CHAPTER 7

SOFTWARE CONFIGURATION MANAGEMENT

ACRONYMS

CCB	Configuration Control Board
CM	Configuration Management
FCA	Functional Configuration Audit
MTBF	Mean Time Between Failures
PCA	Physical Configuration Audit
SCCB	Software Configuration Control Board
SCI	Software Configuration Item
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	Software Change Request
SCSA	Software Configuration Status Accounting
SEI/CMMI	Software Engineering Institute's Capability Maturity Model Integration
SQA	Software Quality Assurance
SRS	Software Requirement Specification
USNRC	U.S. Nuclear Regulatory Commission

INTRODUCTION

A *system* can be defined as a collection of components organized to accomplish a specific function or set of functions (IEEE 610.12-90). The *configuration* of a system is the functional and/or physical characteristics of hardware, firmware, or software, or a combination of these, as set forth in technical documentation and achieved in a product. (Buc96) 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 serve 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. (Ber97) It is formally defined (IEEE610.12-90) 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."

Software configuration management (SCM) is a supporting software life cycle process (IEEE12207.0-96) which benefits project management, development and maintenance activities, assurance activities, and the customers and users of the end product.

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.

SCM is closely related to the software quality assurance (SQA) activity. As defined in the Software Quality KA, SQA processes provide assurance that the software products and processes in the project life cycle conform to their specified requirements by planning, enacting, and performing a set of activities to provide adequate confidence that quality is being built into the software. SCM activities help in accomplishing these SQA goals. In some project contexts (see, for example, IEEE730-02), specific SQA requirements prescribe certain SCM activities.

The SCM activities are: management and planning of the SCM 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.

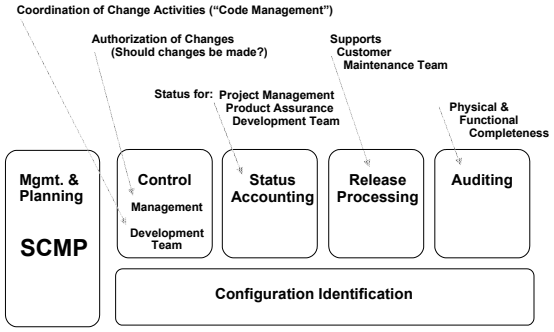

Figure 1. SCM Activities

The Software Configuration Management KA is related to all the other KAs, since the object of configuration management is the artifact produced and used throughout the software engineering process.

BREAKDOWN OF TOPICS FOR SCM

1. Management of the SCM Process

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 software engineer's perspective, SCM facilitates 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 on, the design and implementation of the SCM process.

1.1. Organizational Context for SCM
[Ber92 :c4; Dar90:c2; IEEE828-98:c4s2.1]

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

The organizational elements responsible for the software engineering supporting processes may be structured in various ways. Although the responsibility for performing certain SCM tasks might be assigned to other parts of the organization such as the development organization, the overall responsibility for SCM often 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 [Buc96:c2] describes SCM within this context. Note that firmware contains hardware and software, therefore both hardware and software CM concepts are applicable.

SCM might 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 nonconforming items is usually the responsibility of the quality assurance activity; however, SCM might assist with tracking and reporting on software configuration items falling into this category.

Perhaps the closest relationship is with the software development and maintenance organizations.

It is within this context that many of the software configuration control tasks are conducted. Frequently, the same tools support development, maintenance, and SCM purposes.

1.2. Constraints and Guidance for the SCM Process
[Ber92:c5; IEEE828-98:c4s1,c4s2.3; Moo98]

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 CM. When software products to be developed have the potential to affect public safety, external regulatory bodies may impose constraints (see, for example, USNRC1.169-97). Finally, the particular software life cycle process chosen for a software project and the tools selected to implement the software affect the design and implementation of the SCM process. [Ber92]

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 [Moo98] 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 Integration (SEI/CMMI) (SEI01) and ISO/IEC15504 Software Engineering–Process Assessment (ISO/IEC 15504-98).

