NASA
SOFTWARE CONFIGURATION MANAGEMENT GUIDEBOOK

AUGUST 1995

PREFACE

The growth in cost and importance of software to NASA has caused NASA to address the improvement of software development across the agency. One of the products of this program is a series of guidebooks that define a NASA concept of the assurance processes which are used in software development.

The Software Assurance Guidebook, SMAP-GB-A201, issued in September, 1989, provides an overall picture of the concepts and practices of NASA in software assurance. Lower level guidebooks focus on specific activities that fall within the software assurance discipline, and provide more detailed information for the manager and/or practitioner.

This is the Software Configuration Management Guidebook which describes software configuration management in a way that is compatible with practices in industry and at NASA Centers. Software configuration management is a key software development process, and is essential for doing software assurance.

SOFTWARE CONFIGURATION MANAGEMENT GUIDEBOOK

Approvals

_____

Lawrence E. Hyatt

Manager, Software Assurance Technology Center

_____

Kathryn Kemp

Deputy Director, NASA Software Technology Division

I. GENERAL

This guidebook defines Software Configuration Management (SCM) and describes its constituent functions, processes, and procedures. The guidebook also describes a generic set of organizational elements and responsibilities for operation of SCM. It defines the role of SCM, its interfaces and its functions throughout the software development life cycle. This guidebook also provides a basis for tailoring SCM for different projects with different life cycles and project specific requirements.

Proper application of software configuration management is a key component in the development of quality software. Changes to the software under development are usually a significant cause of changes to a project's schedule and budget; unmanaged change is very possibly the largest single cause of failure to deliver systems on time and within budget.

SCM is the process that has been developed to control and manage change. Change is inevitable during the software development life cycle. Changes to the software come from both external and internal sources. External changes originate from users, from evolution of operational environments, and from improvements in technology. Internal changes come from improved designs and methods, from incremental development, and from correction of errors. A properly implemented SCM process is the project manager's best friend and potential salvation in coping with change.

This guidebook is written for software project managers who must plan for SCM for their project, for SCM practitioners who will implement SCM, and for software developers and acquirers who will be affected by it. The style of the guidebook is intended to be tutorial rather than directive. It is hoped that the reader will find the following sections an easily understood introduction to software configuration management and a useful guide to planning and implementing SCM in a software development project.

The guidebook describes SCM in terms of the concepts and definitions, implementation, applicability in relation to the software life cycle phases, organization and interfaces, discussions of tools, examples of forms, and references to applicable documents.

II. CONCEPTS AND DEFINITIONS

Software configuration management is the process whose objective is the identification of the configuration of software at discrete points in time and the systematic control of changes to the identified configuration for the purpose of maintaining software integrity and traceability throughout the software life cycle.

In order to accomplish the objective given in the above definition, there are four identified SCM functions: 1) identification of the components that make up the software system and that define its functional characteristics; 2) control of changes to those components; 3) reporting of status of the processing of change requests and, for approved requests, their implementation status; and 4) authentication that the controlled items meet their requirements and are ready for delivery.

The components of a software system that are controlled by the SCM process include project documentation, product documentation, code, data, and any other items needed to meet a set of requirements or contractual obligations. All of these items can be controlled by the same SCM process.

The term "configuration" is used repeatedly in this guidebook. Configuration means the functional and/or physical characteristics of software as set forth in technical documentation and realized in a product. Thus "configuration" includes all of the characteristics of the software to be controlled - its content, the content of documents that describe it, the versions of software and documents as these contents are changed, data needed for operation of the software, and any other essential elements or characteristics that make the software what it is.

The software under control is usually divided into "configuration items." Configuration item (CI) is the term used for each of the logically related components that make up some discrete element of software. For example, if a system contains several programs, each program and its related documentation and data might be designated a configuration item. The number of CIs in a system is a design decision. Guidelines for division of software systems into CIs are given in Section III, part B1. Note that if the system being developed has both hardware and software components, then in a general sense a CI may have both hardware and software components. A CI that is purely software is often called a Computer Software Configuration Item, or CSCI. Generally in this guidebook we will use the term CSCI, since we are talking about software systems and software configuration management. The concepts for software CM are similar to hardware CM, but the software CM process must be even more rigorous and deeply imbedded in the engineering process since software is much more flexible and changeable than hardware.

As each software CI goes through its development process, more and more of its components are developed until the final CSCI is available for use. Generally, the life cycle process will first result in a set of requirements, then a design, then code for individual elements of the CSCI, then integrated code with test cases and user manuals, etc. The definition of SCM contains the concept of identifying the configuration of each CSCI at discrete points in time during the life cycle process, and then managing changes to those identified configurations. The configuration of software at a discrete point in time is known as a baseline. Thus, a baseline is the documentation and software that make up a CSCI at a given point in its life cycle. Each baseline serves as a point of departure or reference for the next development stage. In the NASA standard life cycle, baselines are established after each life cycle phase at the completion of the formal review that ends the phase.

Each baseline is subject to configuration control and must be formally updated to reflect approved changes to the CSCI as it goes through the next development stage. At the end of a life cycle phase, the previous baseline plus all approved changes to it becomes the new baseline for the next development stage. The term "baseline management" is often used to describe this control process.

Normally, the first baseline consists of an approved software requirements document, and is known as the requirements baseline. Through the process of establishing a baseline, the functional and other requirements described in the requirements document become the explicit point of departure for software development, against which changes can be proposed, evaluated, implemented, and controlled. The requirements document is also the basis against which the

software is authenticated. Subsequent baselines and their contents are described in Section IV.

SCM requires that processes and procedures be put in place to identify the baselines that are to be established and their contents, to control the CSCIs and their individual elements to prevent unauthorized change, to process requests for changes to those baselines and report on status of changes, and to authenticate that a given configuration contains all of its required components and that it satisfies the functional and performance requirements given in the requirement document. That is, SCM consists of four basic processes:

- Configuration Identification
- Configuration Control
- Configuration Status Accounting
- Configuration Authentication

These processes, their relationships and implementation, are described in detail in Section III.

