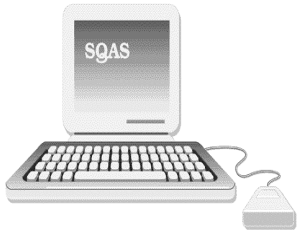
# Department of Energy Quality Managers
# Software Quality Assurance Subcommittee
### Reference Document SQAS20.01.00 - 2000




# Software Configuration Management (SCM)
# A Practical Guide

## April 25, 2000

*United States Department of Energy*

*Albuquerque Operations Office*

*Abstract*

*This document provides a practical guide for integrating software configuration management disciplines into the management of software engineering projects. Software configuration management is the process of identifying and defining the software configuration items in a system, controlling the release and change of these items throughout the system lifecycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items.*

# Acknowledgements

This document was prepared for the Department of Energy (DOE) by a Working Group of the DOE Quality Managers' Software Quality Assurance Subcommittee (SQAS). At the time this document was prepared, the Working Group had the following members:

Y. Faye Brown, Y-12
Kathleen Canal, DOE-HQ
Al Cowen, PX
Ray Cullen, SR
Gary Echert, DOE-AL
Michael Elliott, UK AWE
Jerry Faulkner, KCP
John Hare, UK AWE
Philip Huffman, PX
Ed Kansa, LLNL
Cathy Kuhn, KCP
Michael Lackner, KCP
Carolyn Owens, LLNL
Maysa Peterson, LANL
Gerald Reisz, LANL
Larry Rodin, PX
Ed Russell, LLNL
Don Schilling, KCP
Ellis Sykes, DOE-KC
Nancy Smith, SR
Patricia Tempel, SNL
Anton Tran, DOE-AL
Bill Warren, LLNL

# Table of Contents

# 1. What Is Software Configuration Management?

Configuration Management (CM) is the process of identifying and defining the configuration items in a system, controlling the release and change of these items throughout the system lifecycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items. Configuration Management is practiced in one form or another as part of any software engineering project where several individuals or organizations have to coordinate their activities. While the basic disciplines of Configuration Management are common to both hardware and software engineering projects, there are some differences in emphasis due to the nature of software products. [3]

Software Configuration Management (SCM) is a system for managing the evolution of software products, both during the initial stages of development and during all stages of maintenance. A software product encompasses the complete set of computer programs, procedures, and associated documentation and data designated for delivery to a user. All supporting software used in development, even though not part of the software product, should also be controlled by SCM.

The SCM system is the collection of activities performed during a software engineering project to:

- determine and identify those entities of the software product that need to be controlled;
- ensure those entities have necessary and accurate definitions and documentation;
- ensure changes are made to the entities in a controlled manner;
- ensure that the correct version of the entities/software product are being used; and
- ascertain, at any point in time, the status of an entity (e.g., whether a specific entity is completed, being changed, waiting to be tested, or released to the customer).

SCM is performed within the context of several basic configuration management disciplines, including:

- Configuration Identification
- Configuration Control (change control)
- Configuration Status Accounting
- Audits and Reviews
- Release Processing

Explanations of the basic configuration management disciplines are provided in chapter 2.

## 1.1 Why Is Software Configuration Management Important?

The primary reason for implementing an SCM system is to keep the changing and iterative entities/software product(s) in a non-degrading state throughout the software lifecycle. This is a challenge that must be met in order to develop and maintain quality software products. The quality of the software products is fundamental to the level of quality of the complete system.

SCM provides a common point of integration for all planning, oversight, and implementation activities for a software project or product line. It provides the framework (labeling and identification) for interfacing different activities and defining the mechanisms (change controls) necessary for coordinating parallel activities of different groups. SCM also provides a framework for controlling computer program interfaces with their underlying support hardware, and coordinating software changes when both hardware and software may be evolving during development or maintenance activities. SCM provides management with the visibility (through status accounting and audits) of the evolving software products that make technical and managerial activities more effective. [3]

## 1.2 What Are the Benefits of Software Configuration Management?

SCM provides significant benefits to all projects regardless of size, scope, and complexity. Some of the most common benefits experienced by project teams applying the SCM disciplines described in this guide are possible because the SCM system:

- Provides a snapshot of dynamically changing software to help:
  - Make decisions based on an instantaneous view
  - Determine status at module and system levels
  - Make better decisions about the future of a software project.

- Tracks concurrent development of modules or components of the overall system to:
  - Prevent different developers from making changes to the same module at the same time
  - Allow for the overall system progress to be faster
  - Provide visibility of the entire software project to all developers.

- Organizes all concurrently developing code and associated documentation to:
  - Save overall project time
  - Focus each phase of the software product development to be organized and executed in a documented, prescribed manner.

The effective use of an SCM system also:

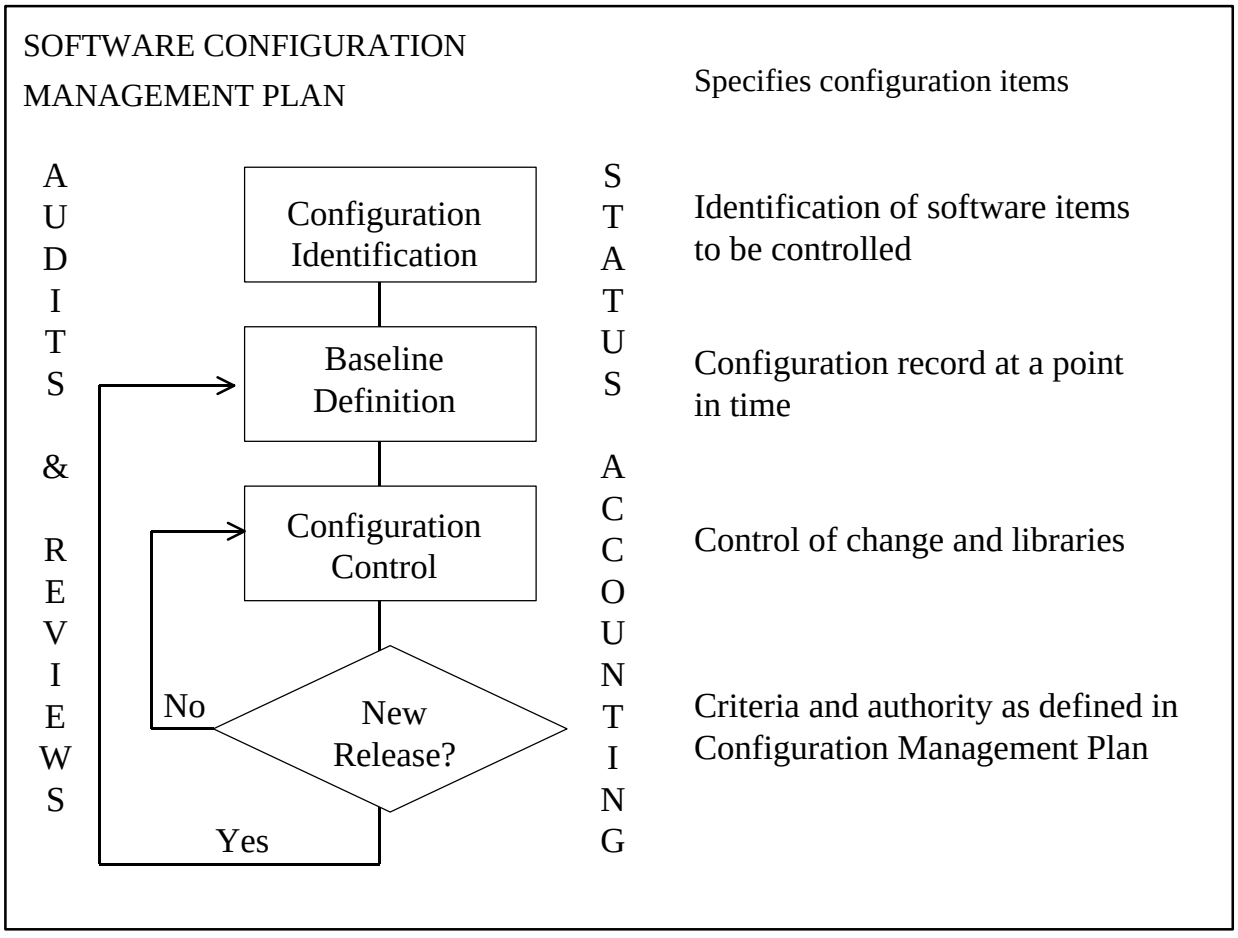
- Permits the orderly development of software configuration items.
- Ensures the orderly release and implementation of new or revised software products.
- Ensures that only approved changes to both new and existing software products are implemented.
- Ensures that the software changes that are implemented are in accordance with approved specifications.
- Ensures that the documentation accurately reflects updates.
- Evaluates and communicates the impact of changes.
- Prevents unauthorized changes from being made.

