CHAPTER 7

SOFTWARE CONFIGURATION MANAGEMENT

John A. Scott and David Nisse
Lawrence Livermore National Laboratory
7000 East Avenue
P.O. Box 808, L-632
Livermore, CA 94550, USA
(925) 423-7655
scott7@llnl.gov

Table of Contents

1 INTRODUCTION

This paper presents an overview of the knowledge area of software configuration management (SCM) for the Guide to the Software Engineering Body of Knowledge (SWEBOK) project. A breakdown of topics is presented for the knowledge area along with a succinct description of each topic. References are given to materials that provide more in-depth coverage of the key areas of software configuration management. Important knowledge areas of related disciplines are also identified.

Keywords

Software configuration management, software configuration identification, software configuration control, software configuration status accounting, software configuration auditing, software release management.

Acronyms

CCB Configuration Control Board
CM Configuration Management
DBMS Database Management System
FCA Functional Configuration Audit
PCA Physical Configuration Audit
SCI Software Configuration Item
SCR Software Change Request
SCM Software Configuration Management
SCMP Software Configuration Management Plan
SCSA Software Configuration Status Accounting
SDD Software Design Description
SQA Software Quality Assurance
SRS Software Requirements Specification

2 DEFINITION OF THE SCM KNOWLEDGE AREA

A system can be defined as a collection of components organized to accomplish a specific function or set of functions [IEEE 610]. The configuration of a system is the function and/or physical characteristics of hardware, firmware, software or a combination thereof as set forth in technical documentation and achieved in a product [Buckley]. It can also be thought of as a collection of specific versions of hardware, firmware, or software items combined according to specific build procedures to accomplish a particular purpose. Configuration management (CM), then, is the discipline of identifying the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle [Bersoff, (3)]. CM is formally defined [IEEE 610] as:

> "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."

The concepts of configuration management apply to all items to be controlled although there are some differences in implementation between hardware CM and software CM.

This chapter presents a breakdown of the key software configuration management (SCM) concepts along with a succinct description of each concept. The concepts are generally accepted in that they cover the areas typically addressed in texts and standards. The descriptions cover the primary activities of SCM and are only intended to be sufficient for allowing the reader to select appropriate reference material according to the reader's needs. The SCM activities are: the management of the software configuration management process, software configuration identification, software configuration control, software configuration status accounting, software configuration auditing, and software release management and delivery.

Figure 1 shows a stylized representation of these activities

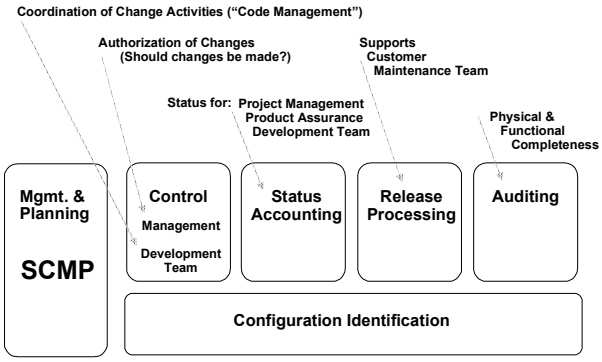

Figure 1. SCM Activities

Following the breakdown of SCM topics, key references for SCM are listed along with a cross-reference of topics that each listed reference covers. Finally, topics in related disciplines that are important to SCM are identified.

3 BREAKDOWN OF TOPICS FOR SCM

Breakdown of Topics

An outline of the breakdown of topics is shown below in Figure 2. Following the chart, a brief description of each breakdown topic is provided. The breakdown covers the concepts and activities of SCM. The variety of SCM tools and tool systems now available, as well as the variety of characteristics of the projects to which they are applied, may make the implementation of these concepts and the nature of the activities appear quite different from project to project. However, the underlying concepts and types of activities still apply.

I. Management of the SCM Process

Software configuration management is a supporting software life cycle process [ISO/IEC 12207] that benefits project and line management, development and maintenance activities, assurance activities, and the customers and users of the end product. From a management perspective, SCM controls the evolution and integrity of a product by identifying its elements, managing and controlling change, and verifying, recording and reporting on configuration information. From the developer's perspective, SCM facilitates the development and change implementation activities. A successful SCM implementation requires careful planning and management. This, in turn, requires an understanding of the organizational context for, and the constraints placed upon, the design and implementation of the SCM process.

I.A Organizational Context for SCM

To plan an SCM process for a project, it is necessary to understand the organizational structure and the relationships among organizational elements. SCM interacts with several other activities or organizational elements.

SCM, like other processes such as software quality assurance and software verification and validation (V&V), is categorized as a supporting life cycle process. The organizational elements responsible for these processes may be structured in various ways. Although the responsibility for performing certain SCM tasks might be assigned to other organizations, such as the development organization, the overall responsibility for SCM typically rests with a distinct organizational element or designated individual.

Software is frequently developed as part of a larger system containing hardware and firmware elements. In this case, SCM activities take place in parallel with hardware and firmware CM activities and must be consistent with system level CM. Buckley [5] describes SCM within this context. Note that firmware contains hardware and software and, therefore, both hardware and software CM concepts are applicable.

SCM is closely related to the software quality assurance (SQA) activity. The goals of SQA can be characterized [Humphrey] as monitoring the software and its development process, ensuring compliance with standards and procedures, and ensuring that product, process, and standards defects are visible to management. SCM activities help in accomplishing these SQA goals. In some project contexts, e.g. see [IEEE 730], specific SQA requirements prescribe certain SCM activities.

SCM might also interface with an organization's quality assurance activity on issues such as records management and non-conforming items. Regarding the former, some items under SCM control might also be project records subject to provisions of the organization's quality assurance program. Managing non-conforming items is usually the responsibility of the quality assurance activity, however, SCM might assist with tracking and reporting on software items that fall in this category.

Perhaps the closest relationship is with the software development and maintenance organizations. The environment for software engineering includes such things as the:

- software life cycle model and its resulting plans and schedules,

- project strategies such as concurrent or distributed development activities,
- software reuse processes,
- development and target platforms, and
- software development tools.