There are some roles that are key to the SCM process. Someone must operate the SCM process; that is, must establish detailed procedures, must make sure all requests for changes are processed properly, must provide reports on the status of all CSCIs and proposed changes, and must have control of all of the baselined items. This role is named Configuration Management Officer or CMO. The individual designated the CMO is critical to the successful operation of a SCM system.

Actual storage and physical control of the contents of baselines is done by a librarian in a location called a program library. The program library must contain the official copies of all baselined items that make up the various CSCIs. It contains all baselined items, including code and object modules, which are checked out by the librarian for authorized changes to be made, and are checked back in after change is complete. The program library is operated by the librarian and is usually under the control of the CMO. Efficient operation of the library is enhanced if automated tools are available. See section VIII for a discussion of tools.

Decision making authority, which must determine if a proposed change is to be made, is vested in a Configuration Change Board (CCB). A CCB is chaired by a senior manager (often the project manager) who can authorize the expenditure of resources. Other members are chosen based on their ability to provide advice on the costs and benefits of a change. Usually the CCB rules of operation are such that the chair unilaterally decides the disposition of the proposed changes after receiving the advice of the other members. The CCB process is operated by the CMO, who provides to the CCB the requests for changes and the associated analysis of impact and who records the decisions of the CCB and provides them to the change requester and the individuals who will implement the change.

SCM is a key process in managing software development, operation, maintenance, and enhancement. The remainder of this document explain in detail the component processes and steps, and the organizational elements needed for the successful implementation of SCM in a software development project.

III. THE SCM PROCESS

A. Overview

Software configuration management is composed of four functions:

- Configuration Identification
- Configuration Control
- Configuration Status Accounting
- Configuration Authentication

Configuration identification is the process of defining each baseline to be established during the software life cycle and describing the software configuration items and their documentation that make up each baseline. First, the software must be grouped into configuration items. Once the CSCIs and their components have been selected, some way of designating the items must be developed. This is done by the development of a numbering and naming scheme that correlates the code and data items with their associated documentation. Finally, the CSCIs must be described by the documentation of their functional, performance, and physical characteristics.

Configuration control is the process of evaluating, coordinating, and deciding on the disposition of proposed changes to the configuration items, and for implementing approved changes to baselined software and associated documentation. The change control process ensures that changes which have been initiated are classified and evaluated, approved or disapproved, and that those approved are implemented, documented, and verified.

Configuration status accounting is the process used to trace changes to the software. It ensures that status is recorded, monitored, and reported on both pending and completed actions affecting software baselines. This process also defines the current as-built status of the code and associated documentation.

Configuration authentication is the process of verifying that a deliverable software baseline contains all of the items which are required for that delivery, and that these items have themselves been verified, i.e., they satisfy their requirements. The authentication function usually consists of two "audits": a functional configuration audit (FCA) and a physical configuration audit (PCA). Functional audits authenticate that the software has been tested to assure that it performs in accordance with requirements in the baseline documentation. Physical audits authenticate that the software to be delivered contains all of the required components, documents, and data.

Each of the above functions is explained in more detail in the following sections.

B. Configuration Identification

Configuration identification is the basis for subsequent control of the software configuration. The configuration identification process involves the selection, designation, and description of the

software configuration items. Selection involves the grouping of software into configuration items that are subject to configuration management. Designation is the development of a numbering and/or naming scheme that correlates the software components and their associated documentation. Description is the documentation of functional, performance, and physical characteristics for each of the software components.

1. CSCI Selection

A software system is generally split into a number of CSCIs which are independently developed and tested, and which are finally put together at the software system integration level. Each CSCIbecomes essentially an independent entity as far as the CM system is concerned, and the four processes defined at the beginning of Section III are carried out on each CSCI.

The division of the software into CSCIs may be contractually specified, or may be done during the requirements analysis or preliminary design phase. As a general rule, a CSCI is established for a separable piece of the software system that can be designed, implemented, and tested independently. Other criteria that may go into the decision to separate out a set of software and manage it as a CSCI are:

- The set of software is critical to the overall performance, or there is a high level of risk involved, or system safety related tasks are contained in the item.
- The set of software is highly complex, incorporates new technologies, or has stringent performance requirements.
- The set of software encapsulates interfaces with other software items that currently exist or are provided by other organizations.
- The set of software is expected to have more than usual change or modification after it becomes operational.
- The set of software contains all of a specific domain of functionality such as application, operating system, etc.
- The set of software is installed on a different computer platform from other parts of the system.
- The set of software is planned to be reused.

2. CSCI Designation

Each software component must be uniquely identified. The unique identifier is used in tracking and reporting CSCI status. Normally, each CSCI is assigned a CSCI identifier, and pieces of the CSCI are given the identifier with an attached descriptor. Thus, a CSCI that consists of the flight software for a spacecraft might be given the designator FS. Components of the flight software, such as the flight executive might be designated FS-EX, showing that it is a second level component of the CSCI FS. Parts (subroutines) of the executive might be numbered FS-EX-001, FS-EX-002, etc. This allows the unique identification of system elements, both code and documents, as to the CSCI of which they are parts. It facilitates the tracking of changes to the

component, the status of the changes and the component, and the reporting of all the information relative to the component.

3. CSCI Description

Software components are described in specifications (i.e., software requirements specifications, software architectural design specifications, software detailed design specifications, interface control documents, and software product specifications). The description of the component becomes more detailed as the design and development proceeds through the life cycle The description forms the basis for configuration control and configuration status accounting. The description is also the basis for the final authentication that the software is complete and verified. The documents, or portions of documents, that describe each CSCI must be identified and made part of the CSCI.

Formats for the documents described above can be found in NASA-STD-2100-91, "NASA Software Documentation Standard".