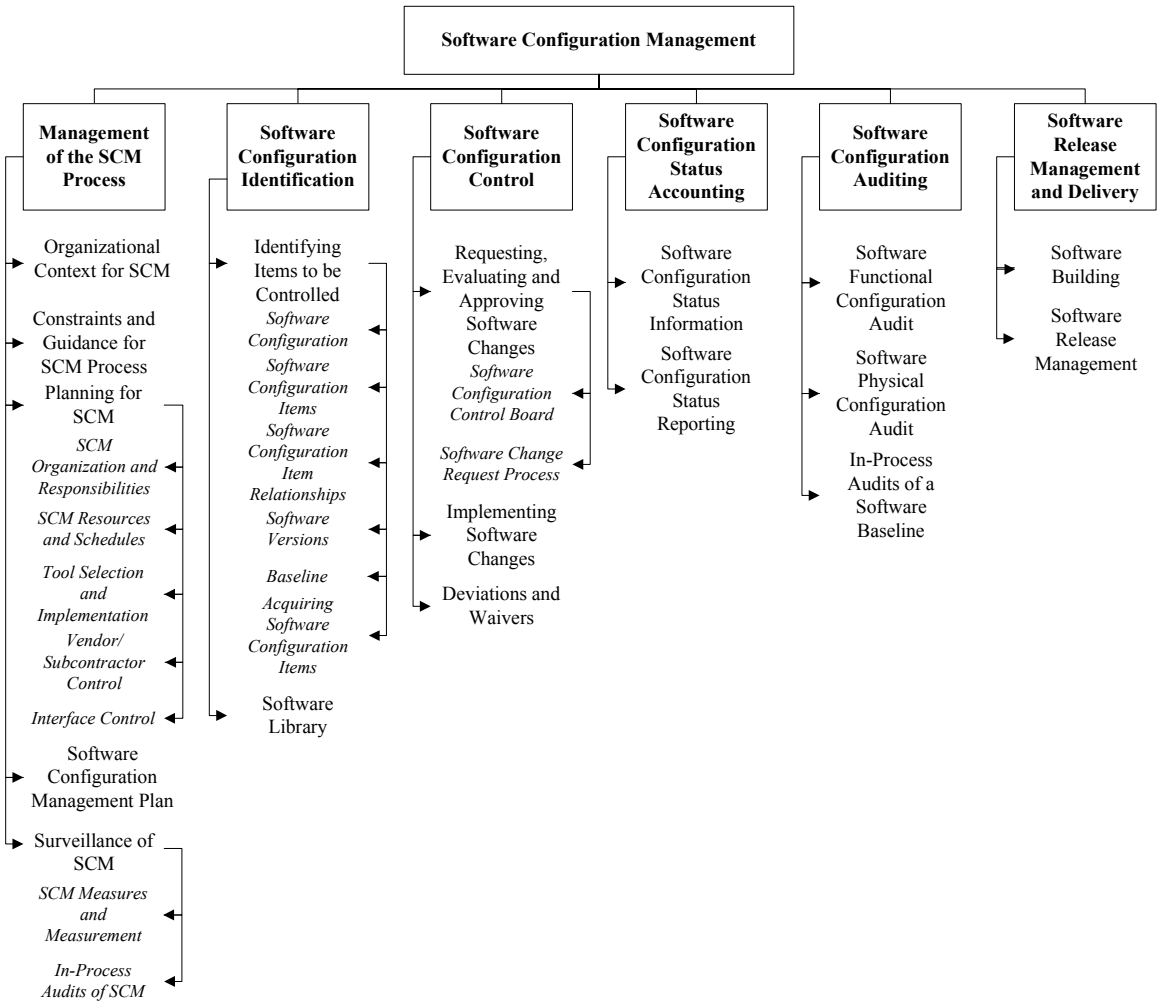

Figure 2 Breakdown of topics for the Software Configuration Management KA

1.3. Planning for SCM
[Dar90:c2; IEEE12207.0-96 :c6.s2.1; Som01:c29]

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 (for example, 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 an SCM Plan (SCMP), which is typically subject to SQA review and audit.

1.3.1. SCM organization and responsibilities
[Ber92:c7; Buc96:c3; IEEE828-98:c4s2]

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 by organizational element. The overall authority and reporting channels for SCM should also be identified, although this might be accomplished at the project management or quality assurance planning stage.