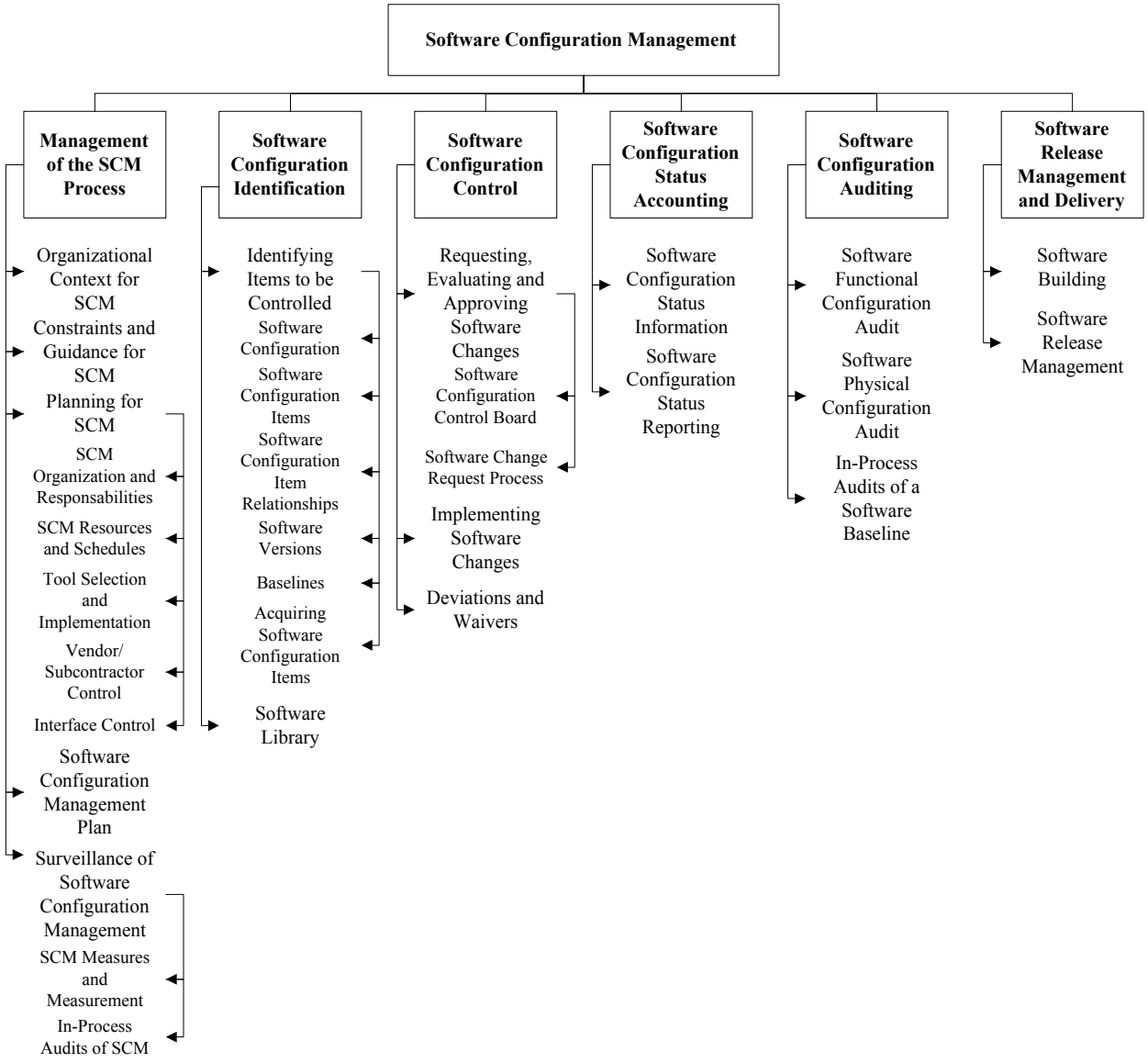

Figure 2 Breakdown of SCM Topics

This environment is also the environment within which many of the software configuration control tasks are conducted. Frequently, the same tools support development, maintenance and SCM purposes.

I.B Constraints and Guidance for SCM

Constraints affecting, and guidance for, the SCM process come from a number of sources. Policies and procedures set forth at corporate or other organizational levels might influence or prescribe the design and implementation of the SCM process for a given project. In addition, the contract between the acquirer and the supplier might contain provisions affecting the SCM process. For example, certain configuration audits might be required or it might be specified that certain items be placed under configuration management. When software products to be developed have the potential to affect the public safety, external regulatory bodies may impose constraints. For example, see [USNRC]. Finally, the particular software life cycle model chosen for a software project and the tools selected to implement the software affect the design and implementation of the SCM process [Bersoff, (4)].

Guidance for designing and implementing an SCM process can also be obtained from 'best practice' as reflected in the

standards on software engineering issued by the various standards organizations. Moore [31] provides a roadmap to these organizations and their standards. Best practice is also reflected in process improvement and process assessment models such as the Software Engineering Institute's Capability Maturity Model (SEI/CMM) [Paulk] and the International Organization for Standardization's Software Process Improvement and Capability determination project (ISO SPICE) [El Emam].

I.C Planning for SCM

The planning of an SCM process for a given project should be consistent with the organizational context, applicable constraints, commonly accepted guidance, and the nature of the project (e.g., size and criticality). The major activities covered are Software Configuration Identification, Software Configuration Control, Software Configuration Status Accounting, Software Configuration Auditing, and Software Release Management and Delivery. In addition, issues such as organization and responsibilities, resources and schedules, tool selection and implementation, vendor and subcontractor control, and interface control are typically considered. The results of the planning activity are recorded in a Software Configuration Management Plan (SCMP). The SCMP is typically subject to SQA review and audit.

I.C.1 SCM Organization and Responsibilities

To prevent confusion about who will perform given SCM activities or tasks, organizations to be involved in the SCM process need to be clearly identified. Specific responsibilities for given SCM activities or tasks also need to be assigned to organizational entities, either by title or organizational element. The overall authority and reporting channels for SCM should also be identified, although this might be accomplished in the project management or quality assurance planning.

I.C.2 SCM Resources and Schedules

The planning for SCM identifies the staff and tools involved in carrying out SCM activities and tasks. It addresses schedule questions by establishing necessary sequences of SCM tasks and identifying their relationships to the project schedules and milestones established in the project management planning. Any training requirements necessary for implementing the plans and training new staff members are also specified.

I.C.3 Tool Selection and Implementation

Different types of tool capabilities, and procedures for their use, support the SCM activities. Depending on the situation, these tool capabilities can be made available with some combination of manual tools, automated tools providing a single SCM capability, automated tools integrating a range of SCM (and, perhaps other) capabilities, or integrated tool environments that serve the needs of multiple participants in the software development process (e.g., SCM, development, V&V). Automated tool support becomes increasingly important, and increasingly difficult to establish, as projects grow in size and as project environments get more complex. These tool capabilities provide support for:

- the SCM Library,
- the software change request (SCR) and approval procedures,
- code (and related work products) and change management tasks,
- reporting software configuration status and collecting SCM measurements,
- software auditing,
- managing and tracking software documentation,
- performing software builds, and
- managing and tracking software releases and their distribution.

The use of tools in these areas increases the potential for obtaining product and process measurements to be used for project management and process improvement purposes. Royce [37] describes seven core measures of value in managing software processes. Information available from the various SCM tools relates to Royce's Work and Progress management indicator and to his quality indicators of Change Traffic and Stability, Breakage and Modularity, Rework and Adaptability, and MTBF (mean time between failures) and Maturity. Reporting on these indicators can be organized in various ways, such as by software configuration item or by type of change requested. Details on specific goals and measures for software processes are described in [Grady].

Figure 3 shows a representative mapping of tool capabilities and procedures to the SCM Activities.