C. Configuration Control

Configuration control is the systematic process for evaluating, coordinating, and deciding on the disposition of proposed changes and for tracking the implementation of those approved changes to baselined code and associated documentation and data. The change control process ensures that the changes are

initiated, classified, evaluated, approved or disapproved, documented, implemented, tested, and incorporated in a new baseline.

An orderly change process is necessary to ensure that only approved changes are implemented into any baselined document or software. Figure 1 shows a simple overview of the change process. The steps within the overall process can be grouped into the following categories:

- Change Initiation
- Classification
- Change Evaluation
- Change Dispositioning
- Implementation
- Verification
- Baseline Change Control

These seven steps in change control are individually discussed below:

1. Change Initiation

Requests for change to software and documents come from many sources. A Change Request

(CR) may be submitted by a potential user, by the system acquirer, by a reviewer, or by a member of the provider's staff. Each project should set up a CR form for documenting the proposed change and its disposition. See the NASA Software Standards, DID-R005, for basic information that can be included on a CR form, however, the actual form set up must correspond to the planned SCM process. An example form, based on the DID, is included as Appendix B. Note that electronic forms, containing the same information, are being increasingly used as direct interfaces to SCM support tools (Tools are discussed in Section VIII). Each project should also name an individual (the Configuration Management Officer, or CMO) to receive the change form, assign it a tracking number and classification, and route it for processing.

The CMO receives the CR and reviews it for clarity and completeness. If the CMO determines that the CR is not complete, it is returned to the originator. Once complete, the CMO assigns the CR a unique identifier for tracking purposes and records information about the CR in the change request tracking data base or files.

2. Change Classification

Changes to software and associated documentation are classified according to the impact of the change and the approval authority needed. The top class is assigned to changes that would affect the system level requirements, external interfaces, system cost, and/or delivery schedule. These changes usually can only be approved by the acquirer of the software. A second class of changes may be established for changes that affect the interfaces between CSCIs and the allocation of functions to CSCIs, or which affect component level cost and schedule. These changes generally can only be approved by the project level management of the provider. A third class of changes, those that affect CSCI internal design and division of functionality, may be approved by CSCI level management.

Generally, change classes are identified by numbers. For instance, Level I is often assigned to that class that contains the highest class and has to be routed to the acquirer; Level II to the next; etc. Class names and classification rules should be spelled out in the agreement (contract) between the acquirer and provider and the processes for assignment of classes should be in the provider's SCM procedures. In addition to defining the scope of changes that are allowed to be made by the provider, the contract often will require that the acquirer be provided copies of at least the next lower class to allow verification of the classification process.

Classification of changes may be suggested by the individual who proposes the change. The CMO reviews suggested classes and assigns a working classification. After assessment of the impact of the CR, the CCB will assign the final class.

3. Change Evaluation

One important aspect of the Configuration Control process is that it provides adequate analysis of changes in terms of impact to system functionality, interfaces, utility, cost, schedule, and contractual requirements. Each change should also be analyzed for impact on software safety, reliability, maintainability, transportability, and efficiency. The project CMO routes the CR to the software engineering staff for evaluation. In some cases, project procedures require that the CR be screened by some board before it is analyzed. This approach saves the cost of analysis for changes that do not have any chance of approval.

The analysis produces documentation which describes the changes that will have to be made to implement the CR, the CSCIs and documents that will have to be changed, and the resources needed to do the change. The documentation becomes part of the change package, along with the CR. After completion of the analysis, the change package is sent back to the CMO.

4. Change Dispositioning

Dispositions for changes to baselined items are done by a Configuration Control Board (CCB). The CCB evaluates the desirability of a change verses the cost of the change, as described in the documentation of the analysis. The CCB may approve, disapprove, or defer a change request. It may have to request more information and additional analysis.

Dispositioned items are sent to the CMO for action. Rejected items are sent to the originator along with the CCB's rationale for rejection. CRs needing further analysis are sent back to the analysis group with the CCB's questions attached. Deferred CRs are filed, to be sent back to the board at the proper time.

The CMO sends approved items to the development organization, unless it is of a level that needs to be processed through higher level CCBs. If additional levels of approval are needed, the CMO submits the CR package to the next level CCB.

The CMO, acting as the secretariat of the CCB, prepares and distributes the meeting minutes, and records the current status of the CR. This information is added to the tracking data base or recorded in files.

5. Change Implementation

Approved CRs are either directly used as a change authorization form or result in a change directive being prepared by the CMO. In either case, approval results in the issuance of instructions which authorize and direct the implementation of the change in the software and associated documentation.

The development organization schedules the resources to make the change. It must get official copies of the baselined component to be changed from the program library. For code changes, design has to be developed, code has to be written, and testing has to be done and the correctness of the change verified. Moreover, the associated documentation has to be revised to reflect the change. Once the change has been made and local testing completed, the revised component and documents are returned to the control of the program library. Upon verification, the new version takes its place in the sequence of baselines.

6. Change Verification

The implemented changes, which have been tested at the unit level, must be verified at the CSCI level. This may require the rerun of tests specified in the test plan or the development of an addition to the test plan. Regression testing will usually have to be included in the test to assure that errors have not been introduced in existing functions by the change. Once the verification is complete, the development organization submits evidence of it to the program library, which will then accept the changed items for inclusion in the SCM controlled files that make up the new version of the baseline. This CSCI or system level of verification may not occur until the next

release of the system is being tested.

After the successful implementation and testing of the change described in the CR, the CMO will record the occurrence of this process into the change request tracking data base or files.

7. Baseline Change Control

Changes to software are not complete until the code and data changes have been implemented and tested and the changes to associated documentation have been made and all of the changes verified. In order to minimize the number of versions and the frequency of delivery of software components, changes to software are usually grouped into releases. Each release contains software and documentation that has been tested and controlled as a total software system.

D. Configuration Status Accounting

The objective of software configuration status accounting is to record and report the status of the evolving software throughout the life cycle. It provides traceability of changes to the baselined requirements, code and data components, and associated documentation. It documents what is in each version of software and the changes that lead up to the version. It accounts for all changes to the baselined items whether they are incorporated in accordance with the change control process or in accordance with the nonconformance reporting and corrective action process. This function keeps track of the changes and the contents of versions and releases.