## 2. Software Configuration Management Disciplines

SCM disciplines are used to simultaneously coordinate configurations of many different representations of the software product (e.g., source code, relocatable code, executable code, and documentation). In practice, the ways in which SCM disciplines are performed will be different for the various kinds of software being developed (e.g., commercial, embedded, original equipment manufacturer). The degree of formal documentation required within and across different lifecycle management phases (e.g., research, product development, operations, and maintenance) will also vary. The use of interactive software development environments extends the concepts of SCM to managing evolutionary changes that occur during the routine generation of specifications, designs, and implementation of code, as well as to the more rigidly documented and controlled baselines defined during development and system maintenance. The effectiveness of the SCM system increases in proportion to the degree that its disciplines are an explicit part of the normal day-to-day activities of those involved in the software development and maintenance efforts. [3]

Figure 2-1 presents a diagram showing how the SCM disciplines interact at the highest level. Sections 2.1 through 2.5 provide additional information on each of the SCM disciplines.

**Figure 2-1. High-Level View of Software Configuration Management**



The Software Configuration Management Plan (SCM Plan) provides information on the requirements and procedures necessary for the configuration management activities of the software engineering project. It specifies who will be responsible for accomplishing the planned activities, what activities will be performed, when the activities will be performed in coordination with the project schedule, and what tools and human resources will be required to perform the activities. Consideration should also be given to the overall system configuration.

The first step in the SCM system is to identify the Configuration Items to be controlled and, as the project progresses, to identify the structure of the intermediate or complete software products. Elements to be controlled will normally include the software code and associated documentation. Documents could include requirements specifications, designs, user guides, test suites, test results, and user lists.

Baselines are defined and created in accordance with the SCM Plan and the phase of the project's lifecycle. A baseline is a specification or product that has been formally reviewed and agreed upon, that serves as the basis for further development, and can only be changed through formal change control procedures. A baseline is like a snapshot of the aggregate of software components as they exist at a given point in time. During each phase of the software lifecycle, configuration items that are completed and accepted by the approval authority become the baseline for that phase.

Any addition, alteration, or deletion to the approved baseline is deemed a change, and is subject to change control. Baselines, plus the approved changes from those baselines, constitute the current configuration identification. Each of the baselines established during a software lifecycle controls subsequent software development.[5] The following are typical configuration baselines that are established during the life of a software project. [3]

- Functional Baseline - Established by the acceptance or approval of the software or system specification. This baseline typically corresponds to the completion of the software requirement review.

- Allocated Baseline – Established by approval of the software requirements specification. This baseline typically corresponds to the completion of the software specification review.

- Developmental Baseline – Established by the approval of the technical documentation that defines the functional and detailed design (including documentation of interfaces and databases for the computer software). Normally, this baseline corresponds to the time frame spanning the preliminary design review and the critical design review.

- Product Baseline – Established by approval of the product specification following completion of the last formal functional configuration audit (FCA).

Once configuration items have been identified and baselined, Configuration Control provides a system to initiate, review, approve, and implement proposed changes. Each engineering change or problem report that is initiated against a formally identified configuration item is evaluated to determine its necessity and impact. Approved changes are implemented, testing is performed to

verify that the change was correctly implemented and that no unexpected side effects have occurred as a result of the change, and the affected documentation is updated and reviewed.

The discipline of Status Accounting collects information about all configuration management activities. The status account should be capable of providing data on the status and history of any configuration item. The information maintained in Status Accounting should enable the rebuild of any previous baseline.

Audits and reviews are conducted to verify that the software product matches the configuration item descriptions in the specifications and documents, and that the package being reviewed is complete. Any anomalies found during audits should be corrected and the root cause of the problem identified and corrected to ensure that the problem does not resurface. Generally, there should be a Physical Configuration Audit (PCA) and the Functional Configuration Audit (FCA) of configuration items prior to the release of a software product baseline or an updated version of a product baseline. The PCA is performed to determine if all items identified as being part of the configuration are present in the product baseline. The audit must also establish that the correct version and revision of each part are included in the product baseline and that they correspond to information contained in the baseline's configuration status report. The FCA is performed on the each software configuration item to determine that it satisfies the functions defined in the specifications or contracts for which it was developed. [3]

Appendix C provides a checklist of some of the activities commonly associated with the basic SCM disciplines. This list can be modified to include additional activities that are appropriate to the size and complexity of a specific software engineering project.

## 2.1 Configuration Identification

The first step in the SCM system is the identification of the items to be managed and the specification of the management authority responsible for each item. This is a critical step in the system since the identification scheme is employed throughout the lifecycle of the software product. The levels of authority assigned responsibility for each configuration item will vary depending on the complexity of the software product, the size of the development team, and the management style of the responsible organization.

Configuration identification consists of selecting the configuration items and recording their functional and physical characteristics as set forth in technical documentation such as specifications, drawings, and associated lists. The following are examples of items managed in the SCM system.

- Management plans
- Specifications (requirements, design)
- User documentation
- Test plans, test design, case, and procedure specifications
- Test data and test generation procedures
- Support software used in development, even though not part of the delivered software product

- Data dictionaries and various cross-references
- Source code (on machine-readable media)
- Executable code (the run-time system)
- Libraries
- Databases (data that are processed and data that are part of a program)
- Maintenance documentation (e.g., listings, detail design descriptions)

Effective management of the development of a software product requires careful definition of the configuration items. Changes to these items also need to be defined since these changes, along with the project baselines, specify the evolution of the software product. SCM includes all of the entities of the software product as well as their various representations in documentation. The entities are not all subject to the same SCM disciplines at the same time.

Support software is the class of software that may or may not be delivered with the software product, but is necessary for designing, enhancing, or testing the changes made during the life of the product. Support software may be user-furnished, developed in-house, leased from a vendor, or purchased off-the-shelf. In SCM, the focus is on describing the necessary controls used to manage support software. Developers and maintainers need to ensure that the support software is available for use for as long as the software product is in use. Whenever a production baseline is established, it is very important to archive all environmental and support tools along with the production code.

Software tools used in development, especially compilers and middleware, should be placed under configuration control so their identities and versions are included in the baseline data about each release. [8]

Consideration also needs to be given to the hierarchy of entities managed during the SCM system. There are several different ways of structuring this hierarchy. One way is a three-level hierarchy consisting of configuration items, components, and units. Another way of structuring the entities is in terms of the interrelationships between the software being developed and the other software entities used during development and testing such as support software, test software, and the operating system. A third method for structuring entities is in terms of the intermediate products generated in the process of building the software product, such as: modifiable entities (e.g., source code, document, test data); compilation entities (e.g., compilers, support software); and configuration items (e.g., different representations created in the process of producing deliverables).

An important aspect of configuration identification is the use of a formal naming convention for each entity. This naming convention, which typically uses a combination of mnemonic labels and version numbers, should be applied to all components of a configuration item. In establishing the naming convention, consideration should be given to system constraints such as name length or composition. Consistency in the application of this identification scheme is critical to accurate tracking of the software engineering process.

## 2.2 Configuration Control (Change Control)

Configuration control is a critically important feature of configuration management. The primary function of configuration control is to provide the administrative mechanism for initiating, preparing, evaluating, and approving or disapproving all change proposals throughout the software lifecycle.[5] Configuration control ensures that changes to configuration items are controlled and that consistency between component parts of a system is maintained. It reduces the possibility of making changes that may later adversely affect functionality. Figure 2-2 provides a diagram of the configuration control process.

The configuration control process involves three elements: [5]

- Procedures for controlling changes to a software product.
- Levels of authority (e.g., Configuration Control Board) for formally evaluating and approving or disapproving proposed changes to the software.
- Documentation, such as administrative forms and supporting technical and administrative material, for formally initiating and defining a proposed change to a software system.