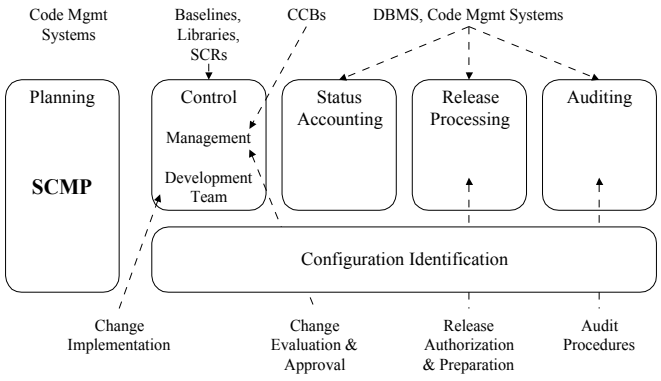

Figure 3 Characterization of SCM Tools and Related Procedures

In this example, code management systems support the operation of software libraries by controlling access to library elements, coordinating the activities of multiple users, and helping to enforce operating procedures. Other tools support the process of building software and release

documentation from the software elements contained in the libraries. Tools for managing software change requests support the change control procedures applied to controlled software items. Other tools can provide database management and reporting capabilities for management, development, and quality assurance activities. As mentioned above, the capabilities of several tool types might be integrated into SCM systems, which, in turn, are closely coupled to various other software activities.

The planning activity assesses the SCM tool needs for a given project within the context of the software engineering environment to be used and selects the tools to be used for SCM. The planning considers issues that might arise in the implementation of these tools, particularly if some form of culture change is necessary. An overview of SCM systems and selection considerations is given in [Dart, (7)], a recent case study on selecting an SCM system is given in [Midha], and [Hoek] provides a current web-based resource listing web links to various SCM tools.

I.C.4 Vendor/Subcontractor Control

A software project might acquire or make use of purchased software products, such as compilers. The planning for SCM considers if and how these items will be taken under configuration control (e.g., integrated into the project libraries) and how changes or updates will be evaluated and managed.

Similar considerations apply to subcontracted software. In this case, the SCM requirements to be imposed on the subcontractor's SCM process as part of the subcontract and the means for monitoring compliance also need to be established. The latter includes consideration of what SCM information must be available for effective compliance monitoring.

I.C.5 Interface Control

When a software item will interface with another software or hardware item, a change to either item can affect the other. The planning for the SCM process considers how the interfacing items will be identified and how changes to the items will be managed and communicated. The SCM role may be part of a larger system-level process for interface specification and control and may involve interface specifications, interface control plans, and interface control documents. In this case, SCM planning for interface control takes place within the context of the system level process. A discussion of the performance of interface control activities is given in [Berlack].

I.D Software Configuration Management Plan

The results of SCM planning for a given project are recorded in a Software Configuration Management Plan (SCMP). The SCMP is a 'living document' that serves as a reference for the SCM process. It is maintained (i.e., updated and approved) as necessary during the software life cycle. In implementing the plans contained in the SCMP, it is typically necessary to develop a number of more detailed, subordinate procedures that define how specific requirements will be carried out during day-to-day activities.

Guidance for the creation and maintenance of an SCMP, based on the information produced by the planning activity, is available from a number of sources, such as [IEEE 828 and IEEE 1042]. This reference provides requirements for the information to be contained in an SCMP. It also defines and describes six categories of SCM information to be included in an SCMP:

1. Introduction (purpose, scope, terms used)
2. SCM Management (organization, responsibilities, authorities, applicable policies, directives, and procedures)
3. SCM Activities (configuration identification, configuration control, etc.)
4. SCM Schedules (coordination with other project activities)
5. SCM Resources (tools, physical, and human resources)
6. SCMP Maintenance

I.E Surveillance of Software Configuration Management

After the SCM process has been implemented, some degree of surveillance may be conducted to ensure that the provisions of the SCMP are properly carried out (e.g., see [Buckley]). There are likely to be specific SQA requirements for ensuring compliance with specified SCM processes and procedures. This could involve an SCM authority ensuring that the defined SCM tasks are performed correctly by those with the assigned responsibility. The software quality assurance authority, as part of a compliance auditing activity, might also perform this surveillance.

The use of integrated SCM tools that have capabilities for process control can make the surveillance task easier. Some tools facilitate process compliance while providing flexibility for the developer to adapt procedures. Other tools enforce process, leaving the developer less flexibility. Surveillance requirements and the level of developer flexibility to be provided are important considerations in tool selection.

I.E.1 SCM Measures and Measurement

SCM measures can be designed to provide specific information on the evolving product or to provide insight into the functioning of the SCM process. A related goal of monitoring the SCM process is to discover opportunities for process improvement. Quantitative measurements against SCM process measures provide a good means for monitoring the effectiveness of SCM activities on an ongoing basis. These measurements are useful in characterizing the current state of the process as well as in providing a basis for making comparisons over time. Analysis of the measurements may produce insights leading

to process changes and corresponding updates to the SCMP.

The software libraries and the various SCM tool capabilities provide sources for extracting information about the characteristics of the SCM process (as well as providing project and management information). For example, information about the processing time required for various types of changes would be useful in an evaluation of the criteria for determining what levels of authority are optimal for authorizing certain types of changes.

Care must be taken to keep the focus of the surveillance on the insights that can be gained from the measurements, not on the measurements themselves.

I.E.2 In-process Audits of SCM

Audits can be carried out during the development process to investigate the current status of specific elements of the configuration or to assess the implementation of the SCM process. In-process auditing of SCM provides a more formal mechanism for monitoring selected aspects of the process and may be coordinated with the SQA auditing function.

II. Software Configuration Identification

The software configuration identification activity identifies items to be controlled, establishes identification schemes for the items and their versions, and establishes the tools and techniques to be used in acquiring and managing controlled items. These activities provide the basis for the other SCM activities.

II.A Identifying Items to be Controlled

A first step in controlling change is to identify the software items to be controlled. This involves understanding the software configuration within the context of the system configuration, selecting software configuration items, developing a strategy for labeling software items and describing their relationships, and identifying the baselines to be used, along with the procedure for a baseline's acquisition of the items.

II.A.1 Software Configuration

A software configuration is the set of functional and physical characteristics of software as set forth in the technical documentation or achieved in a product [IEEE 610]. It can be viewed as a part of an overall system configuration.

II.A.2 Software Configuration Item

A software configuration item (SCI) is an aggregation of software that is designated for configuration management and is treated as a single entity in the SCM process [IEEE 610]. A variety of items, in addition to the code itself, are typically controlled by SCM. Software items with potential to become SCIs include plans, specifications and design documentation, testing materials, software tools, source and executable code, code libraries, data and data dictionaries, and documentation for installation, maintenance, operations and software use.

Selecting SCIs is an important process that must achieve a balance between providing adequate visibility for project control purposes and providing a manageable number of controlled items. A list of criteria for SCI selection is given in [Berlack].