Status accounting begins when the first specification (i.e., software requirements specification) is baselined and continues throughout the life cycle. The accounting information is a key element used during the functional configuration audits and physical configuration audits that are done in the authentication process. Status accounting provides a list of the contents of each delivery of the software and associated documents.

Software configuration status accounting is a record keeping and reporting activity. The records contain the identification of the initial software and associated documents and their current status, status of evolving baselines, status of proposed and approved changes, and the implementation status of approved changes. Reports document the information contained in the records and are the means for disseminating the information. Examples of routine reports furnished by the status accounting function are: status of change requests (CRs), specification change notices (SCNs), and version description documents (VDDs).

E. Configuration Authentication

Configuration authentication is the process of assuring that the baselined configuration has been tested to demonstrate that it meets its functional requirements and that it contains all deliverable entities. As the principal means of configuration authentication, audits are scheduled before each delivery of the software system. Configuration audits ensure that the CSCIs conform to the specifications that are part of the baseline. A functional configuration audit (FCA) authenticates that the software performs in accordance with the requirements and as stated in the baselined documentation. A physical configuration audit (PCA) authenticates that the components to be delivered actually exist and that they contain all of the required items, such as the proper versions of source and object code, documentation, installation instructions, etc.

1. Functional Configuration Audit

The functional configuration audit authenticates that the actual performance of the CSCI complies with the requirements stated in the baselined design documentation. This is accomplished by evaluating the test methods, procedures, reports, and other engineering and design documentation. It may not be possible to completely authenticate a CSCI until higher level integration is accomplished. Audit reports are prepared to document the results of the FCA. These audit reports are generally used as part of the documentation provided to the acquirer of the software as part of a phase ending review, such as the Acceptance Test Readiness Review.

2. Physical Configuration Audit

The physical configuration audit is the formal examination of the as-built version of the component against the baselined technical documentation defining the component. The PCA assures that changes to be included in the version of software to be delivered are really included, that all required items of software, data, procedures, and documentation are included. Audit reports are prepared to document the results of the PCA and are used in the same reviews as the reports of the FCA.

3. External Audits of the SCM Process

Other types of audits of SCM are also conducted during the development process. These audits are done by the Quality Assurance (QA) organization to assure that the SCM process is being conducted according to approved procedures. The role of the CMO and the program librarian during these audits is to make their records available to QA and to respond to any problems noted in the audit reports. These types of QA audits are described in the Guidebook "Software Quality Assurance Audits", SMAP-GB-A301.

IV. SCM DURING THE SOFTWARE LIFE CYCLE

There are phase-specific SCM activities that should be conducted during the software acquisition life cycle and specific baselines that are established at the end of each phase. In a complete waterfall model life cycle, the recommended baselines are (see Figure 2):

- Software Requirements Baseline
- Software Allocated Baseline
- Software Design Baseline
- Software Code Baseline
- Software Product Baseline
- Software Accepted (As-Built) Baseline

Figure 2 is an adaptation of the NASA Software Acquisition Life Cycle chart, version 4.0, dated 1988. The figure focuses on the relationships between the baselines and life-cycle phases. If the life cycle to be used is modified by either the acquirer or the provider, some of the baselines may be eliminated. In other situations, such as where development by builds is done, some baselines may be struck repeatedly as parts of the life cycle are repeated. The important point is that baselines need to be established and changes thereto be documented and authorized.

In the remainder of this section the development activities of each phase of the life cycle are briefly described, along with the SCM activities during the phase and the contents of the baseline that is established at the end of the phase.

A. Software Concept and Initiation Phase

During the software concept and initiation phase, the software concept is developed and the feasibility of building the software system is evaluated. The acquirer's software management plan is then written and approved and an acquisition strategy is developed. If a contract is to be used to acquire the software, procurement is initiated and a contract is awarded.

The acquirer CMO is responsible for developing the SCM portion of the project software management plan, and is responsible for developing a compatible set of provider SCM requirements that are included in the Request for Proposal (RFP).

During the proposal reviews, the CMO provides expertise in the evaluation of the proposed SCM activities by each potential provider. The provider's SCM activities may be presented in the SCM portion of their proposal or as an initial draft of the SCM plan or section of a software management plan provided as part of the proposal. The acquirer CMO assesses whether or not the potential provider has proposed an adequate plan that meets all of the SCM requirements in the RFP.

During the Concept and Initiation Phase, the acquirer should place under SCM the project software management plan, procedures developed to carry out sections of the plan, and the system level requirements passed to the provider.

B. Software Requirements Phase

During the software requirements phase, the software concept and allocated system requirements are analyzed and documented as software requirements. Test planning begins, with a method for verifying each requirement identified and included in a preliminary test plan. Risks are identified and risk management control mechanisms are established. The size and scope of the remainder of the project is reevaluated, and changes in resources and schedules are made. Methods, standards, and procedures are detailed and put in place.

During this phase, the provider CMO should complete the final SCM Plan and submit it to the acquirer for review. The acquirer CMO will evaluate the provider's SCM Plan to ascertain that all of the SCM requirements are satisfied, and that the plan is complete and the procedures to be used are clearly stated. As part of the SCM planning, the staff required to perform SCM will have been determined and the assignment of the SCM staff will begin in this phase. Upon agreement between the acquirer and the provider, the Provider SCM Plan is placed under

provider configuration management.

The software requirements baseline is struck after the completion of this phase and the satisfactory resolution of issues raised at the phase ending Software Requirements Review (SRR). The major component of the requirements baseline is the approved software requirements specification and interface requirements documents. However, it should also contain other relevant provider management documentation such as development plans, assurance and SCM plans, test plans, etc. These management and development documents detail the approach to managing, developing, testing, assuring, and controlling the software. They include or refer to applicable standards and procedures that will be adhered to during the development of the software.