Many automated tools support the configuration control process. The major ones aid in controlling software change once the coding stage has been reached. Automation of other functions, such as library access control, software and document version maintenance, change recording, and document reconstruction, greatly enhance both the control and maintenance processes. [5]

### 2.2.1 Procedures for Controlling Changes

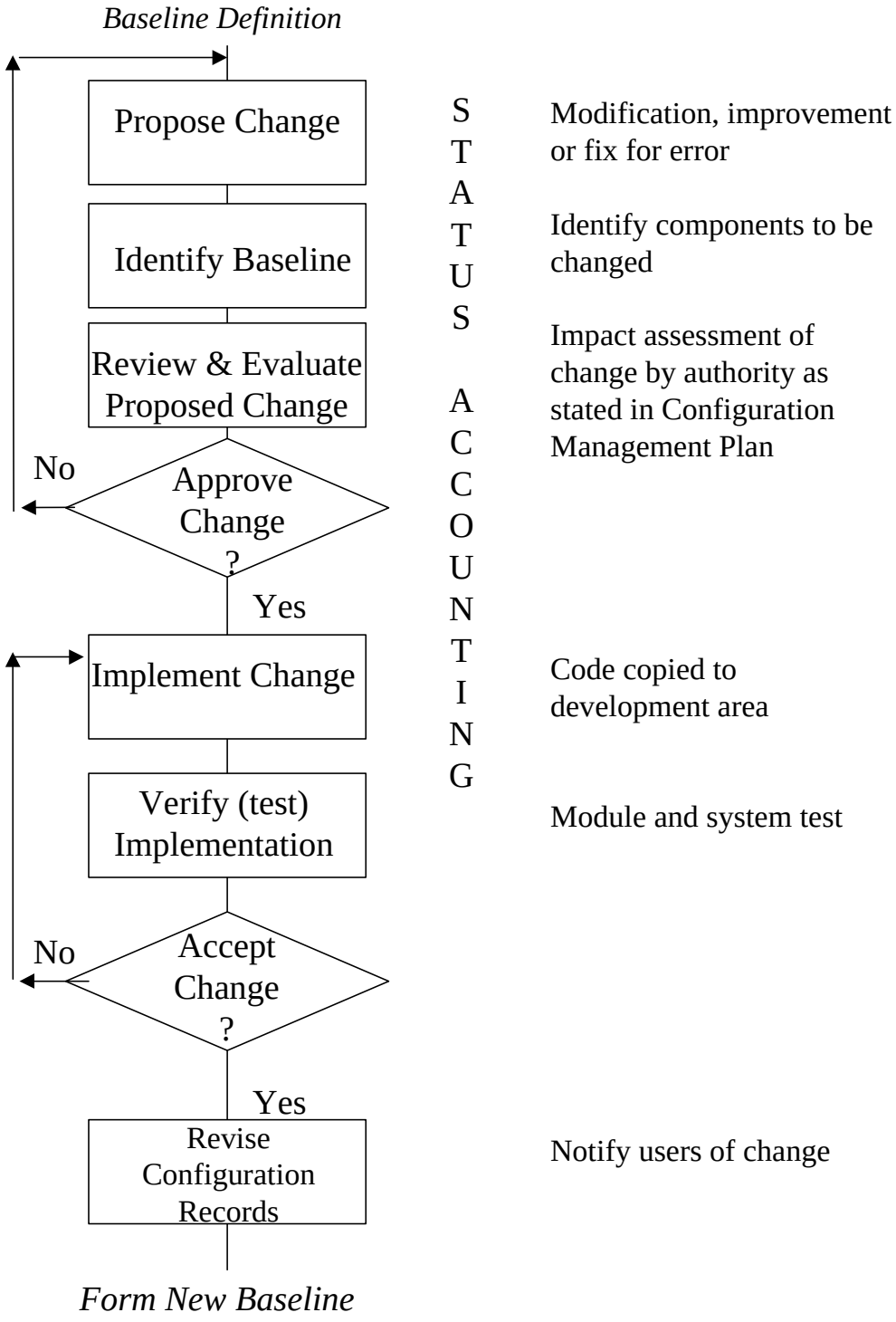
Changes occur at all phases in the software lifecycle. Design or implementation changes may be necessary if requirements change, or when deficiencies are identified in the design or implementation approach to a stable requirement. Testing may uncover defects that require changes in the code or the design and requirements. Changes must be made to the right version of the code, testing must verify performance of the change and the integrity of the remaining software, and all associated documentation must be updated to be consistent with the final code.

A mechanism is needed to process change requests (e.g., discrepancies, failure reports, requests for new features) from a variety of sources throughout the development process and during operation and support of the software product. This mechanism should extend to a definition of a process to track, evaluate, and implement change requests into a new version and new release of the software. Generally, no single procedure can meet the needs of all levels of change management and approval levels.

The minimum activities needed for processing changes include:

- Defining the information needed for approving the requested change.
- Identifying the review process and routing of information.
- Describing the control of the libraries used to process the change.
- Developing a procedure for implementing each change in the code, the documentation, and the released software product.

**Figure 2-2. Configuration Control Process**



## Software Libraries (Repositories)

The techniques and methods used for implementing control and status reporting in SCM generally center around the operation of software libraries. Software libraries are a controlled collection of software and related documentation designed to aid in software development, use, and maintenance. Software libraries provide the means for identifying and labeling baselined entities, and for capturing and tracking the status of changes to those entities. [3]

Libraries have been historically composed of documentation on hard copy and software on machine-readable media, but the trend is moving towards all information being created and maintained on machine-readable media. This trend, which encourages the increased use of automated tools, leads to higher productivity. The trend also means that the libraries are a part of the software engineering working environment. The SCM functions associated with the libraries have to become part of the software engineering environment, making the process of configuration management more transparent to the software developers and maintainers.

The number and kind of libraries will vary from project to project or organization to organization according to variations in the access rights and needs of their users, which are directly related to levels of control. The items maintained in the libraries may vary in physical form based on the level of technology of the software tools. When the management of the libraries is automated, the libraries that represent different levels of control may be functionally (logically) different even though they are physically the same. The insertion of entities and changes to entities in a controlled library should produce an auditable authorization trail.

The names of libraries may vary, but fundamentally there are three kinds: dynamic, controlled, and static. The dynamic library, sometimes called the programmer's library, is a library used for holding newly created or modified software entities (units/modules or data files and associated documentation). This is the library used by programmers in developing code. It is freely accessible to the programmer responsible for that unit at any time. It is the programmers' workspace and is usually controlled by the programmers.

The controlled library, sometimes called the master library, is a library used for managing the current baseline(s) and for controlling changes made to them. This is the library where the components and units of a configuration item that have been promoted for integration are maintained. Copies can be freely made for use by programmers and others. Changes to components or units in this library must be authorized by the responsible authority (which could be a configuration control board or other body with delegated authority).

The static library, sometimes called the software repository, is a library used to archive various baselines released for general use. This is the library where the master copies plus authorized copies of software configuration items that have been released for operational use are maintained. Copies of these masters may be made available to requesting organizations.

## 2.2.2 Levels of Authority

The levels of authority required for approving changes to configuration items under SCM control can vary. The level of control needed to authorize changes depends on the level of the item in relation to the product as a whole. The system or contractual requirements may dictate the level of authority needed. For example, internally controlled software tools typically require less change controls than critical software. Levels of authority can vary throughout the software lifecycle. For example, changes to code in the development cycle usually require a lower level of control to be authorized than changes to the same code after it has been released for general use. The level of authority can also depend on how broadly the change impacts the system. A change affecting specifications during the requirements analysis phase has less impact than a change affecting software in operational use. Likewise, changes to draft versions of documents are less controlled than changes to final versions.

### Configuration Control Boards

Configuration Control Boards (CCBs) enable the implementation of change controls at optimum levels of authority and scope. CCBs can exist in a hierarchical fashion (for example, at the program, system design, and program product level), or one board may have authority over all levels of the change process. In most organizations, the CCB is composed of senior level managers. They include representatives from the major software, hardware, test, engineering, and support organizations as defined in the SCM Plan. The purpose of the CCB is to control major issues such as schedule, function, and the configuration of the system as a whole. The CCB functions may be included in formal computer-aided software engineering (CASE) tools. [3]

Some of the functions of the CCB include:

- Directing the entire configuration management effort.
- Resolving all problems and situations that arise during the effort.
- Using the SCM system as its primary decision-making resource.
- Taking into account organizational management considerations for decision making.
- Orchestrating all activities from the beginning to the approval of the baselines for SCM establishment.
- Determining which products should be baselined or managed, the method to be used, and the order in which they should be done.
- Resolving all issues such as the most suitable product name to be used from among numerous documented aliases. Multiple aliases often occur when an SCM system was not originally in place.