II.A.3 Software Configuration Item Relationships

The structural relationships among the selected SCIs, and their constituent parts, affect other SCM activities or tasks, such as software building or analyzing the impact of proposed changes. Proper tracking of these relationships is also important for supporting traceability verifications. The design of the identification scheme for SCIs should consider the need to map the identified items to the software structure as well as the need to support the evolution of the software items and their relationships.

II.A.4 Software Versions

Software items evolve as a software project proceeds. A *version* of a software item is a particular identified and specified item. It can be thought of as a state of an evolving item [Conradi]. A *revision* is a new version of an item that is intended to replace the old version of the item. A *variant* is a new version of an item that will be added to the configuration without replacing the old version. The management of software versions in various software engineering environments is a current research topic; for example, see [Conradi], [Estublier], and [Sommerville, (39)].

II.A.5 Baseline

A software baseline is a set of software items formally designated and fixed at a specific time during the software life cycle. The term is also used to refer to a particular version of a software item that has been agreed upon. In either case, the baseline can only be changed through formal change control procedures. A baseline, together with all approved changes to the baseline, represents the current approved configuration.

Commonly used baselines are the functional, allocated, developmental, and product baselines; e.g. see [Berlack]. The functional baseline corresponds to the reviewed system requirements. The allocated baseline corresponds to the reviewed software requirements specification and software interface requirements specification. The developmental baseline represents the evolving software configuration at selected times during the software life cycle. Change authority for this baseline typically rests primarily with the development organization, but may be shared by other organizations (e.g., SCM or Test). The product baseline corresponds to the completed software product delivered for system integration. The baselines to be used for a given project, along with their associated levels of authority

needed for change approval, are typically identified in the SCMP.

II.A.6 Acquiring Software Configuration Items

Software configuration items are placed under SCM control at different times; i.e. they are incorporated into a particular baseline at a particular point in the software life cycle. The triggering event is the completion of some form of formal acceptance task, such as a formal review. Figure 4 characterizes the growth of baselined items as the life cycle proceeds. This figure is based on a waterfall model for purposes of illustration only; the subscripts used in the figure indicate versions of the evolving items. The software change request (SCR) is described in section III.A.

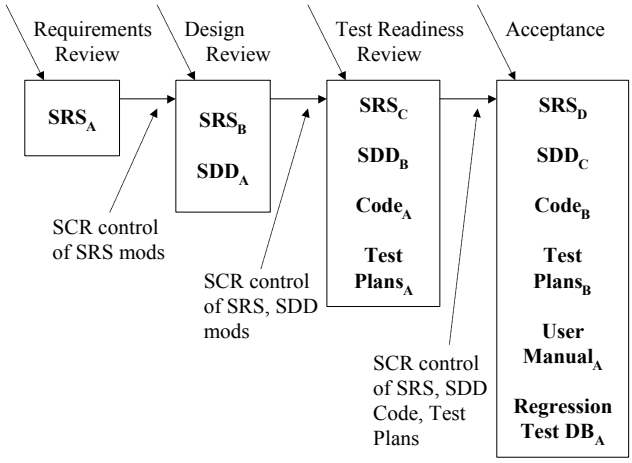

Figure 4 Acquisition of Items

Following the acquisition of an SCI, changes to the item must be formally approved as appropriate for the SCI and the baseline involved, as defined in the SCMP. Following the approval, the item is incorporated into the software baseline according to the appropriate procedure.

II.B Software Library

A software library is a controlled collection of software and related documentation designed to aid in software development, use, and maintenance [IEEE 610]. It is also instrumental in software release and delivery activities. Several types of libraries might be used, each corresponding to a particular level of maturity of the software item. For example a working library could support coding and a project support library could support testing, whereas a master library could be used for finished products. An appropriate level of SCM control (associated baseline and level of authority for change) is associated with each library. Security, in terms of access control and the backup facilities, is a key aspect of library management. A model of a software library is described in [Berlack].

The tool(s) used for each library must support the SCM control needs for that library, both in terms of controlling SCIs and controlling access to the library. At the working library level, this is a code management capability serving developers, maintainers and SCM. It is focused on managing the versions of software items while supporting the activities of multiple developers. At higher levels of control, access is more restricted and SCM is the primary user.

These libraries are also an important source of information for measurements of work and progress.

III. Software Configuration Control

Software configuration control is concerned with managing changes during the software life cycle. It covers the process for determining what changes to make, the authority for approving certain changes, support for the implementation of those changes, and the concept of formal deviations and waivers from project requirements. Information derived from these activities is useful in measuring change traffic, breakage, and aspects of rework.

III.A. Requesting, Evaluating and Approving Software Changes

The first step in managing changes to controlled items is determining what changes to make. The software change request process (see Figure 5) provides formal procedures for submitting and recording change requests, evaluating the potential cost and impact of a proposed change, and accepting, modifying or rejecting the proposed change. Requests for changes to software configuration items may be originated by anyone at any point in the software life cycle and may include a suggested solution and requested priority. One source of change requests is the initiation of corrective action in response to problem reports. Regardless of the source, the type of change (e.g. defect or enhancement) usually recorded on the SCR.

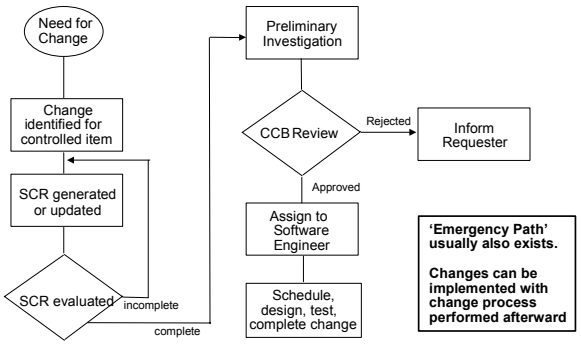

Figure 5 Flow of a Change Control Process

This provides an opportunity for tracking defects and collecting change activity measurements by change type. Once an SCR is received, a technical evaluation (also known as an impact analysis) is performed to determine the extent of modifications that would be necessary should the change request be accepted. A good understanding of the relationships among software (and possibly, hardware) items is important for this task. Finally, an established

authority, commensurate with the affected baseline, the SCI involved, and the nature of the change, will evaluate the technical and managerial aspects of the change request and either accept, modify, reject or defer the proposed change.

III.A.1. Software Configuration Control Board

The authority for accepting or rejecting proposed changes rests with an entity typically known as a Configuration Control Board (CCB). In smaller projects, this authority actually may reside with the responsible leader or an assigned individual rather than a multi-person board. There can be multiple levels of change authority depending on a variety of criteria, such as the criticality of the item involved, the nature of the change (e.g., impact on budget and schedule), or the current point in the life cycle. The composition of the CCBs used for a given system varies depending on these criteria (an SCM representative would always be present). All stakeholders, appropriate to the level of the CCB, are represented. When the scope of authority of a CCB is strictly software, it is known as a software configuration control board (SCCB). The activities of the CCB are typically subject to SQA audit or review.