The contents of the software requirements baseline become a permanent part of all succeeding baselines and are the basis against which the remaining development effort is authenticated. Any proposed change to this baseline will follow the change control process described in the Software Configuration Control within the SCM Process, Section III.

C. Software Architectural Design Phase

The objective of the software architectural design phase is to develop an overall design for the software, allocating all of the requirements to software components. The software requirements are controlled and managed, and the contents of the requirements baseline are changed only by a formal process. The phase ends with the preliminary design review, during which the acquirer and provider agree on the architecture of the system that is to be produced. Rework and action items resulting from the review are tracked and completed.

The software allocated baseline contains the architectural design of the system and documents showing how the requirements are allocated to the design. This baseline is struck after the completion of this phase and the resolution of any problems raised at the Software Preliminary (Architectural) Design Review (PDR). The baseline contains all the updated documents from the Requirements baseline, along with the architectural design specification. The baseline may also contain a software build (or release) plan and test plans. If present, these plans are usually still at a high level, with general functions assigned to the proposed builds or releases.

The contents of the software allocated baseline become a permanent part of all later baselines and a part of the basis against which the remaining development effort is authenticated. Any proposed change to this baseline will follow the change control process described in the Software Configuration Control within the SSCM Process, Section III.

D. Software Detailed Design Phase

During the software detailed design phase, the architectural design is expanded to the unit level. Interface control documents are completed and test plans revised. Constraints and object system resource limits are re-estimated and analyzed, and staffing and test resources are validated. The phase ends with the Critical Design Review. During this phase, both the requirements and the architectural design are under configuration management control of the provider CMO as part of the allocated baseline, and can only be changed by a formal process.

The software design baseline is struck after the completion of this phase and the resolution of problems raised at the phase ending Software Critical Design Review (CDR). The software design baseline contains the detailed (code to) design for the software. The major new addition in this baseline is the software detailed design specification. It details the design of the CSCIs that will provide all the capabilities and meet the deign constraints specified in the software allocated baseline. Software specifications include designs at a level and in a form such that unit design, coding, and testing can be performed. This specification identifies the modules that make up the CSCIs, the architecture of each module to the unit level, the module and unit interfaces, the data files to be used during the execution of the software, and the user interface. The updated contents of the allocated baseline are part of this baseline, in addition to more complete test and build plans.

The content of the software detailed design specification becomes a permanent part of following baselines. Many of the components of this baseline will be deliverable products. Any proposed change to this baseline will follow the change control process described in the Software Configuration Control within the SCM Process, Section III.

E. Software Implementation Phase

During the software implementation phase, the software is coded and unit tested. All documentation is produced in quasi-final form, including internal code documentation. At the end of the phase, all required products should be ready for delivery, subject to modification during integration and testing.

After the software components (units) have been coded and successfully passed unit test, they are transferred from the developers control to the provider CMO control and placed under configuration management in a program library.

At the end of this phase, the Code Baseline is struck. This is the first time that the code itself becomes part of a baseline. This baseline is normally an internal baseline, without delivery and review of products by the acquirer. This baseline is not created at a single event, but rather the code baselining occurs repeatedly throughout the coding process as each unit of code is inspected and unit tested. As each unit successfully passes its unit tests, it is placed under configuration control. When implementation of all units is complete, the baselined unit code is the basis for CSCI and system integration testing in the next phase.

F. Software Integration and Test Phase

The objectives of the software integration and test phase are to integrate the software units into a completed system, discover and correct any nonconformances, and prepare for the formal acceptance of the system. The phase ending review is the test readiness review, during which the developer provides to the acquirer evidence that the software system is ready for acceptance testing. During this phase, the test plan is executed, the documentation is updated and completed, and the products are finalized for delivery. The provider CMO will begin to prepare the Version Description Document (VDD).

The provider's testing organization uses the code baseline, which should include baselined test plans, to test and integrate the CSCIs and then to integrate them into a deliverable system.

After the controlled software components have been integrated and tested, the integrated software is placed under configuration management control in the program library. Once under control, the tested software can only be changed by an approved CR or as the result of a nonconformance (discrepancy) report that requires corrective action.

After the system testing has been completed and put under formal control, an FCA is performed to authenticate that the actual performance of the CSCIs complies with the requirements stated in the baselined software requirements document. This is accomplished by evaluating the test methods, procedures, reports, and other engineering and design documentation.

After the FCA has been successfully completed, a PCA is conducted to examine the as-built CSCIs against required deliverables, usually as defined in a contract deliverables requirements list (CDRL). The provider performs the PCA to ensure that all deliverable items are present and complete, and the system is ready to be turned over for acceptance testing by the acquirer or designated representative.

The phase ends with the Test Readiness Review (TRR). After resolution of any problems found during the TRR, the software product (or integrated) baseline is struck. This baseline contains the deliverable software and documents, updated to show as-built design. Along with the software, all other deliverable items, such as populated data bases and tables, computer installation procedures, and test beds are part of this baseline. The software is ready for system level integration testing and acceptance testing.

G. Software Acceptance and Delivery Phase

During the software acceptance and delivery phase, the formal acceptance procedures are carried out. As a minimum, there is a requirements-driven demonstration of the software to show that it meets those requirements. The process also may include acquirer tests, field usage, or other arrangements that are intended to assure that the software will function correctly in its intended environment. At the end of the phase, an FCA and PCA are completed and a software acceptance review is conducted. Their successful completion results in the establishment of the accepted or as-built baseline. Now the software is ready to be used for the purpose for which it was acquired.

During this phase, the software is still under configuration management control of the provider. The software and all documents that have been placed under configuration management can only be changed by the change request process and an approved CR.

At the successful completion of the acceptance process, the software and all associated data and documentation are turned over to the acquirer and are placed under acquirer's CM control.

The software that is delivered is known as the accepted or as-built baseline.

H. Software Sustaining Engineering and Operations Phase

During this phase of the software life cycle, the software is used to achieve the intended objectives for which it was acquired. Corrections and modifications are made to the software to sustain its operational capabilities and to upgrade its capacity to support its users. Software changes may range in scope from simple corrective action up to major modifications that require

a full life cycle process.