1.3.2. SCM resources and schedules
[Ber92:c7; Buc96:c3; IEEE828-98:c4s4; c4s5]

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

1.3.3. Tool selection and implementation
[Ber92:c15; Con98:c6; Pre01:c31]

Different types of tool capabilities, and procedures for their use, support 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 which serve the needs of multiple participants in the software engineering process (for example, SCM, development, V&V). Automated tool support becomes increasingly important, and increasingly difficult to establish, as projects grow in size and as project environments become 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 configuration auditing
- managing and tracking software documentation
- performing software builds
- managing and tracking software releases and their delivery

The tools used in these areas can also provide measurements for process improvement. Royce [Roy98] describes seven core measures of value in managing software engineering 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.

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

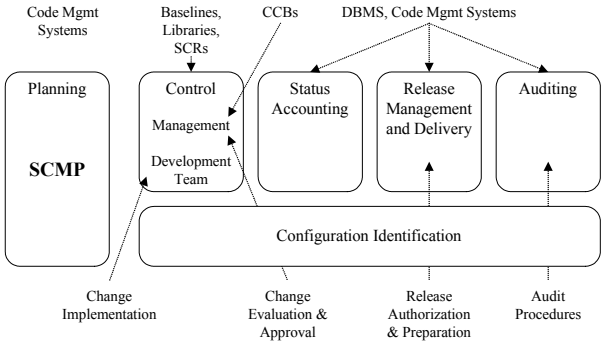

Figure 3 Characterization of SCM Tools and related procedures

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.

In planning, the software engineer picks SCM tools fit for the job. 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 [Dar90:c3,AppA], and a case study on selecting an SCM system is given in [Mid97]. Complementary information on SCM tools can be found in the Software Engineering Tools and Methods KA.

1.3.4. Vendor/Subcontractor Control
[Ber92:c13; Buc96:c11; IEEE828-98:c4s3.6]

A software project might acquire or make use of purchased software products, such as compilers or other tools. SCM planning considers if and how these items will be taken under configuration control (for example, 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.

1.3.5. Interface control
[IEEE828-98:c4s3.5]

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 [Ber92:c12].

1.4. SCM Plan [Ber92:c7; Buc96:c3; Pau93:L2-81]

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

Guidance on 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 [IEEE828-98:c4]. 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:

- Introduction (purpose, scope, terms used)
- SCM Management (organization, responsibilities, authorities, applicable policies, directives, and procedures)
- SCM Activities (configuration identification, configuration control, and so on)
- SCM Schedules (coordination with other project activities)
- SCM Resources (tools, physical resources, and human resources)
- SCMP Maintenance

1.5. Surveillance of Software Configuration Management
[Pau93:L2-87]

After the SCM process has been implemented, some degree of surveillance may be necessary to ensure that the provisions of the SCMP are properly carried out (see, for example [Buc96]). 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 those with the assigned responsibility perform the defined SCM tasks correctly. The software quality assurance authority, as part of a compliance auditing activity, might also perform this surveillance.

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

1.5.1. SCM measures and measurement
[Buc96:c3; Roy98]

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. Measurements of SCM processes 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.

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 time required to accomplish 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. Discussion of process and product measurement is presented in the Software Engineering Process KA. The software measurement program is described in the Software Engineering Management KA.

1.5.2. In-process audits of SCM
[Buc96:c15]

Audits can be carried out during the software engineering 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 function. See also subarea 5 *Software Configuration Auditing*.

2. Software Configuration Identification
[IEEE12207.0-96:c6s2.2]

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.

2.1. Identifying Items to Be Controlled
[Ber92:c8; IEEE828-98:c4s3.1; Pau93:L2-83; Som05:c29]

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.

2.1.1. Software configuration
[Buc96:c4; c6, Pre04:c27]

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 (IEEE610.12-90). It can be viewed as a part of an overall system configuration.

2.1.2. Software configuration item
[Buc96:c4;c6; Con98:c2; Pre04:c27]

A software configuration item (SCI) is an aggregation of software designated for configuration management and is treated as a single entity in the SCM process (IEEE610.12-90). A variety of items, in addition to the code itself, is 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 in which a balance must be achieved 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 [Ber92].

2.1.3. Software configuration item relationships
[Con98:c2; Pre04:c27]

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. 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.

2.1.4. Software version
[Bab86:c2]

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. [Con98:c3-c5] 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.

2.1.5. Baseline
[Bab86:c5; Buc96:c4; Pre04:c27]

A software baseline is a set of software configuration 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 configuration item that has been agreed on. 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 (see, for example, [Ber92]). 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 with other organizations (for example, 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.