### Software Configuration Control Boards

The more technical issues that do not relate to issues of performance, cost, and schedule are often assigned to a Software Configuration Control Board (SCCB). The SCCB discusses issues related to specific schedules for partial functions, interim delivery dates, common data structures, design changes, and the like. This is the place for decision-making concerning the items that must be coordinated across configuration items, but they do not require the attention of high-level management. The SCCB members should be technically well versed in the details

of their area, while the CCB members are more concerned with broad management issues facing the project as a whole and with customer issues. [3]

Depending on the size and nature of the software project, the SCCB may consist of a single librarian, multiple librarians, or include many SCCBs, each with a different functional responsibility for ensuring that changes are implemented and tested according to standard procedures and that hardware/software interfaces and interfaces between software modules are not violated. The SCCB also focuses on overall project management responsibility for ensuring that design or requirements specifications are not violated and that software changes are implemented according to cost and schedule constraints. The size and structure of SCCBs normally change over the lifecycle of the software.

To simplify the task of the SCCB, formal procedures for an initial analysis of each change should be established to aid the board in prioritizing the change. Review and testing requirements for prospective changes should also be established. Results from reviewing and testing the change should then be submitted to the SCCB to assist in evaluating the change.

### 2.2.3 Documentation

The Software Change Request (SCR) is one of the major tools used for identifying and coordinating changes to software and documentation. The SCR is used to capture important information about the change and to track the status of a change from its proposal to its eventual disposition. A typical SCR form contains a narrative description of the change or problem, information to identify the source of the report, and some basic information to aid in evaluating the report. An SCR is submitted only against baselined software or documentation. An SCR may be submitted by anyone associated with the project, but usually is submitted by a member of the software development team. A sample Software Change Request Form is provided in Appendix E. [3]

The Software Change Authorization (SCA) form is used to control changes to all software and documents under SCM control and software and documents that have been released to SCM. The SCA form is used by software developers to submit changes to software and documents to SCM in order to update the master library. A sample Software Change Authorization Form is provided in Appendix E. [3]

Several reports may be needed to support the configuration status accounting needs. Typical reports include a list of all SCRs by status (e.g., open, closed); a monthly summary of the SCRs and SCAs; all SCRs submitted per unit or software component within a selected range of submittal dates; the status of all documentation under SCM control; and a version description document. [3]

## 2.3 Configuration Status Accounting

The formal process of tracking software entities through the steps in their evolution is referred to as Status Accounting or Configuration Status Accounting (CSA). Status Accounting provides the project manager with the data necessary to gauge and report project progress.

Status Accounting provides a mechanism for maintaining a record of how the software has evolved and where the software is at any time relative to the information contained in baseline documents. Status Accounting answers the following questions: (1) What happened? (2) Who did it? (3) When did it happen? (4) What else will be affected? [6]

Status Accounting is a function that increases in complexity as the software lifecycle progresses because of the multiple software representations that emerge with later baselines. This complexity generally results in larger amounts of data to be recorded and reported. [5]

The purpose of Status Accounting is to have the ability, at any point in the software lifecycle, to provide the current content of a given software configuration item or computer software component. If the entity is a software design description, a status accounting mechanism should allow developers to easily pull together any text files, graphic files, and data files that may comprise the document. Likewise, change requests to software products should be easy to track and the status of these requests should be readily reproducible in an organized manner.

Each time a software configuration item is assigned a new or updated identification, a status report entry is made. Each time a change is approved, a status report entry is made. Each time a configuration audit is conducted, the results are reported as part of a status reporting task. Output from a status accounting report may be placed in an online database so that software developers or maintainers can access change information by key-word category. Status accounting reports are generated on a regular basis and are intended to keep management and practitioners appraised of important changes. [6]

Status accounting reports should be maintained in electronic files at logical transition points in the software lifecycle. For example, when a baseline configuration is advanced from the design phase to an implementation (coding) phase, a record of this should be filed in the configuration records. Similarly, when a newly developed or modified module has been approved for system testing and integration, this transition in status should be recorded.

## 2.4 Audits and Reviews

Audits and reviews are performed to verify that the software product matches the capabilities defined in the specifications or other contractual documentation and that the performance of the product fulfills the requirements of the user or customer. [3]

### Audits

Software configuration audits provide the mechanism for determining the degree to which the current state of the software mirrors the software depicted in baseline and requirements documentation. Software auditing increases software visibility and establishes the traceability of changes throughout the lifecycle. It reveals whether the project requirements are being satisfied and whether the intent of the preceding baseline has been fulfilled. The audits enable project management to evaluate the integrity of the software product being developed, resolve issues that may have been raised by the audit, and correct defects in the development process.

There are two types of audits that should be performed prior to release of a product baseline or a revision of an existing baseline: Physical Configuration Audit (PCA) and Functional Configuration Audit (FCA). A PCA is performed to determine whether all items identified as being part of the configuration are present in the product baseline. The audit must establish that the correct version and revision of each part are included in the product baseline and that they correspond to information contained in the baseline's configuration status report. An FCA is conducted to ensure that each item of the software product has been tested or inspected to determine that it satisfies the functions defined in the specifications. An important aspect of these two audits is the assurance that all documentation (change requests, test data, and reports, etc.) relevant to the release are current and complete. [3]

In large software projects, audits may be necessary throughout the evolution of a product to ensure conformance with planned configuration management actions and to prevent significant problems from being encountered just prior to release. There should always be an audit of the configuration items at the time a product is released (see section 2.5). This will vary according to the baseline being released and the criteria for the audit as stated in the SCM Plan.

### Technical Reviews

Formal technical reviews are performed throughout the project, although SCM disciplines generally do not initiate or direct reviews. Quite often the mechanisms used by SCM to process changes are used to organize and process items in a review conducted by other functions such as Software Quality Assurance. SCM supports reviews and provides the mechanism for acting on changes resulting from the reviews. [3]

## 2.5 Release Processing

Major releases of software must be described so that the recipient understands what has been delivered. Often the recipient will need installation instructions and other data concerning the release. The installation instructions should define the environment in which the software product will run—both hardware and software. The release package typically includes documentation (often referred to as the version description document). The more critical or the larger the software product, the more complete the documentation needs to be. SCM verifies that the release package is complete and ready to be delivered.

Once a release is completed, a configuration baseline should be captured that associates components and their versions with the release identity of the whole product. In this way, a release can be created or the presence of a component version can be traced. This feature is likely to be associated with a configuration management tool that controls software "builds" producing a report that lists all of the components. [8]

# 3. Tailoring Software Configuration Management to Different Software Environments

Software Configuration Management disciplines should be practiced as a part of every software engineering project. The completeness and level of detail for the configuration management disciplines depend on the size, complexity, and importance of the project.

For small user/developer projects a simple version control mechanism can be utilized to track development activities. One example is a three-digit numbering system (e.g., 1.2.3) where 1 is the number of the current accepted release; 2 identifies a certain level of development that has been verified as achieved (reset when the higher field changes), and 3 is changed with routine development (again reset when the higher field changes).

When more development freedom can be allowed, formal configuration management may apply once the developer has achieved a level of functionality or confidence to release to users. In this situation, a 'freeze' on development would occur with verification of the functionality of the software. Items to be configured would then be booked into a configuration management tool or library. All further changes or development would then be checked in and out, and controlled to a documented process. For larger, more complex software projects this formal configuration management process could be used at the start of the development activity. Appendix A provides a diagrammatic representation of how SCM fits into the software development lifecycle.

The level of formality associated with SCM depends on the risks to be mitigated. Using a broad definition of SCM as those processes capturing a logical snapshot of a dynamic development process, the formality can range from totally informal to fully controlled with auditable processes. Software with minimal risks associated with loss of historical or baseline information may have no formally defined process. For instance, the software developer may decide which versions should be kept based on a personal risk assessment. At the other end of the formality scale, large projects introduce processes to assure the capture of a useful set of final and intermediate deliverables with rules for changes to the deliverables. As there is a cost to the implementation of formal procedures, it is important to select processes that minimize risks proportioned to their costs.

The disciplines of SCM apply to the development of programmed logic, regardless of the form of packaging used for the application. Whether software is released for general use as programs (e.g., RAM or DRAM) or embedded in read-only memory (ROM), it is a form of logic. Therefore, SCM disciplines can and should be extended to include development of the software's component parts (e.g., source code and executable code).

Firmware raises some special considerations for configuration management. While being developed, the disciplines of SCM apply, but when made part of the hardware (e.g., burned into [EP]ROM or [EEP]ROM), the disciplines of hardware configuration management apply. Testing may vary, but the SCM requirements are generally the same. The packaging of [EP]ROM or [EEP]ROM versus RAM or DRAM code also introduces and necessitates different identification procedures.