During this phase, the baselined operational software and all baselined documents are under strict configuration management control of the acquirer CMO. No baselined software and applicable documents can be changed without following the change request process, including CCB approval.

In the event that a major modification using the full life cycle process is necessary to implement an approved change, the same configuration management control processes as are applicable to a new development are applied by the acquirer and provider CMOs throughout the life cycle of the modification.

V. SCM IN OTHER LIFE CYCLES

In the previous parts of this guidebook, we have defined Software Configuration Management, defined the processes that make up SCM, and defined the baselines that are produced in the standard waterfall life cycle. In the real world, few software systems are produced by going once through the waterfall. Most systems are large enough that they have to be done piece by piece, with each piece being integrated and tested and then used as a basis for the next development phase. This is called phased development. There are a number of phased development variations on the basic waterfall life cycle. For example, development by builds usually involves completing the requirements analysis and the architectural design phases of the waterfall and the completing the PDR. The software is then divided into sections that contain groups of functions or sections of the architecture (usually the former). The detailed design, implementation, and integration and test of each functional section is then built sequentially. Development by builds allows each tested build to be used as a basis and test bed for the next set of builds. Development by builds is probably the most frequently used development life cycle. Please note that builds may be delivered to the acquirer for acceptance and use. If so, the process is termed "phased delivery", but differs in no fundamental way from the process where the builds are retained by the provider until the system is complete.

If less is known about the real world requirements for the delivered system, other phased development life cycles may be used. For example, an evolutionary model can be used when only a portion of the requirements can be defined well enough to justify developing software to meet them. In this case, it is known that other requirements exist, and work in gathering and understanding them may be ongoing, for example by development of prototypes. The inability to define the total set of requirements means that the total development life cycle must be repeated several times, with each pass through involving an additional set of requirements that have been defined sufficiently well that development can take place. In each requirements analysis phase and review, the new and changed requirements must be addressed and baselined. The subsequent phases of the development process must accommodate both the development of new designs and code and the modification of (potentially) significant parts of the existing code. The spiral model is an example of an evolutionary development process.

Once a software system enters operation, the SCM problem becomes much like the one described above. That is, there will be in existence a current version of the software that is being used for operations, previous versions that were used to produce products that may have been

distributed, and versions under development that fix problems and/or incorporate new capabilities.

From the SCM point of view, the use of these multiple pass development life cycles does not change the fundamental SCM processes as discussed in this guidebook. The four functions of configuration identification, configuration control, configuration status accounting, and configuration authentication still must be done. However, the actual application of SCM becomes both more important and more difficult. Many versions of design and code, for example, may exist at the same time and be used in different builds or versions of the software system. Each version of the system and the changes incorporated in it will have to be controlled and its configuration clearly known.

At each delivery of the system it is especially important to do a PCA, to ensure that the contents of the delivered code, data, and documentation is the correct set that includes all of the items that are intended. It is likely that revision of the contents of a version of the software system will be made late in the development cycle, as lower priority functions (or those that have been difficult to add) are postponed to later releases or builds, and changes or fixes that have risen in priority are rushed into the release.

SCM on a large project with multiple releases does not change in nature, but becomes more difficult to do correctly and consumes more resources. The rush to deliver a version must not cause Configuration Management Control to be lost, or the confusion that will result will cost a lot more to rectify that it would have cost to do it right.

VI. SCM ORGANIZATION AND RELATIONSHIPS

A. SCM Structure

A software project's configuration management structure has three major components.

First, there is a Configuration Management Officer (CMO), who is responsible for the operation of the configuration management process and the maintenance of configuration control over the evolving products. Second, there is a program library and librarian, who has control and custody of all of the software products, both electronic and hard copy. Third, there is a Configuration Control Board (CCB), which decides which suggested changes will actually be made.

The roles and responsibilities of each of these components of the CM system are discussed below.

B. The Role and Responsibilities of the CMO

The Configuration Management Officer is the key individual in a project's SCM system. The CMO's role begins with the development of a SCM plan for the project and the establishment of a procedure that details each step in the SCM process. The CMO has to set up the program library in a manner compatible with the project's size and resources and establish the project's

Change Request (CR) tracking data base. After these procedures and resources are in place, the CMO manages and operates the CM system.

Each CR initiated is sent to the CMO as the first step in the SCM process. The CMO receives the CR and reviews it for clarity and completeness. If the CMO determines that the CR is not complete, it is returned to the originator. The CMO reviews the suggested class of the change and assigns a working classification. The CMO then assigns the CR a unique identifier for tracking purposes and records information about the CR into the change request data base.

At this point, an official, complete CR exists and begins its process through the SCM system according to the procedures established by the CMO. After each step in the process, the CR is returned to the CMO, who ensures that the actions have been completed and records its new status in the data base. For example, the CR has to be routed to the proper individuals for assessment of its impact. After the assessment is complete, the CR is routed back to the CMO with the assessment report attached. The CMO will then change the status in the data base to assessment complete, and add the CR to the agenda for a CCB meeting.

The CMO is the secretary for the project's CCB, and is responsible for preparing and distributing its agendas and minutes and recording status of CRs that have been dispositioned by the CCB. The dispositioned items are acted upon by the CMO. The CMO sends rejected items back to the originator along with the CCB rationale for rejection. If the CCB feels that a CR needs further analysis, then the CMO sends it back to the analysts with the CCB's questions attached. If the CCB deferred a CR, it must be filed, to be sent back to the board at the proper time. If an approved CR needs to be processed through higher level CCBs, the CMO is responsible for submitting the CR package to the next level CCB. Each of these actions requires the CMO to update the tracking database.

The CMO sends approved CRs to the development organization for action. After the change has been implemented (and tested, if the change was to code), the CMO updates the tracking data base to show the status of the CR as closed.