2.1.6. Acquiring software configuration items
[Buc96:c4]

Software configuration items are placed under SCM control at different times; that is, 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 2 characterizes the growth of baselined items as the life cycle proceeds. This figure is based on the 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 topic 3.1 *Requesting, Evaluating, and Approving Software Changes*.

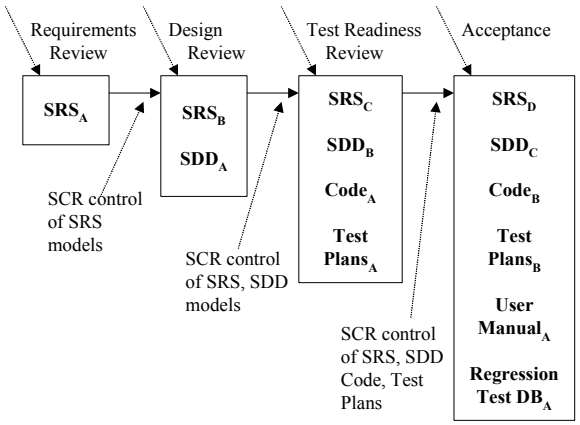

Figure 4 Acquisition of items

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 approval, the item is incorporated into the software baseline according to the appropriate procedure.

2.2. Software Library
[Bab86:c2; c5; Buc96:c4; IEEE828- 98:c4s3.1; Pau93:L2-82; Som01:c29]

A software library is a controlled collection of software and related documentation designed to aid in software development, use, and maintenance (IEEE610.12-90). It is also instrumental in software release management 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, while 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 [Ber92:c14].

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.

3. Software Configuration Control

[IEEE12207.0-96:c6s2.3; Pau93:L2-84]

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 from project requirements, as well as waivers of them. Information derived from these activities is useful in measuring change traffic and breakage, and aspects of rework.

3.1. Requesting, Evaluating, and Approving Software Changes
[IEEE828-98:c4s3.2; Pre04:c27; Som05:c29]

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 (for example, defect or enhancement) is usually recorded on the SCR.

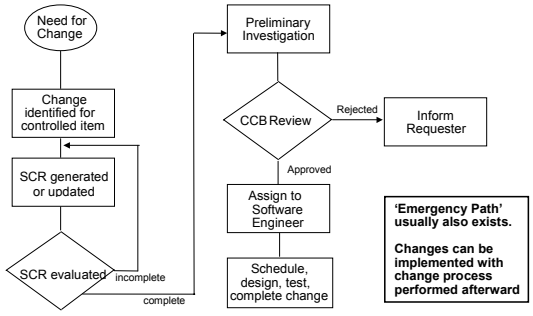

Figure 5 Flow of a Change Control Process

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 the 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.

3.1.1. Software Configuration Control Board
[Ber92:c9; Buc96:c9,c11; Pre04:c27]

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 may actually reside with the 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 (for example, 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 software quality audit or review.

3.1.2. Software change request process
[Buc96:c9,c11; Pre04:c27]

An effective software change request (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, for example [Ber92:c9].

3.2. Implementing Software Changes
[Bab86:c6; Ber92:c9; Buc96:c9,c11; IEEE828-98:c4s3.2.4; Pre04:c27; Som05:c29]

Approved SCRs are implemented using the defined software procedures in accordance with the applicable schedule requirements. Since a number of approved SCRs might be implemented simultaneously, it is necessary to provide a means for tracking which SCRs are incorporated into particular software versions and baselines. As part of the closure of the change process, completed changes may undergo configuration audits and software quality verification. This includes ensuring that only approved changes have been 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, which 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 the control of an independent SCM group. They may also appear as an integrated part of the software engineering environment. Finally, they may be as elementary as a rudimentary change control system provided with an operating system.

3.3. Deviations and Waivers
[Ber92:c9; Buc96:c12]

The constraints imposed on a software engineering effort or the specifications produced during the development activities might contain provisions which 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 from, or waivers of, the provisions.

4. Software Configuration Status Accounting
[IEEE12207.0-96:c6s2.4; Pau93:L2-85; Pre04:c27; Som05:c29]

Software configuration status accounting (SCSA) is the recording and reporting of information needed for effective management of the software configuration.

4.1. Software Configuration Status Information
[Buc96:c13; IEEE828-98:c4s3.3]

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 [Ber92:c10].