III.A.2 Software Change Request Process

An effective SCR process requires the use of supporting tools and procedures ranging from paper forms and a documented procedure to an electronic tool for originating change requests, enforcing the flow of the change process, capturing CCB decisions, and reporting change process information. A link between this tool capability and the problem reporting system can facilitate the tracking of solutions for reported problems. Change process descriptions and supporting forms (information) are given in a variety of references, e.g. [Berlack] and [IEEE 1042]. Typically, change management tools are tailored to local processes and tool suites and are often locally developed. The current trend is towards integration of these kinds of tools within a suite referred to as a software engineering environment.

III.B. Implementing Software Changes

Approved change requests are implemented using the defined software procedures in accordance with the applicable schedule requirements. Since a number of approved change requests might be implemented simultaneously, it is necessary to provide a means for tracking which change requests are incorporated into particular software versions and baselines. As part of the closure of the change process, completed changes may undergo configuration audits and SQA verification. This includes ensuring that only approved changes were made. The change request process described above will typically document the SCM (and other) approval information for the change.

The actual implementation of a change is supported by the library tool capabilities that provide version management and code repository support. At a minimum, these tools provide check-in/out and associated version control capabilities. More powerful tools can support parallel development and geographically distributed environments. These tools may be manifested as separate specialized applications under control of an independent SCM group. They may also appear as an integrated part of the software development environment. Finally, they may be as elementary as a rudimentary change control system provided with an operating system.

III.C. Deviations and Waivers

The constraints imposed on a software development effort or the specifications produced during the development activities might contain provisions that cannot be satisfied at the designated point in the life cycle. A deviation is an authorization to depart from a provision prior to the development of the item. A waiver is an authorization to use an item, following its development, that departs from the provision in some way. In these cases, a formal process is used for gaining approval for deviations to, or waivers of, the provisions.

IV. Software Configuration Status Accounting

Software configuration status accounting (SCSA) is the recording and reporting of information needed for effective management of the software configuration. The design of the SCSA capability can be viewed from an information systems perspective, utilizing accepted information systems design techniques.

IV.A. Software Configuration Status Information

The SCSA activity designs and operates a system for the capture and reporting of necessary information as the life cycle proceeds. As in any information system, the configuration status information to be managed for the evolving configurations must be identified, collected, and maintained. Various information and measurements are needed to support the SCM process and to meet the configuration status reporting needs of management, software engineering, and other related activities. The types of information available include the approved configuration identification as well as the identification and current implementation status of changes, deviations and waivers. A partial list of important data elements is given in [Berlack].

Some form of automated tool support is necessary to accomplish the SCSA data collection and reporting tasks. This could be a database capability, such as a relational or object-oriented database management system. This could be a stand-alone tool or a capability of a larger, integrated tool environment.

IV.B. Software Configuration Status Reporting

Reported information can be used by various organizational and project elements, including the development team, the maintenance team, project management, and quality

assurance activities. Reporting can take the form of ad hoc queries to answer specific questions or the periodic production of pre-designed reports. Some information produced by the status accounting activity during the course of the life cycle might become quality assurance records.

In addition to reporting the current status of the configuration, the information obtained by SCSA can serve as a basis for various measurements of interest to management, development, and SCM. Examples include the number of change requests per SCI and the average time needed to implement a change request.

V. Software Configuration Auditing

A software audit is an activity performed to independently evaluate the conformance of software products and processes to applicable regulations, standards, guidelines, plans, and procedures [IEEE 1028]. Audits are conducted according to a well-defined process consisting of various auditor roles and responsibilities. Consequently, each audit must be carefully planned. An audit can require a number of individuals to perform a variety of tasks over a fairly short period of time. Tools to support the planning and conduct of an audit can greatly facilitate the process. Guidance for conducting software audits is available in various references, such as [Berlack], [Buckley], and [IEEE 1028].

The software configuration auditing activity determines the extent to which an item satisfies the required functional and physical characteristics. Informal audits of this type can be conducted at key points in the life cycle. Two types of formal audits might be required by the governing contract (e.g., in contracts covering critical software): the Functional Configuration Audit (FCA) and the Physical Configuration Audit (PCA). Successful completion of these audits can be a prerequisite for the establishment of the product baseline. Buckley [5] contrasts the purposes of the FCA and PCA in hardware versus software contexts and recommends careful evaluation of the need for the software FCA and PCA before performing them.

V.A. Software Functional Configuration Audit

The purpose of the software FCA is to ensure that the audited software item is consistent with its governing specifications. The output of the software verification and validation activities is a key input to this audit.

V.B. Software Physical Configuration Audit

The purpose of the software PCA is to ensure that the design and reference documentation is consistent with the as-built software product.

V.C. In-process Audits of a Software Baseline

As mentioned above, audits can be carried out during the development process to investigate the current status of specific elements of the configuration. In this case, an audit could be applied to sampled baseline items to ensure that performance was consistent with specification or to ensure that evolving documentation was staying consistent with the developing baseline item.

VI. Software Release Management and Delivery

The term “release” is used in this context to refer to the distribution of a software configuration item outside the development activity. This includes internal releases as well as distribution to customers. When different versions of a software item are available for delivery, such as versions for different platforms or versions with varying capabilities, it is frequently necessary to recreate specific versions and package the correct materials for delivery of the version. The software library is a key element in accomplishing release and delivery tasks.

VI.A. Software Building

Software building is the activity of combining the correct versions of software items, using the appropriate configuration data, into an executable program for delivery to a customer or other recipient, such as the testing activity. For systems with hardware or firmware, the executable is delivered to the system building activity. Build instructions ensure that the proper build steps are taken and in the correct sequence. In addition to building software for new releases, it is usually also necessary for SCM to have the capability to reproduce previous releases for recovery, testing, or additional release purposes.

Software is built using particular versions of supporting tools, such as compilers. It might be necessary to rebuild an exact copy of a previously built software item. In this case, the supporting tools and associated build instructions need to be under SCM control to ensure availability of the correct versions of the tools.

A tool capability is useful for selecting the correct versions of software items for a given target environment and for automating the process of building the software from the selected versions and appropriate configuration data. For large projects with parallel development or distributed development environments, this tool capability is necessary. Most software development environments provide this capability. These tools vary in complexity from requiring the engineer to learn a specialized scripting language to graphics-oriented approaches that hide much of the complexity of an “intelligent” build facility.

The build process and products are often subject to SQA verification. Outputs of the build process might be needed for future reference and may become quality assurance records.

VI.B Software Release Management

Software release management encompasses the identification, packaging and delivery of the elements of a product, for example, the executable, documentation, release notes, and configuration data. Given that product changes can be occurring on a continuing basis, one issue for release management is determining when to issue a release. The severity of the problems addressed by the

release and measurements of the fault densities of prior releases affect this decision [Sommerville, (38)]. The packaging task must identify which product items are to be delivered and select the correct variants of those items, given the intended application of the product. The set of information documenting the physical contents of a release is known as a version description document and may exist in hardcopy or electronic form. The release notes typically describe new capabilities, known problems, and platform requirements necessary for proper product operation. The package to be released also contains loading or upgrading instructions. The latter can be complicated by the fact that some current users might have versions that are several releases old. Finally, in some cases, the release management activity might be required to track the distribution of the product to various customers or target systems. An example would be a case where the supplier was required to notify a customer of newly reported problems.