# 4. Planning for Software Configuration Management

Planning for software configuration management (SCM) is essential to its success. The routine clerical-type functions associated with SCM are repetitious and can be automated fairly easily. The more important disciplines of SCM, such as defining a scheme for identifying the configuration items, components, and units; or the systematic review of changes before authorizing their inclusion in a program, are activities that require engineering judgment. Relating engineering judgment with management decisions, while also providing the necessary clerical support without slowing the decision-making process, is the critical role of SCM planning. Effective SCM involves planning how all of these activities are to be performed, and performing the activities in accordance with the plan. [3]

The planning and application of SCM is very sensitive to the context of the project and the organization being served. If SCM is applied as a corporate policy, it must be done in such a way that the details of a particular SCM system are reexamined for each project (or phase for very large projects). It must take into consideration the size, complexity, and criticality of the software project being managed; and the number of individuals, amount of personnel turnover, and organizational form and structure.

## Software Configuration Management Plan

The SCM system defines the interaction between a number of activities extending throughout the lifecycle of the product. The Software Configuration Management Plan (SCM Plan) functions as the focal point for integrating and maintaining the necessary details for the SCM system. The SCM Plan is a dynamic document that is maintained by approved changes throughout the life of the software product. [3]

The plan is used to:

- Identify the specific SCM concerns.
- Define what the SCM Plan will and will not address.
- Identify the configuration items and how they will be managed.
- State the actions to be performed by software engineering.
- State supporting activities that are required to maintain visibility of the evolving configuration of the software products.
- Support management in the process of evaluating and implementing changes to each configuration
- Assure that the changes have been properly and completely incorporated into each software product.

Control of changes to support third-party software presents a special challenge when planning SCM. Such software entities are usually maintained at a different site from the primary software product and are usually not subject to the same SCM Plan. Controls should be in place to identify the dependencies on third-party software and to ensure the continued availability of all required support software including compilers, operating system, etc. It may be desirable to retain support software and documentation along with the primary software product.

Maintenance of the SCM Plan throughout the life of the software product is especially important as the disciplines of identification, configuration control, status reporting, and release processing apply throughout the maintenance part of the lifecycle. Differences may be expected in how change processing is managed; and these need to be understood by all participants.

Projects differ in scope and complexity and a single format for all SCM Plans may not be desirable or applicable. ANSI/IEEE Std 828-1983 describes a format for plans with a minimum set of required information and a maximum amount of flexibility. Appendix B provides a template for an SCM Plan that is compliant with the ANSI/IEEE standard. If a section of the template is not applicable, insert wording that conveys there is no pertinent information for that section to indicate that the section has not been overlooked.

A review of each project's SCM Plan should be periodically performed to assess the effectiveness of the approach and the extent to which configuration management procedures are being followed by project staff. This enables adjustments to the SCM Plan to improve the staff's ability to follow the procedures and allows for more effective approaches to be incorporated as they are developed.

# 5. Automated Tools for Software Configuration Management

The Software Configuration Management (SCM) automated tools used for a project and described in the Software Configuration Management Plan (SCM Plan) need to be compatible with the software engineering environment in which the development or maintenance is to occur. SCM tools offer a wide range of capabilities and choices have to be made as to the tool set most useful for supporting the engineering and management environment. [2]

There are many different ways of evaluating SCM tools. One way is to categorize the tools according to characteristics of their products, such as: a filing system, a database management system, and an independent knowledge-based system. Another way is to examine the functions that the tools perform, such as: clerical support, testing and management support, and transformation support. A third way of categorizing the SCM tools is by how they are integrated into the software engineering environment on the project. [3]

## 5.1 Benefits of Using Tools

Configuration management of complex software may depend on ad hoc systems whose effectiveness is based on close adherence to written procedures. Such systems, which may be paper-based or electronic, require considerable effort to achieve the necessary levels of integrity and traceability, particularly for team development projects. Software tools designed for configuration management can simplify this process and provide better access to the information required by the developer, project manager, or user.

Software tools for configuration management can provide a number of benefits over paper-based systems, including: [4]

- Increased reliability and repeatability of the development process through automation of many mundane processes.
- Ability to track and manage changes within a user-defined process workflow.
- Enhanced productivity, by eliminating wasted effort and speeding up frequent activities such as the build and support of concurrent development.
- A means to manage product variants.
- Real-time analysis of all development activities.
- Complete audit trails of both software and documentation.
- More effective team management.

## 5.2 Types of Automated Tools

The automated tools described in this section are classified into broad categories in terms of the level of automation they provide to the programming environment on a project.

### 5.2.1 Basic Tool Set

The basic tool set is compatible with a programming environment that is relatively unsophisticated. The tools control the information on hard copy regarding a program product. These tools provide a capability that distinguishes between controlled and uncontrolled units or components. The tools simplify and minimize the complexity, time, and methods needed to generate a given baseline.

The basic tool set includes:
- Basic database management systems.
- Report generators.
- Means for maintaining separate dynamic and controlled libraries.
- File system for managing the check in and check out of units, for controlling compilations, and capturing the resulting products. [3]

### 5.2.2 Advanced Tool Set

The advanced tool set provides a capability for an SCM group to perform more efficiently on larger, more complex software engineering projects. These tools provide a programming environment that has more computing resources available. It provides the means of efficiently managing information about the units or components and associated data items. It also has rudimentary capabilities for managing the configurations of the product (building run-time programs from source code) and providing for more effective control of the libraries.

The advanced tool set includes:
- Items in the basic tool set.
- Source code control programs that will maintain version and revision history.
- Tools for comparing programs for identifying (and helping verify) changes.
- Tools for building or generating executable code.
- A documentation system (word processing) to enter and maintain the specifications and associated user documentation files.
- A system/software change request/authorization (SCR/SCA) tracking system that makes requests for changes machine-readable.
- Capability to manage concurrent development efforts. [3]

### 5.2.3 Online Tool Set

The online tool set requires an interactive programming environment that is available to the project. It also provides an organization with the minimal SCM capabilities needed to support the typical interactive programming environment currently available in industry. These tools provide online access to the programming database and the resources necessary for using the tools.

The online tool set includes:

- Generic tools of the advanced tool set integrated so they work from a common database.
- An SCR/SCA tracking and control system that brings generation, review, and approval of changes online.
- Report generators working online with the common database, and an SCR/SCA tracking system that enables the SCM group to generate responses to online queries of a general nature. [3]

### 5.2.4 Integrated Tool Set

The integrated tool set integrates the SCM functions with the software engineering environment so that the SCM functions are transparent to the engineer. The software engineer becomes aware of the SCM functions only when he/she attempts to perform a function or operation that has not been authorized (for example, changing a controlled entity when the engineer does not have the required level of authority or control).

An integrated tool set includes:

- Online SCM tools covering all functions.
- An integrated engineering database with SCM commands built into the online engineering commands commonly used in designing and developing programs (most functions of CM are heavily used during design and development phases).
- The integration of the SCM commands with online management commands for building and promoting units and components. [3]

## 5.3 Tool Selection

SCM tools should be carefully selected to match working practices and ideology. Selection of SCM tools should be based on the following features.

- Cross-platform support.
- Developer empowerment (librarian or other supervisory role optional).
- Match to existing work practices (task or lifecycle based).
- Tool integration (between components and other software engineering tools).
- Ease of installation and use.
- Code visibility outside tool (to support intranet technology).
- Supplier support before and after delivery.
- Market position of supplier and product.
- Overall cost.