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

4.2. Software Configuration Status Reporting
[Ber92:c10; Buc96:c13]

Reported information can be used by various organizational and project elements, including the development team, the maintenance team, project management, and software quality activities. Reporting can take the form of ad hoc queries to answer specific questions or the periodic production of predesigned 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 the 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.

5. Software Configuration Auditing

[IEEE828-98:c4s3.4; IEEE12207.0-96:c6s2.5; Pau93:L2-86; Pre04:c26c27]

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 (IEEE1028-97). 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 [Ber92:c11; Buc96:c15] and (IEEE1028-97).

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 (for example, 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 [Buc96:c15] contrasts the purposes of the FCA and PCA in hardware versus software contexts, and recommends careful evaluation of the need for a software FCA and PCA before performing them.

5.1. 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.

5.2. Software Physical Configuration Audit

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

5.3. 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 is consistent with specifications or to ensure that evolving documentation continues to be consistent with the developing baseline item.

6. Software Release Management and Delivery

[IEEE12207.0-96:c6s2.6]

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.

6.1. Software Building

[Bab86:c6; Som05:c29]

Software building is the activity of combining the correct versions of software configuration 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 program 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, maintenance, 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 configuration 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 engineering environments provide this capability. These tools vary in complexity from requiring the software 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 software quality verification. Outputs of the build process might be needed for future reference and may become quality assurance records.

6.2. Software Release Management

[Som05:c29]

Software release management encompasses the identification, packaging, and delivery of the elements of a product, for example, executable program, documentation, release notes, and configuration data. Given that product changes can occur on a continuing basis, one concern 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. (Som01) The packaging task must identify which product items are to be delivered, and then select the

correct variants of those items, given the intended application of the product. The information documenting the physical contents of a release is known as a version description document. The release notes typically describe new capabilities, known problems, and platform requirements necessary for proper product operation. The package to be released also contains installation 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.

MATRIX OF TOPICS VS. REFERENCE MATERIAL

		[Bab86]	[Ber92]	[Buc96]	[Con98]	[Dar90]	[IEEE828-98]	[IEEE12207.0-96]	[Mid97]	[Moo98]	[Pau93]	[Pre04]	[Roy98]	[Som05]
1.	**Management of the SCM Process**													
1.1	*Organizational Context for SCM*		c4	c2		c2	c4s2.1							
1.2	*Constraints and Guidance for SCM Process*		c5				c4s1, c4s2.3			*				
1.3	*Planning for SCM*					c2		6.2.1						c29
	SCM organization and responsibilities		c7	c3			c4s2							
	SCM resources and schedules		c7	c3			c4s4, c4s5							
	Tool selection and implementation		c15		c6	c3, App A			*				*	
	Vendor/subcontractor control		c13	c11			c4s3.6							
	Interface control		c12				c4s3.5							
1.4	*SCM Plan*		c7	c3			c4				L2-81			
1.5	*Surveillance of SCM*			*			c4				L2-87			
	SCM measures and measurement			c3									188-202, 283-298	
	In-process audits of SCM			c15										
2.	**Software Configuration Identification**							c6s2.2						
2.1	*Identifying Items to be Controlled*		c8				c4s3.1				L2-83			c29
	Software configuration			c4,c6								c27		
	Software configuration item		*	c4,c6	c2							c27		
	Software configuration item relationships				c2							c27		
	Software versions	c2										c27		
	Baseline	c5	*	c4								c27		
	Acquiring software configuration items			c4										
2.2	*Software Library*	c2,c5	c14	c4			c4s3.1				L2-82			c29
3.	**Software Configuration Control**							c6s2.3			L2-84			
3.1	*Requesting, Evaluating and Approving Software Changes*						c4s3.2					c27		c29
	Software configuration control board		c9	c9,c11								c27		
	Software change request process		c9	c9,c11								c27		
3.2	*Implementing Software Changes*	c6	c9	c9,c11			c4s3.2.4					c27		c29
3.3	*Deviations and Waivers*		c9	c12										
4.	**Software Configuration Status Accounting**							c6s2.4			L2-85	c27		c29
4.1	*Software Configuration Status Information*		c10	c13			c4s3.3							
4.2	*Software Configuration Status Reporting*		c10	c13										
5.	**Software Configuration Auditing**		c11	c15			c4s3.4	c6s2.5			L2-86	c26, c27		
5.1	*Software Functional Configuration Audit*													
5.2	*Software Physical Configuration Audit*													
5.3	*In-Process Audits of a Software Baseline*													
6.	**Software Release Management and Delivery**							c6s2.6						
6.1	*Software Building*	c6												c29
6.2	*Software Release Management*													c29

RECOMMENDED REFERENCES FOR SCM

[Bab86] W.A. Babich, *Software Configuration Management, Coordination for Team Productivity*, Addison-Wesley, 1986.

