented and managed through a change control process that is well defined and followed by the project team to ensure that the

# MAKING CONFIGURATION MANAGEMENT WORK FOR YOU

AUTHORS: JEANNE BALSAM, MARK PELLEGRINI, AND JEAN SWANK

changes to requirements are propagated and traceable through all the affected work products. The process for auditing and reviewing CIs is more efficient when data in the CMS is conveniently available, as described previously. Intelligently integrating the CMS into the product lifecycle and the software development process will maximize its effectiveness.

A project team that utilizes CM techniques presented in this paper is on a path for successful requirements definition, design, implementation, integration, and testing of their product. Transitioning requirements, design, implementation, and test components to formal control at the earliest possible point in the lifecycle and maintaining appropriate traceability matrices will increase the probability of delivering products on time and within budget that also meet customer expectations.