A tool capability is needed for supporting these release management functions. It is useful to have a connection with the tool capability supporting the change request process in order to map release contents to the SCRs that have been received. This tool capability might also maintain information on various target platforms and on various customer environments.

4 BREAKDOWN RATIONALE

One of the primary goals of the Guide to the SWEBOK is to arrive at a breakdown that is 'generally accepted'. Consequently, the breakdown of SCM topics was developed largely by attempting to synthesize the topics covered in the literature and in recognized standards, which tend to reflect consensus opinion. The topic on Software Release Management and Delivery is an exception since it has not commonly been broken out separately in the past. The precedent for this was set by the ISO/IEC 12207 standard [23], which identifies a 'Release Management and Delivery' activity.

There is widespread agreement in the literature on the SCM activity areas and their key concepts. However, there continues to be active research on implementation aspects of SCM. Examples are found in ICSE workshops on SCM such as [Estublier] and [Sommerville, (39)].

5 MATRIX OF TOPICS VS. REFERENCE MATERIAL

Table 1. Coverage of the Breakdown Topics by the Recommended References

	Babich	Berlack	Buckley	Conradi	Dart	Hoek	IEEE 828	IEEE/EIA 12207	Midha	Moore	Paulk	Pressman	Royce	Sommerville
I. Management of the SCM Process														
A. Organizational Context for SCM		C4	C2		C2		4.2.1							
B. Constraints and Guidance for SCM		C5					4.1, 4.2.3			X				
C. Planning for SCM					C2			6.2.1						C33
1. SCM Organization and Responsibilities		C7	C3				4.2							
2. SCM Resources and Schedules		C7	C3				4.4, 4.5							
3. Tool Selection and Implementation		C15		C6	C3, App A	X			X			C29		
4. Vendor/Subcontractor Control		C13	C11				4.3.6							
5. Interface Control		C12					4.3.5							
D. SCM Plan		C7	C3				4				L2-81			
E. Surveillance of SCM											L2-87			
1. SCM Measures and Measurement			C3										202,283-	
2. In-Process Audits of SCM			C15											
II. Software Configuration Identification								6.2.2						
A. Identifying Items to be Controlled		C8					4.3.1				L2-83			C33
1. Software Configuration			C4,6									C9		
2. Software Configuration Item			C4,6	C2								C9		
3. Software Configuration Item Relationships				C2								C9		
4. Software Versions	C2			C3,C4,C5								C9		
5. Baseline	C5		C4									C9		
6. Acquiring Software Configuration Items			C4											
B. Software Library	C2,5	C14	C4				4.3.1				L2-82			C33
III. Software Configuration Control								6.2.3			L2-84			
A. Requesting, Evaluating and Approving Software Changes							4.3.2					C9		C33
1. Software Configuration Control Board		C9	C9,11									C9		
2. Software Change Request Process		C9	C9,11									C9		

	Babich	Berlack	Buckley	Conradi	Dart	Hoek	IEEE 828	IEEE/EIA 12207	Midha	Moore	Paulk	Pressman	Royce	Sommerville
B. Implementing Software Changes	C6	C9	C9,11				4.3.2.4					C9		C33
C. Deviations & Waivers		C9	C12											
IV. Software Configuration Status Accounting								6.2.4			L2-85	C9		C33
A. Software Configuration Status Inf.		C10	C13				4.3.3							
B. Software Configuration Status Rptg.		C10	C13											
V. Software Configuration Auditing							4.3.4	6.2.5			L2-86	C9,C17		
A. Software Functional Configuration Audit		C11	C15											
B. Software Physical Configuration Audit		C11	C15											
C. In-Process Audits of a Software Baseline			C15											
VI. Software Release Management and Delivery								6.2.6						
A. Software Building	C6													C33
B. Software Release Management														C33

6 RECOMMENDED REFERENCES FOR SCM

Cross Reference Matrix

Table 1, in Appendix A, provides a cross reference between the recommended references and the topics of the breakdown. Note that, where a recommended reference is also shown in the Further Reading section, the cross reference reflects the full text rather than just the specific passage referenced in the Recommended References.

Recommended References

Specific recommendations are made here to provide additional information on the topics of the SCM breakdown.

W.A. Babich, Software Configuration Management, Coordination for Team Productivity, Addison-Wesley, 1986 [1]

Pages 20-43 address the basics of code management.

H.R. Berlack, Software Configuration Management, Wiley 1992 [2]

See pages 101-175 on configuration identification, configuration control and configuration status accounting, and pages 202-206 on libraries.

F.J. Buckley, Implementing Configuration Management: Hardware, Software, and Firmware 2nd edition, IEEE Computer Society Press, 1996 [5]

See pages 10-19 on organizational context, pages 21-38 on CM planning, and 228-250 on CM auditing.

R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management", ACM Computing Surveys, vol. 30, no. 2, June 1998 [6]

An in-depth article on version models used in software configuration management. It defines fundamental concepts and provides a detailed view of versioning paradigms. The versioning characteristics of various SCM systems are discussed.

S.A. Dart, Spectrum of Functionality in Configuration Management Systems [7]

This report covers features of various CM systems and the scope of issues concerning users of CM systems. As of this writing, the report can be found on the Internet at: http://www.sei.cmu.edu/about/website/search.html

Hoek, "Configuration Management Yellow Pages," [13]

This web page provides a current compilation of SCM resources.
http://www.cmtoday.com/yp/configuration_management.html

IEEE/EIA Std 12207.0-1996, Software Life Cycle Processes, [20] and IEEE/EIA Std 12207.1-1996, Software Life Cycle Processes - Life Cycle Data, [21]

These standards provide the ISO/IEC view of software processes along with specific information on life cycle data keyed to software engineering standards of other standards bodies.

IEEE Std.828-1990, IEEE Standard for Software Configuration Management Plans [17] and IEEE Std.1042-1987, IEEE Guide to Software Configuration Management [19]

These standards focus on SCM activities by specifying requirements and guidance for preparing the SCMP. These standards reflect commonly accepted practice for software configuration management.

A.K. Midha, "Software Configuration Management for the 21st Century", Bell Technical Labs Journal, vol. 2 no. 1, Winter 1997, pp. 154-165 [30]

This article discusses the characteristics of SCM systems, assessment of SCM needs in a particular environment, and the issue of selecting and implementing an SCM system. It is a current case study on this issue.

J.W. Moore, Software Engineering Standards, A User's Road Map, IEEE Computer Society Press, 1998 [31]

Pages 118-119 cover SCM and pages 194-223 cover the perspective of the 12207 standards.