The CMO is responsible for producing and distributing periodic CR data base and individual product CR status reports. These reports keep everyone concerned informed as to the status of the proposed changes. The CMO is also responsible for management of the SCM library and for conducting functional and physical configuration audits.

C. The Role of the Program Librarian

The program librarian operates the program library to manage the baseline software, data, and documents. The librarian accepts documents, code, data files, and other components of baselines and puts them in secure storage. The librarian issues working copies to developers for authorized changes, and keeps records and historical copies of all versions of the components of baselines. He or she makes copies of baselined software for testing and distribution, and prepares version description documents.

The role of the program library and librarian includes the storage and control of both software and associated documentation. The program library must control hard copy documents, computer files, and the physical media on which the latter is stored. It will have to have a procedure for

archiving old versions of the system, of controlling the current version, and of accepting from developers potential new versions which have to be verified. In addition, the program librarian often supports the CMO in the maintenance of records and the production of reports.

D. Role of the Configuration Control Board

The CCB provides a forum for the review and disposition of the proposed changes to baselined requirements, documentation, and software. The CCB is responsible for discussion of proposed changes and for voting or otherwise recommending to the CCB chair the disposition of those changes.

The CCB is a working group consisting of representatives from the various disciplines and organizations of the developing project. The exact number, skills, and level of management of the CCB participants will vary, depending upon the change request to be reviewed. The project manager or the senior manager of the organization is usually the CCB chairperson. The CMO prepares a review package for each CR, containing the change proposal, relevant documents, and the analysis by the developers, and sends it to the CCB members. The CMO, who is normally the CCB Secretary, prepares the meeting agendas and records the meeting minutes.

At a CCB meeting, each CR on the agenda is covered in turn. Each member discusses the pros and cons of accepting the CR from her/his point of view and within his/her area of expertise. However, the CCB chairperson is responsible for making the final decision. CCB members do not have "voting rights" and CCB decisions are not made by majority rule. CCB decisions are management decisions that include the expenditure of resources, and the decisions must be made by the responsible manager. However, the CCB chairperson should carefully weigh the advice of each member before making the decision.

E. SCM Relationships

The SCM structure has to interface with all of the other entities that make up the development project and with potential users of the system under development.

The SCM organization interfaces with project management to provide information and reports on the number and status of changes proposed to the software. The CMO will provide both routine reports and answer special requests for data. The CMO acts as the secretary for the CCB, which is made up of senior project managers. Most importantly, the CMO and the program librarian develop the Version Description Document (VDD) and the deliverable system, and certify to management, via the results of FCAs and PCAs, that the system is ready for delivery to the customer.

The SCM organization interfaces with the development organization both to accept CSCI components to become part of a new baseline or version and to provide to the developers authorized copies of existing products for updating in accordance with approved changes.

The testing organization is dependent on the SCM organization to provide copies of the proper version of all code and documents for the development of test plans and procedures and to use in executing tests.

The SCM organization interacts with QA both as an entity that is audited by QA, and as a participant with QA in the conduct of FCAs and PCAs. In many organizations, QA must be the final signoff before a product can be accepted from the development organization and included in the program library.

VII. GUIDANCE IN ADAPTING SCM TO A PROJECT

Plans for software configuration management and the structure and size of the organization needed to implement it should be influenced by the complexity of the software to be developed, the size of the project, and on the specific responsibilities assigned to the SCM organization. Resource needs will be influenced by the tools and computer resources available.

A. Project Size

A small project with a reasonable tool environment may well be able to assign all of the duties of the CMO and the program library to one individual. It may be able to use less formal procedures than a large project. The program library may be a series of files - a set of individual developer controlled files for writing and documenting software, a set of files that contains developed code that has been unit tested but is not yet integrated or baselined, the current baseline, and archive versions. The last three sets should not be able to be written to by anyone other than the program librarian.

Larger projects must have more formal procedures, especially for the CCB. There will be many changes suggested, and the burden of tracking all of them can be quite heavy. Large projects may set up a screening board, which recommends to the CCB the disposition of certain types of proposed changes. For example, a screening board might be set up to handle all user interface changes. For the type of changes delegated to the screening board, the procedure could be that the CCB will automatically accept the disposition of the screening board unless some CCB member wants a full discussion.

Another approach used by large projects is to set up sub-boards for a subset of the total project, for example, for one or more CSCIs. The CSCI board may have authority to accept changes, within certain resource limits, that do not affect the interfaces between CSCIs controlled by other sub-boards. The full CCB should review the dispositions of the sub-boards only if a CCB member feels the review is warranted.

B. Informal Controls

As the CM system is set up and operated, care needs to be taken not to impede development and testing while changes to the software system are being proposed, processed, and implemented. It is especially important that managers have an informal adaptation of the CM process to control changes to interim products while they are in development and use, but before they are part of a formal baseline. During this period, software developers can use the same types of baseline management to manage the new or changed products. For example, as the design at each level is documented, each informally proposed expansion, enhancement, or other change should be

examined for its impact. Working documents are modified to reflect all approved changes and the current status of approved design is made available to all participants. Informal records are kept to provide an audit trail as the design evolves. This sequence is likely to occur on an active and continuing basis as the design of the software is developed incrementally through more detailed levels. No formal paperwork is used, but adequate management control is exerted.

As another example, as units of code are unit tested, the development manager may need to control the changes made to those units that have completed unit testing so that related modules can use the tested modules in their own testing. No formal paperwork is needed at this stage, but lack of control will yield confusion and duplication of effort. Many managers put in place a multi-file program library system like that described above, with units progressing through the sets of files as testing progresses.

Use of the term "configuration management" to describe techniques for this type of informal control during development is common. This is often a source of confusion to software managers involved in a system development effort, since they may not distinguish between their need to control the changes made to products they are developing and the formal SCM that responds only to the official CCB. However, this type of informal control cannot substitute for the formal process needed to manage approved or completed products. It is a recommended augmentation to the more formal process.