While the selection of tools will be influenced by the functional requirements, tools are characterized by a series of additional attributes that reflect software engineering practices. These attributes are briefly summarized below. [4]

**Team Support**

| Feature | Description |
|---|---|
| Workspace Management | Independent workspaces, enabling developers to work autonomously and yet be informed of any relevant changes |
| Concurrent Development | Developers work on file(s) concurrently; tools have merge facilities to ease amalgamation |
| Tool Integration | Interface with development environments and other tools |

**Remote Development**

| Feature | Description |
|---|---|
| Distributed Development | Support for synchronization and integration of projects developed at different sites |
| Third-Party Software | Ability to manage subcontracted software development |
| Internet Support | Use of the Internet for file check-in, check-out |

**Configuration Management**

| Feature | Description |
|---|---|
| Configuration Specification | Record of information pertaining to a specific build |
| Impact Analysis | Calling trees and dependencies |
| Traceability | Relationships between items |

**Change Management**

| Feature | Description |
|---|---|
| Problem Tracking | Complete record of a problem history, from initial logging to final close-out |
| Change Control | Authorization and approval for change |
| Change Sets | File-based or task-based changes |
| Reports and Metrics | Dynamic provision of management information |

**Build and Release Support**

| Feature | Description |
|---|---|
| Building<br>Re-building | Automation of the build process<br>Identification of source files of the original build |
| Release Support | Formal release and customer records |

**Process Management**

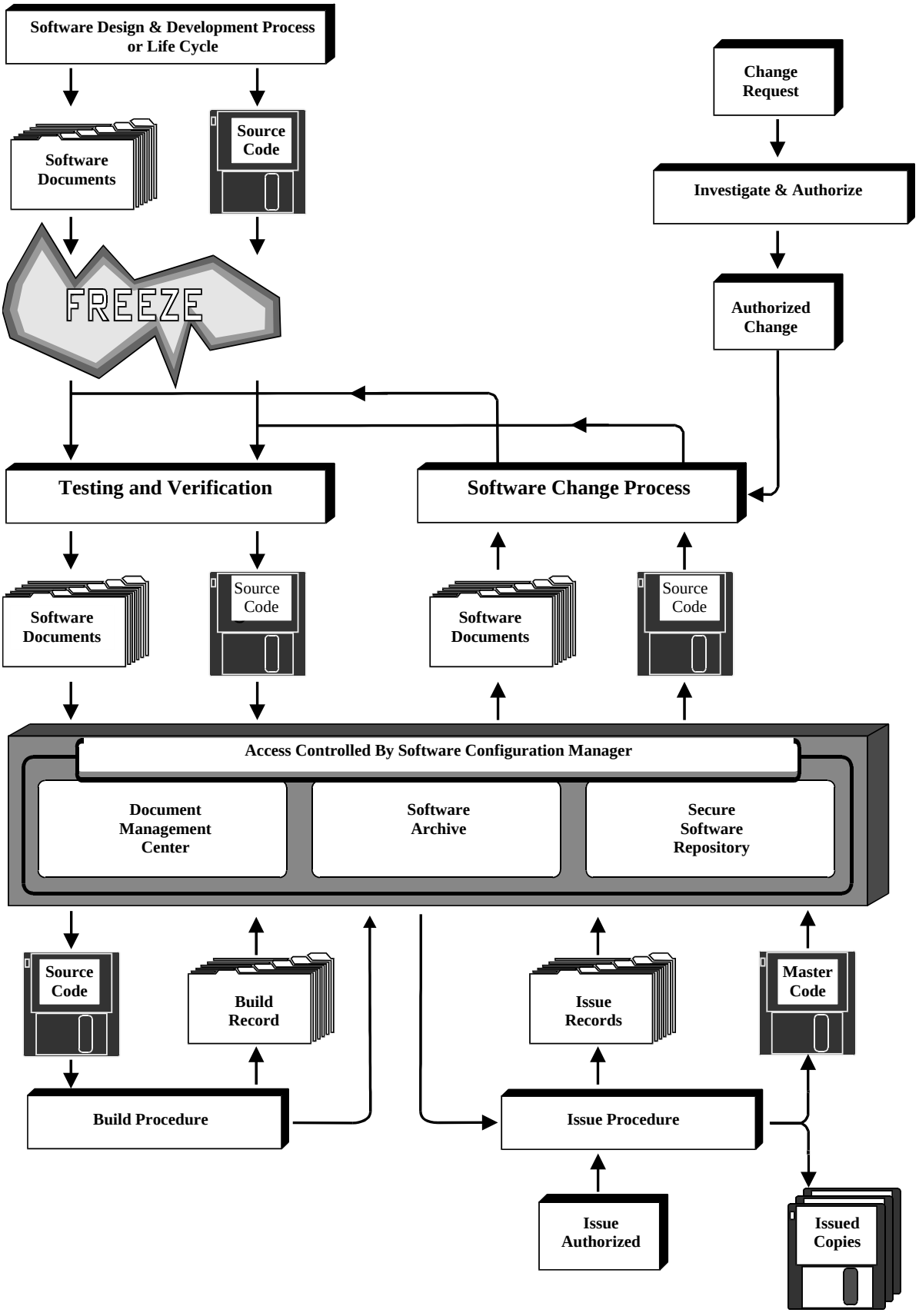
| Feature | Description |
|---|---|
| Lifecycle Support | Support of customized lifecycle stages for different product components |
| User Roles | Promotion through the lifecycle with user-defined roles, with associated access permissions and responsibilities |

# Appendix A: Diagrammatic Representation of Software Configuration Management and Interfacing Processes

This appendix provides a detailed example of one method for implementing Software Configuration Management (SCM) and its interfacing processes.

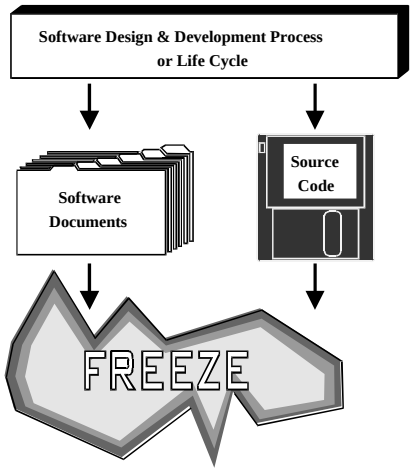
SCM systems interface with all other software processes. The diagram on the next page represents how these various processes link to a SCM system that consists of a document management center, a software archive, and a secure software repository.

The text following the diagram describes each process and its interfacing arrangements to the SCM system.

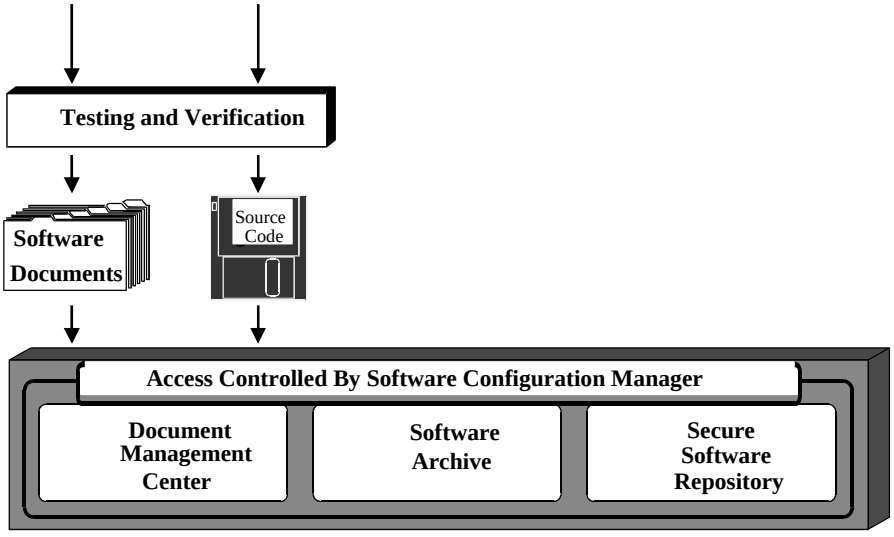

Software Design & Development Process or Life Cycle
Software Documents
Source Code
FREEZE
Change Request
Investigate & Authorize
Authorized Change
Testing and Verification
Software Change Process
Software Documents
Source Code
Software Documents
Source Code
Access Controlled By Software Configuration Manager
Document Management Center
Software Archive
Secure Software Repository
Source Code
Build Record
Issue Records
Master Code
Build Procedure
Issue Procedure
Issue Authorized
Issued Copies

## Design and Development Procedure

Software will be developed to a design and development process or lifecycle. The deliverables for this process could include source code with its associated documentation.

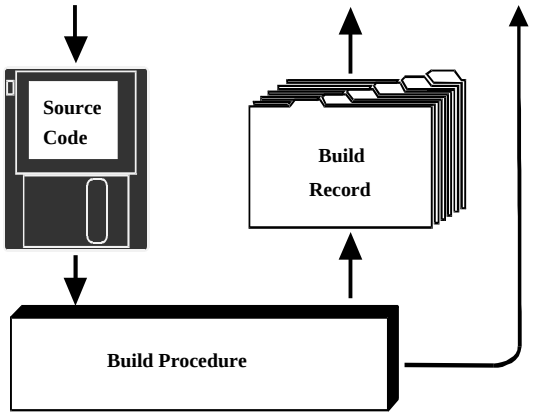



As development is completed or the project is at a predetermined point such as a software module, the design is frozen and then verified against its specified requirements. This will be tested via a test procedure that will also generate results. All items that constitute the software and its supporting documentation will be identified and then placed under formal configuration management. This is achieved by placing the software under the control of the software configuration management system and entered into a secure repository. These configured items would normally consist of a single set of definitive modules.