M.C. Paulk, et al., Key Practices of the Capability Maturity Model, Software Engineering Institute, 1993 [32]

Pages 180-191 cover the SCM key process area of the SEI CMM.

R.S. Pressman, Software Engineering: A Practitioner's Approach, 4th edition, McGraw-Hill, 1997 [36]

Pages 209-226 address SCM in the context of a textbook on software engineering.

Walker Royce, Software Project Management, A Unified Framework, Addison-Wesley, 1998 [37]

Pages 188-202 and 283-298 cover measures of interest to software project management that are closely related to SCM.

I. Sommerville, Software Engineering, 5th edition, Addison-Wesley, 1996 [38]

Pages 675-696 cover SCM with an emphasis on software building and release management.

APPENDIX A – LIST OF FURTHER READINGS

The following set of references was chosen to provide coverage of all aspects of SCM, from various perspectives and to varying levels of detail. The author and title are cited; the complete reference is given in the References section. Some items overlap with those in the Recommended References since they cover the full texts rather than specific passages.

W.A. Babich, Software Configuration Management, Coordination for Team Productivity [1]

This text is focused on code management issues from the perspective of the development team.

H.R. Berlack, Software Configuration Management [2]

This textbook provides detailed, comprehensive coverage of the concepts of software configuration management. This is one of the more recent texts with this focus.

F.J. Buckley, Implementing Configuration Management: Hardware, Software, and Firmware [5]

This text presents an integrated view of configuration management for projects in which software, hardware and firmware are involved. It is a recent text that provides a view of software configuration management from a systems perspective.

J. Estublier, Software Configuration Management, ICSE SCM-4 and SCM-5 Workshops Selected Papers [10]

These workshop proceedings are representative of current experience and research on SCM. This reference is included with the intention of directing the reader to the whole class of conference and workshop proceedings.

The suite of IEEE/EIA and ISO/IEC 12207 standards, [20]-[24]

These standards cover software life cycle processes and address SCM in that context. These standards reflect commonly accepted practices for software life cycle processes. Note - the developing ISO/IEC TR 15504 (SPICE99) expands on SCM within the context of the ISO/IEC 12207 standard.

IEEE Std.1042-1987, IEEE Guide to Software Configuration Management [19]

This standard provides guidance, keyed to IEEE 828, for preparing the SCMP.

J.W. Moore, Software Engineering Standards, A User's Road Map [31]

This text provides a comprehensive view of current standards and standards activities in the area of software engineering.

APPENDIX B – REFERENCES USED TO WRITE AND JUSTIFY THE KNOWLEDGE AREA DESCRIPTION

These references were used in preparing this paper; the recommended references for SCM are listed in Section 3.1.

1. W.A. Babich, Software Configuration Management: Coordination for Team Productivity, Addison-Wesley, Reading, Massachusetts, 1986.
2. H.R. Berlack, Software Configuration Management, John Wiley & Sons, New York, 1992.
3. E.H. Bersoff, "Elements of Software Configuration Management," Software Engineering, M. Dorfman and R.H. Thayer ed., IEEE Computer Society Press, Los Alamitos, CA, 1997.
4. E.H. Bersoff and A.M. Davis, "Impacts of Life Cycle Models on Software Configuration Management," Communications of the ACM, Vol. 34, No. 8, August 1991, pp104-118.
5. F.J. Buckley, Implementing Configuration Management: Hardware, Software, and Firmware, Second Edition, IEEE Computer Society Press, Los Alamitos, CA, 1996.
6. R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," ACM Computing Surveys, Vol. 30, No. 2, June 1998, pp. 232-282.
7. S.A. Dart, Spectrum of Functionality in Configuration Management Systems, Technical Report CMU/SEI-90-TR-11, Software Engineering Institute, Carnegie Mellon University, 1990.
8. S.A. Dart, "Concepts in Configuration Management Systems," Proceedings of the Third International Workshop on Software Configuration Management, ACM Press, New York, 1991, pp1-18.
9. Khaled El Emam, et al., SPICE, The Theory and Practice of Software Process Improvement and Capability Determination, IEEE Computer Society, Los Alamitos, CA, 1998.
10. J. Estublier, Software Configuration Management, ICSE SCM-4 and SCM-5 Workshops Selected Papers, Springer-Verlag, Berlin, 1995.
11. P.H. Feiler, Configuration Management Models in Commercial Environments, Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie Mellon University, 1991.
12. R.B. Grady, Practical Software Metrics for Project Management and Process Improvement, Prentice-Hall, Englewook Cliffs, NJ, 1992.
13. Hoek, "Configuration Management Yellow Pages," http://www.cs.colorado.edu/users/andre/configuration_management.html
14. W.S. Humphrey, Managing the Software Process, Addison-Wesley, Reading, MA, 1989.
15. IEEE Std.610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE, Piscataway, NJ, 1990.
16. IEEE Std.730-1998, IEEE Standard for Software Quality Assurance Plans, IEEE, Piscataway, NJ, 1998.
17. IEEE Std.828-1998, IEEE Standard for Software Configuration Management Plans, IEEE, Piscataway, NJ, 1998.
18. IEEE Std.1028-1997, IEEE Standard for Software Reviews, IEEE, Piscataway, NJ, 1997.
19. IEEE Std.1042-1987, IEEE Guide to Software Configuration Management, IEEE, Piscataway, NJ, 1987.
20. IEEE/EIA Std 12207.0-1996, Software Life Cycle Processes, IEEE, Piscataway, NJ, 1996.
21. IEEE/EIA Std 12207.1-1996, Guide for Software Life Cycle Processes – Life Cycle Data, IEEE, Piscataway, NJ, 1996.
22. IEEE/EIA Std 12207.2-1996, Guide for Software Life Cycle Processes – Implementation Considerations, IEEE, Piscataway, NJ, 1996.
23. ISO/IEC 12207:1995(E), Information Technology - Software Life Cycle Processes, ISO/IEC, Geneve, Switzerland, 1995.
24. ISO/IEC TR 15846:1998, Information Technology - Software Life Cycle Processes - Configuration Management , ISO/IEC, Geneve, Switzerland, 1998.
25. ISO/DIS 9004-7 (now ISO 10007), Quality Management and Quality System Elements, Guidelines for Configuration Management, International Organization for Standardization, Geneve, Switzerland, 1993.
26. P. Jalote, An Integrated Approach to Software Engineering, Springer-Verlag, New York, 1997
27. John J. Marciniak and Donald J. Reifer, Software Acquisition Management, Managing the Acquisition of Custom Software Systems, John Wiley & Sons, 1990.
28. J.J. Marciniak, "Reviews and Audits," Software Engineering, M. Dorfman and R.H. Thayer ed., IEEE Computer Society Press, Los Alamitos, CA, 1997.
29. K. Meiser, "Software Configuration Management Terminology," Crosstalk, 1995, http://www.stsc.hill.af.mil/crosstalk/1995/jan/terms.html, February 1999.
30. A.K. Midha, "Software Configuration Management for the 21st Century," Bell Labs Technical Journal, Winter 1997.
31. J.W. Moore, Software Engineering Standards, A User's Roadmap, IEEE Computer Society, Los Alamitos, CA, 1998.