[Ber92] H.R. Berlack, *Software Configuration Management*, John Wiley & Sons, 1992.

[Buc96] F.J. Buckley, *Implementing Configuration Management: Hardware, Software, and Firmware*, second ed., IEEE Computer Society Press, 1996.

[Con98] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys*, vol. 30, iss. 2, June 1998.

[Dar90] S.A. Dart, *Spectrum of Functionality in Configuration Management Systems*, Software Engineering Institute, Carnegie Mellon University, 1990.

[IEEE828-98] IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*, IEEE, 1998.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

[Mid97] A.K. Midha, "Software Configuration Management for the 21st Century," *Bell Labs Technical Journal*, vol. 2, iss. 1, Winter 1997, pp. 154-165.

[Moo98] J.W. Moore, *Software Engineering Standards: A User's Roadmap*, IEEE Computer Society, 1998.

[Pau93] 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.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, Sixth ed, McGraw-Hill, 2004.

[Roy98] W. Royce, *Software Project Management, A United Framework*, Addison-Wesley, 1998.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

APPENDIX A. LIST OF FURTHER READINGS

(Bab86) W.A. Babich, *Software Configuration Management, Coordination for Team Productivity*, Addison-Wesley, 1986.

(Ber92) H.R. Berlack, *Software Configuration Management*, John Wiley & Sons, 1992.

(Ber97) E.H. Bersoff, "Elements of Software Configuration Management," in *Software Engineering*, M. Dorfman and R.H. Thayer, eds., IEEE Computer Society Press, 1997.

(Buc96) F.J. Buckley, *Implementing Configuration Management: Hardware, Software, and Firmware*, second ed., IEEE Computer Society Press, 1996.

(ElE98) K. El-Emam et al., "SPICE, The Theory and Practice of Software Process Improvement and Capability Determination," presented at IEEE Computer Society, 1998.

(Est95) J. Estublier, "Software Configuration Management," presented at ICSE SCM-4 and SCM-5 Workshops, Berlin, 1995.

(Gra92) R.B. Grady, *Practical Software Metrics for Project Management and Process Management*, Prentice Hall, 1992.

(Hoe02) A. v. d. Hoek, "Configuration Management Yellow Pages," 2002, available at http://www.cmtoday.com/yp/configuration_management.html.

(Hum89) W. Humphrey, *Managing the Software Process*, Addison-Wesley, 1989.

(Pau95) M.C. Paulk et al., *The Capability Maturity Model, Guidelines for Improving the Software Process*, Addison-Wesley, 1995.

(Som01a) I. Sommerville, "Software Configuration Management," presented at ICSE SCM-6 Workshop, Berlin, 2001.

(USNRC1.169-97) USNRC Regulatory Guide 1.169, "Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants," presented at U.S. Nuclear Regulatory Commission, Washington, D.C., 1997.

(Vin88) J. Vincent, A. Waters, and J. Sinclair, *Software Quality Assurance: Practice and Implementation*, Prentice Hall, 1988.

(Whi91) D. Whitgift, *Methods and Tools for Software Configuration Management*, John Wiley & Sons, 1991.

APPENDIX B. LIST OF STANDARDS

(IEEE730-02) IEEE Std 730-2002, *IEEE Standard for Software Quality Assurance Plans*, IEEE, 2002.

(IEEE828-98) IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*, IEEE, 1998.

(IEEE1028-97) IEEE Std 1028-1997 (R2002), *IEEE Standard for Software Reviews*, IEEE, 1997.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

(IEEE12207.1-96) IEEE/EIA 12207.1-1996, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes - Life Cycle Data*, IEEE, 1996.

(IEEE12207.2-97) IEEE/EIA 12207.2-1997, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes - Implementation Considerations*, IEEE, 1997.

(ISO15846-98) ISO/IEC TR 15846:1998, *Information Technology - Software Life Cycle Processes - Configuration Management*, ISO and IEC, 1998.