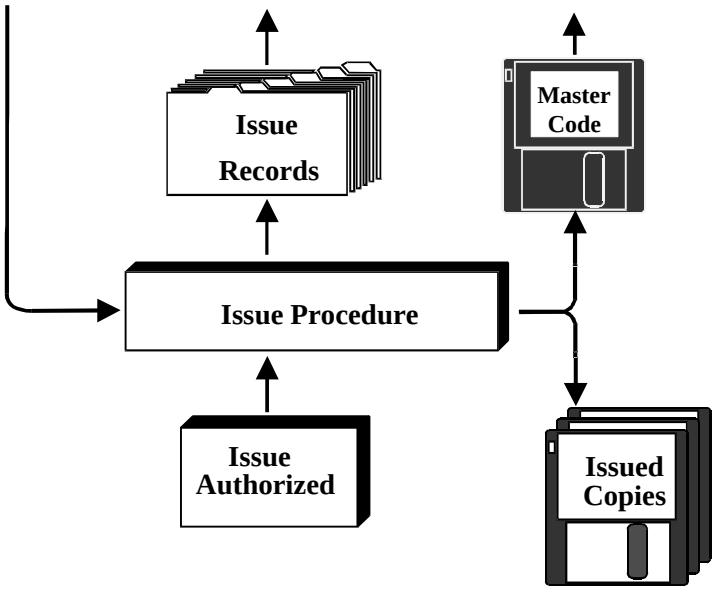


## Build Procedure

Assembling the software system will usually involve the use of tools to transform the source item or modules produced by the developer into executable programs. The exact procedure used to build a piece of software can affect its operation and must therefore be controlled and documented.




The build procedure includes definition of the tools involved (including versions) and their settings. It will possibly call up “make” files, “linker” files, “batch” files, etc. These files must be managed in the same way as source modules and be subject to the same controls. The decision to build a new version must be appropriately authorized and documented. The documentation (build record) shall include the new version number and a list of all the relevant modules (with their version numbers). The build procedure could include the input of the overall version number into the software. Following the build, the new version of the program, together with all of the source files required (including library files, header files, make and linker files) must be archived. This archive will be under control of the Configuration Manager. The archive should provide the facility to rebuild the program at some future date.

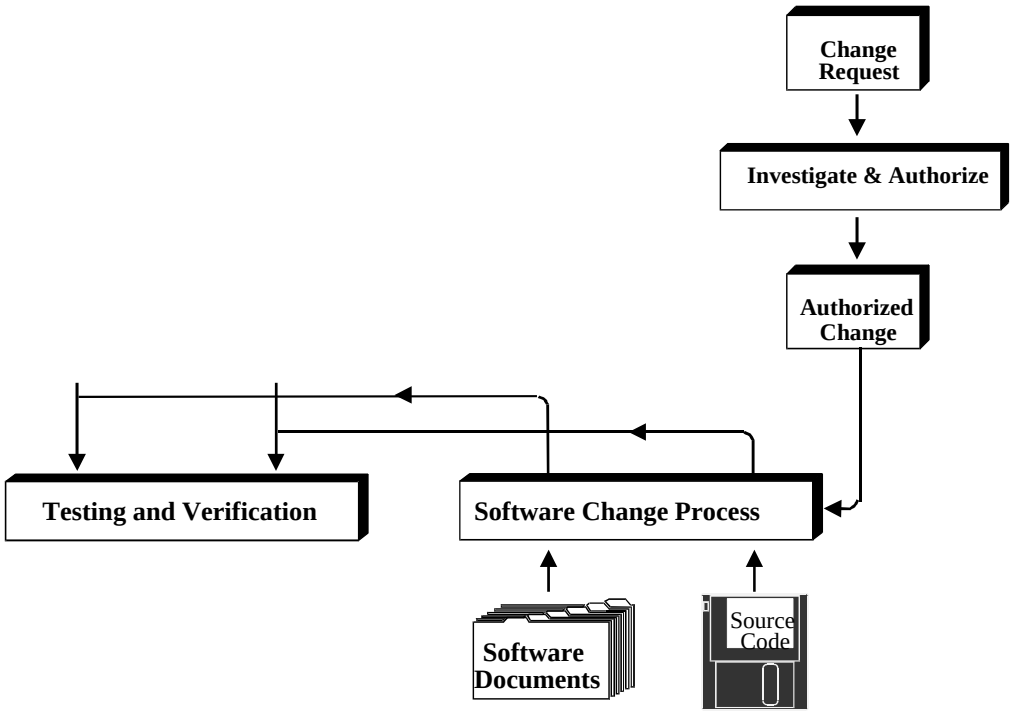

Issue
Records
Master
Code
Issue Procedure
Issue
Authorized
Issued
Copies

## Issue Procedure

Each new version of a program should be used to produce a new master disk(s) stored under controlled access conditions. A unique serial number should be allocated and documentation up-dated. Any old master disks should be archived. The master disks should not be used for any other purpose than creating other disks. The program should never be run from them and a master disk should be "write protected" after creation. Access to the master disk should be restricted to the Configuration Manager or designated deputy.

## Change Procedure

During the lifetime of a software module it is likely that it will become necessary to modify it. It is essential that modifications be carried out in a controlled manner. A software change is initiated by the submission of a Software Change Request. It must specify the change required or fault reported, reasons for the change, and an indication of priority.




Once the Software Change Request has been submitted it should be investigated. The investigation should identify the software modules affected including any effects on other modules or overall program features and the effort required to implement and verify the change. Reference to the module(s) and documentation held in the software repository should be made. The Software Change Request must be authorized by the appropriate level of authority. Once authorized, the change can be implemented. The software change then enters the testing, verification, and review process. Following review the new version is authorized and issued; the previous version (software and documentation) should be archived. It is essential that all associated documentation is modified and re-issued appropriately; the change documentation should also be archived.

Consideration should be given to informing all users of the availability and reason for the new version. The withdrawal of old versions should be controlled and documented.

## Software Repository

The software repository is a secure container with access controlled by the Configuration Manager. It may simply be a drawer with a store for disks and a file with appropriate control forms. It is emphasized that the control of the software documentation is as important as the actual software.

# Appendix B: SCM Plan Template

## Software Configuration Management Plan

The following sample outline provides a brief description of the content of each section and subsection for a SCM Plan. [3] Each section should be further subdivided to reflect the level of detail appropriate for the project for which the SCM Plan applies.

### 1. INTRODUCTION

- *Purpose* – Provide information on the requirements and procedures necessary for the configuration management of the software engineering project. State that the plan identifies the SCM requirements and establishes the methodology for generating configuration items, controlling engineering changes, maintaining status accounting, and performing audits and reviews.

- *Scope* – Identify the phases of the software development lifecycle addressed by the plan. State that the plan applies to all software and related documentation used in the production of the software product.

- *Definitions and Abbreviations* – Provide definitions of the terms used in the plan or cite an industry standard source for definitions such as the ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology. Provide full text wording for all abbreviations used in the plan.

- *References* – Identify the standards and other resources used in developing the SCM Plan.

- *Management* – Define and describe the framework within which administrative and technical control of SCM activities will be integrated into the software engineering project. Describe organizational features that impact the SCM system planned for the software engineering project. Identify all SCM policies, directives, and procedures that apply to the project.

### 2. CONFIGURATION IDENTIFICATION

- *Responsibility* – Designate appropriate responsibility and authority for SCM identification activities.

- *Identify Software Configuration Items* – Identify the software and documentation items in the project that are subject to change. Provide detailed information on how to identify each item to be controlled. (Refer to section 2.1, Configuration Identification, for information about configuration items.)

## 3. CONFIGURATION CONTROL

- *Responsibility* – Designate appropriate responsibility and authority for SCM control activities.

- *Third-Party Controls* – Describe the controls that will be implemented for software provided by subcontractors or vendors.

- *Configuration Control Board (CCB)* – Describe the responsibilities of those involved with SCM. A list of CCB members, or the titles of members, should also be included. (Refer to section 2.2.2, Levels of Authority, for information on configuration control boards.)

- *SCM Libraries* – Define the SCM libraries required to maintain controlled SCM environments. Note any differences in the SCM procedures for each library. (Refer to section 2.2.1, Procedures for Controlling Changes, for information on software libraries.)