C. SCM Resources

The level of SCM resource support required by the development project will vary dependent upon the life cycle phase. The Software Concept and Initiation Phase is usually the function of the acquirer, and begins with minimal SCM resources whose role is to develop an acquirer SCM plan and to develop provider SCM requirements. After selection of a provider, the provider's resource needs will increase in successive life cycle phases as the requirement for SCM activities increases with the greatest needs occurring during system integration and test. The acquirer's needs will also grow in order to process level one changes and to review level two changes, but will be much less than that of the provider. The acquirer's greatest SCM resource needs will occur after acceptance of the software system.

The maximum need for provider and acquirer SCM resources will occur in a project that is using a phased delivery process (See section V). During delivery, there will be some versions of the software delivered and being maintained, some being tested, and others in development. These resource needs are critical - lack of adequate resources will result in an unacceptable situation. Either the project will relax its rules and allow changes to be made with no change control, or progress will be slowed by an inability to process changes in a timely manner. Adequacy of the SCM resources commensurate with the size and phasing of the development process is essential to the development of quality software and its delivery.

VIII. TOOLS AND TECHNIQUES

Many of the processes in SCM are labor intensive and involve considerable paperwork. Tools can be used to help alleviate some of the paperwork and to reduce the cost of SCM. Many tools are available from commercial and shareware developers, and some tools can be easily

developed using software normally available on personal computers or as part of the development environment.

A set of tools that is simple to implement are those the program library uses as part of the configuration management function to stage the sets of products. Baselined products can be kept in a library (set of disk files) that only the librarian can write into. This allows the developers and testers to get a copy of baselined documents, data, and code for modification and testing, but does not allow them to change the baselined products. Often the library will be set up with several stages of each product under control. There will be the current delivered version, the version under test for the next delivery, a development version that includes tested units of the version two steps away from delivery, and developer libraries of components not yet unit tested. The version staging diagram, Figure 3, shows these libraries and the flow among them.

Another set of tools that are readily available or easily developed support the configuration status accounting function. A simple data base will allow the entry of change requests (or at least change request summary data), and the updating of the status of the change request as it flows through the system. Informational reports can be generated from the data base to allow submittors and project staff to know the status of a report at any time. The data base can be used to record the version of the system to which a change is assigned and this information will support the PCA/FCA activity.

High end tools are available that will record each version of each product, using a transaction based scheme to note exactly what part of each product was changed by a CR. These tools record the reason for each change, and may automatically generate CM reports and other such features.

It is important to provide a set of tools to support the SCM function on a project. Because SCM must keep track of very detailed information, it is easy to make errors that can cause the project rework and time lost due to confusion. The tool set chosen should be appropriate to the size and structure of the development project. The same CM tools that are used by the provider should be considered for use by the acquirer for the purpose of continuity and consistency throughout the sustaining engineering and operations phase.

APPENDIX A
SUMMARY OF BASELINE CONTENTS

This appendix briefly lists the contents of each baseline established during the standard waterfall life cycle. It summarizes information found in Section IV. In using this section it must be remembered that the development of baselines is cumulative, that is, all of the contents of each baseline are part of the next baseline, with updates as needed. Generally this appendix only lists the additions to the baseline at the end of each phase.

Initial Acquirer Baseline

At the end of the software Concept and Initiation Phase, the Acquirer should establish an Initial Acquirer's Baseline that contains:

- The project Software Management Plan (SMP)

- Procedures developed to carry out the SMP
- System level requirements passed to the provider

After a software provider is selected, selected provider plans may be added to this baseline (provider SCM plan, for example). Alternately, these plans may be under provider control.

Software Requirements Baseline

This baseline is established after the completion of the requirements analysis phase and the satisfactory resolution of issues raised at the phase ending Software Requirements Review (SRR). It should contain:

- The software requirements specification
- Interface requirements documents

In addition the following should be baselined at this time if not done at the time of agreement between the acquirer and provider:

- Software development plan
- Software assurance plan
- Software SCM plan

Software Allocated Baseline

This baseline is struck after the completion of this phase and the resolution of any problems raised at the Software Preliminary (Architectural) Design Review (PDR). The baseline contains all the updated documents from the Requirements baseline, along with the following:

- The architectural design specification
- Documents showing how the requirements are allocated to the design
- Also baselined at this time should be:
    - Software build (or release) plan
    - Software test plan (high level)

Software Design Baseline

This baseline is established at the completion of the CDR. It must contain all updated documents from the previous baselines and the software detailed design specification. In addition, the build and test plans begun during the requirements phase and included (at a high level) in the previous baseline should be completed and baselined at this time.

Software Code Baseline

This baseline is established at the end of the software implementation phase. It should include, in addition to the updated contents of the previous baseline, the following:

- The code itself
- Code level documentation
- Users Manuals
- Test Procedures for the I&T Phase
- Data needed for operation of the software

Software Product Baseline

This baseline is established at the completion of the Integration and Test Phase. The software is to be ready for acquirer acceptance testing and delivery. It should include, in addition to the updated contents of the previous baseline, the following:

- The tested code
- Final versions of all products and documents

Software Accepted (As-Built) Baseline

This baseline is established after the software has been accepted by the acquirer. It should contain updated versions of the items in the product baseline, with corrections for nonconformances found during the acceptance process.APPENDIX B
EXAMPLE CHANGE REQUEST FORM

APPENDIX C
ACRONYM AND ABBREVIATIONS

CCB Configuration Control Board

CDR Critical Design Review

CDRL Contract Documentation Requirements List

CI Configuration Item

CM Configuration Management

CMO Configuration Management Officer

CR Change Request

CSCI Computer Software Configuration item

DID Data Item Description

FCA Functional Configuration Audit

NASA National Aeronautics and Space Administration

PCA Physical Configuration Audit

PDR Preliminary Design Review

QA Quality Assurance

RFP Request For Proposal

SCM Software Configuration Management

SCN Software Change Notice

SRR Software Requirements Review

TRR Test Readiness Review

VDD Version Description Document

Last revised: July 16, 1997