32. M.C. Paulk, et al., Key Practices of the Capability Maturity Model, Version 1.1, Technical Report CMU/SEI-93-TR-025, Software Engineering Institute, Carnegie Mellon University, 1993
33. M.C. Paulk, et al., The Capability Maturity Model, Guidelines for Improving the Software Process, Addison-Wesley, Reading, Massachusetts, 1995.
34. S.L. Pfleeger, Software Engineering: Theory and Practice, Prentice Hall, Upper Saddle River, NJ, 1998
35. R.K. Port, "Software Configuration Management Technology Report, September 1994, " http://www.stsc.hill.af.mil/cm/REPORT.html, February 1999.
36. R.S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, New York, 1997.
37. Walker Royce, Software Project Management, A United Framework, Addison-Wesley, Reading, Massachusetts, 1998.
38. Sommerville, Software Engineering, Fifth Edition, Addison-Wesley, Reading, Massachusetts, 1995.
39. Sommerville, Software Configuration Management, ICSE SCM-6 Workshop, Selected Papers, Springer-Verlag, Berlin, 1996.
40. USNRC Regulatory Guide 1.169, Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants, U.S. Nuclear Regulatory Commission, Washington DC, 1997.
41. J.P. Vincent, et al., Software Quality Assurance, Prentice-Hall, Englewood Cliffs, NJ, 1988.
42. W.G. Vincenti, What Engineers Know and How They Know It, The Johns Hopkins University Press, Baltimore, MD, 1990.
43. D. Whitgift, Methods and Tools for Software Configuration Management, John Wiley & Sons, Chichester, England, 1991.

M.C. Paulk, et al., Key Practices of the Capability Maturity Model [32]

This report describes the key practices that could be evaluated in assessing software process maturity. Therefore, the section on SCM key practices provides a view of SCM from a software process assessment perspective.

R.S. Pressman, Software Engineering: A Practitioner's Approach [36]

This reference and the Sommerville reference address SCM in the context of a textbook on software engineering.

I. Sommerville, Software Engineering [38]

This reference and the Pressman reference address SCM in the context of a textbook on software engineering.

J.P. Vincent, et al., Software Quality Assurance [41]

In this text, SCM is described from the perspective of a complete set of assurance processes for a software development project.

D. Whitgift, Methods and Tools for Software Configuration Management [43]

This text covers the concepts and principles of SCM. It provides detailed information on the practical questions of implementing and using tools. This text is out of print but still available in libraries.

APPENDIX C – RATIONALE DETAILS

Criterion (a): Number of topic breakdowns

One breakdown is provided.

Criterion (b): Reasonableness

The breakdowns are reasonable in that they cover the areas typically discussed in texts and standards, although there is somewhat less discussion of release management as a separate topic. In response to comments on version 0.5 of the paper, the tool discussion under 'Planning for SCM' has been expanded. The various tool subheadings used throughout the text have been removed (so they do not appear as topics), however, the supporting text has been retained and incorporated into the next higher level topics.

Criterion (c): Generally Accepted

The breakdowns are generally accepted in that they cover the areas typically discussed in texts and standards.

At level 1, the breakdown is identical to that given in IEC 12207 (Section 6.2) except that the term "Management of the Software Configuration Management Process" was used instead of "Process Implementation" and the term "Software Configuration Auditing" was used instead of "Configuration Evaluation." The typical texts discuss Software Configuration Management Planning (our topic A.3); We have expanded this to a "management of the process" concept in order to capture related ideas expressed in many of the references that we have used. These ideas are captured in topics A.1 (organizational context), A.2 (constraints and guidance), and A.4 (surveillance of the SCM process). A similar comparison can also be made to [Buckley] except for the addition of "Software Release Management and Delivery."

We have chosen to include the word "Software" as a prefix to most of the configuration topics to distinguish the topics from hardware CM or system level CM activities. We would reserve "Configuration Management" for system purposes and then use HCM and SCM for hardware and software respectively.

The topic A.1, "Software Configuration Management Organizational Context," covers key topics addressed in multiple texts and articles and it appears within the level 1 headings consistently with the placement used in the references. This new term on organizational context was included as a placeholder for capturing three concepts found in the references. First, [Buckley] discusses SCM in the overall context of a project with hardware, software, and firmware elements. We believe that this is a link to a related discipline of system engineering. (This is similar to what IEEE 828 discusses under the heading of "Interface Control"). Second, SCM is one of the product assurance processes supporting a project, or in IEC 12207 terminology, one of the supporting lifecycle processes. The processes are closely related and, therefore, interfaces to them should be considered in planning for SCM. Finally, some of the tools for implementing SCM might be the same tools used by the developers. Therefore, in planning SCM, there should be awareness that the implementation of SCM is strongly affected by the environment chosen for the development activities.

The inclusion of the topic "Release Management and Delivery" is somewhat controversial since the majority of texts on software configuration management devote little or no attention to the topic. We believe that most writers assume the library function of configuration identification would support release management and delivery but, perhaps, assume that these activities are the responsibility of project or line management. The IEC 12207 standard, however, has established this as a required area for SCM. Since this has occurred and since this topic should be recognized somewhere in the overall description of software activities, "Release Management and Delivery" has been included.

Criterion (d): No Specific Application Domains

No specific application domains have been assumed.

Criterion (e): Compatible with Various Schools of Thought

SCM concepts are fairly stable and mature.

Criterion (f): Compatible with Industry, Literature, and Standards

The breakdown was derived from the literature and from key standards reflecting consensus opinion. The extent to which industry implements the SCM concepts in the literature and in standards varies by company and project.

Criterion (g): As Inclusive as Possible

The inclusion of the level 1 topic on management of SCM expands the planning concept into a larger area that can cover all management-related topics, such as surveillance of the SCM process. For each level 1 topic, the level 2 topics categorize the main areas in various references' discussions of the level 1 topic. These are intended to be general enough to allow an open-ended set of subordinate level 3 topics on specific issues. The level 3 topics cover specifics found in the literature but are not intended to provide an exhaustive breakdown of the level 2 topic.

Criterion (h): Themes of Quality and Measurement

The relationship of SCM to product assurance and measurement is provided for in the breakdowns. The description also conveys the role of SCM in achieving a consistent, verified, and validated product.

Criterion (i): 2 to 3 levels, 5 to 9 topics at the first level

The proposed breakdown satisfies this criterion.

Criterion (j): Topic Names Meaningful Outside the Guide

For the most part, we believe this is the case. Some terms, such a "Baselines" or "Physical Configuration Audit"

require some explanation but they are obviously the terms to use since appear throughout the literature.

Criterion (k): Topics only sufficiently described to allow reader to select appropriate material

We believe this has been accomplished. We have not attempted to provide a tutorial on SCM.

Criterion (l): Text on the Rationale Underlying the Proposed Breakdowns

This document provides the rationale.