- *Baseline Change Control Procedures* – Define software and documentation problem reporting, software change procedures, and software release methods. Outline the procedures for processing various types of changes to the identified baselines, including utilization of the Change Control Board, change control forms, testing, and DOE interfaces. (Refer to section 2.2, Configuration Control, for information about change control procedures.)

## 4. CONFIGURATION STATUS ACCOUNTING

- *Responsibility* – Designate appropriate responsibility and authority for SCM status accounting activities.

- *Recording* – Define the tracking processes for collecting, storing, handling, and verifying software configurations.

- *Reporting* – Define the method by which information will be presented to management. Both ad hoc inquiry capability and periodic milestone reporting should be covered.

  (Refer to section 2.3, Configuration Status Accounting, for related information.)

## 5. AUDITS AND REVIEWS

- *Responsibility* – Appropriate responsibility and authority must be designated for SCM auditing activities.

- *Audits* – Describe and provide the schedule for the planned project audits including the Physical Configuration Audits and the Functional Configuration Audits.

- *Baseline Review Procedures* – Provide plans for the required project reviews. Typical project reviews include the critical design review, system requirements review, and software specifications review.

  (Refer to section 2.4, Audits and Reviews, for related information.)

## 6. TOOLS, TECHNIQUES AND METHODOLOGIES

Describe the automated tools, techniques, and methodology that will be used to perform the activities described in the SCM Plan. Include records collection and retention plans for all SCM artifacts.

## APPENDIXES

- *Forms and Instructions* – Attach any forms and their instructions that are identified in the SCM Plan and used in the SCM system such as the Software Change Request and Software Change Authorization forms.

- *System Definition* – Describe the owners and potential users of the project, the hardware and software configuration, interfaces, and other vendor software. Also identify any further enhancements that may cause a redefinition of the system.

- *Site-Specific Software Configuration Management Plan* – Describe any site-specific SCM that may vary according to remote system locations.

# Appendix C: SCM Checklist

- ☐ Create procedures to ensure identification and control of all configuration items and baselines including necessary changes to those items.
- ☐ Identify appropriate tools or methods to support the configuration system including change control and methods of backup.
- ☐ Document procedures for review and authorized release of products consistent with the level of testing applied.
- ☐ Develop a mechanism for coordinating the updating of software at all customer locations.
- ☐ Create procedures for replication and subsequent verification and product identification activities.
- ☐ When a configuration management software tool is employed, clearly document the use of that tool.
- ☐ Develop a mechanism to support the labeling or other means of identification of third-party supplied products including, where necessary, integration into the configuration management system.
- ☐ Develop a mechanism to ensure that software or designs produced or modified externally to the organization (for example by a subcontractor) are fully integrated into the configuration control process.
- ☐ Determine how the software configuration management system will be tailored to accommodate the size and complexity of the project.

# Appendix D: Glossary and Abbreviations

The following abbreviations and definitions apply to the terms used in this practical guide.

**Baseline** – A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures. (2) A document or set of documents formally designated and fixed at a specific time during the lifecycle of a configuration item. [1]

**Component** – One of the parts that make up a system. A component may be hardware or software and may be subdivided into other components. [1]

**Configuration Control** – An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification. [1]

**Configuration Control Board (CCB)** – A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. [1]

**Configuration Identification** – An element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation. [1]

**Configuration Item (CI)** – An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process. [1]

**Configuration Management (CM)** – A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements. [1]

**Configuration Status Accounting (CSA)** – An element of configuration management, consisting of the recording and reporting of information needed to manage a configuration effectively. This information includes a listing of the approved configuration identification, the status of proposed changes to the configuration, and the implementation status of approved changes. [1]

**DRAM** – Dynamic Random Access Memory

**[EEP]ROM** – Erasable Electrically Programmable Read Only Memory

**[EP]ROM** – Electrically Programmable Read Only Memory

**Entity** – Any component of a software product such as source code, specifications, documents, development tools, and test data.

**Firmware** – The combination of a hardware device and computer instructions and data that reside as read-only software on that device. [1] Computer software and data loaded in a class of memory that cannot be dynamically modified by the computer during processing.

**Functional Configuration Audit (FCA)** – An audit conducted to verify that the development of a configuration item has been completed satisfactorily, that the item has achieved the performance and functional characteristics specified in the functional or allocated configuration identification, and that its operational and support documents are complete and satisfactory. [1]

**Physical Configuration Audit (PCA) -** An audit conducted to verify that a configuration item, as built, conforms to the technical documentation that defines it. [1]

**RAM** - Random Access Memory

**ROM** - Read Only Memory

**SCA** - Software Change Authorization

**SCCB** - Software Configuration Control Board

**SCM** – Software Configuration Management

**SCMP** – Software Configuration Management Plan

**SCR** - Software Change Request

**Software** – Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system. [1]

**Software Configuration Management -** The process of identifying and defining the software configuration items in a system, controlling the release and change of these items throughout the system lifecycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items.

**Software Engineering** – The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

**Software Library** – A controlled collection of software and related documentation designed to aid in software development, use, or maintenance. [1]

**Software Product** – The complete set of computer programs, procedures, and possibly associated documentation and data designated for delivery to a user. [1]

**Software Repository** – A software library that provides permanent, archival storage for software and related documentation. [1]

**Source Code** – Computer instructions and data definitions expressed in a form suitable for input to an assembler, complier, or other translator.

**Unit** – A separately testable element specified in the design of a computer software component. A software component that is not subdivided into other components. [1]

**Version** – An initial release or re-release of a computer software configuration item, associated with a complete compilation or recompilation of the configuration item. [1]

# Appendix E: Sample Configuration Management Forms

## E.1 Software Change Request Form [1]

### System/Software Change Request

SCR Number _____

1. Submitted by: _____ Date: / /

   Project Name: _____

2. Software Program/Document Name: _____

   Version/Revision _____

3. SCR Type: (1- Development, 2 - Problem, 3 - Enhancement)

4. Short Task Description:

5. Detail Description

6. Submitter's Priority [ ] 1 = Critical 2 = Very Important 3 = Important 4 = Inconvenient 5 = Interesting

7. CCB Action: _____ CCB Priority [ ]

8. Assigned to: _____ Target Release Date

9. Solution Comments:

10. Other Affected Software Configuration Items:

11. I&T Approval _____ Date: _____

    SCM Approval _____ Date: _____

12. Actual Release _____ Date: _____

13. Closed by _____ Date: _____

    SCA Reference No. _____

14. SQA Approval _____ Date: _____

[1] **This form was modified from the original version that appeared in ANSI/IEEE Std 1042-1987**

# E.2 Software Change Authorization Form[2]

## Software Change Authorization

SCA Number:
Sheet Number: 1
Submitter: System: Date / / Time: / / 00:00:00

Product Version ID: _____ Computer Name _____

| Input Names | Release Names | Module Types | L N | A C | System/Software Change Request Numbers | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

Comments

| Approvals | Integration & Test (I&T) | SCM | SQA |
|---|---|---|---|
| Signature | | | |
| Date | | | |

[2] **This form was modified from the original version that appeared in ANSI/IEEE Std 1042-1987**

# Appendix F: References/Resources

The following publications were used in the preparation of this guide. Later versions of some of the standards may have been published after the research was performed for this guide.

[1] ANSI/IEEE Std 729-1983, *IEEE Standard Glossary of Software Engineering Terminology.*

[2] ANSI/IEEE Std 828-1990, *IEEE Standard for Software Configuration Management Plans.*

[3] ANSI/IEEE Std 1042-1987 (Reaff 1993), *IEEE Guide to Software Configuration Management.*

[4] AWE/DOQ/15/16/00-AT44, *A Guide to Software Configuration Management*, AWE Hunting-Brae, December 24, 1997.

[5] Bersoff, Edward H., "Elements of Software Configuration Management," *IEEE Transactions on Software Engineering*, January 1984.

[6] Pressman, R., *Software Engineering: A Practitioner's Approach*, 3rd ed., McGraw-Hill, New York, NY, 1997.

[7] SAND85-2347, *Sandia Software Guidelines, Volume 4, Configuration Management*, Sandia National Laboratories, 1992.

[8] The TickIT Guide, A Guide to Software Quality System Construction and Certification to ISO 9001, British Standards Institution, London, January 12, 1998.

[9] DOE Software and Systems Engineering Web site: http://cio.doe.gov/smp

[10] Dorofee, Audrey J., et al, *Continuous Risk Management Guidebook*, Carnegie Mellon University, 1996.

[11] Carnegie Mellon University Software Engineering Institute Web site: http://www.sei.cmu.edu