

AS 4043—1992

IEEE 1042:1987

Reconfirmed 2014

Australian Standard[®]

**Software configuration
management**

This Australian Standard was prepared by Committee IT/15, Software Engineering. It was approved on behalf of the Council of Standards Australia on 22 May 1992 and published on 14 September 1992.

The following interests are represented on Committee IT/15:

Australian Bankers Association
Australian Computer Society
Australian Computer Society National Software Industry Committee
Australian Information Industry Association
Australian Software Metrics Association
Computer Aided Software Engineering Special Interest Group
Department of Defence
Griffith University
Institute of Quality Assurance
Software Quality Association, Qld
Software Verification Research Centre
Telecom Australia
University of New South Wales

Review of Australian Standards. To keep abreast of progress in industry, Australian Standards are subject to periodic review and are kept up to date by the issue of amendments or new editions as necessary. It is important therefore that Standards users ensure that they are in possession of the latest edition, and any amendments thereto.

Full details of all Australian Standards and related publications will be found in the Standards Australia Catalogue of Publications; this information is supplemented each month by the magazine 'The Australian Standard', which subscribing members receive, and which gives details of new publications, new editions and amendments, and of withdrawn Standards.

Suggestions for improvements to Australian Standards, addressed to the head office of Standards Australia, are welcomed. Notification of any inaccuracy or ambiguity found in an Australian Standard should be made without delay in order that the matter may be investigated and appropriate action taken.

STANDARDS AUSTRALIA

RECONFIRMATION

OF

AS 4043—1992

Software configuration management

RECONFIRMATION NOTICE

Technical Committee IT-015 has reviewed the content of this publication and in accordance with Standards Australia procedures for reconfirmation, it has been determined that the publication is still valid and does not require change.

Certain documents referenced in the publication may have been amended since the original date of publication. Users are advised to ensure that they are using the latest versions of such documents as appropriate, unless advised otherwise in this Reconfirmation Notice.

Approved for reconfirmation in accordance with Standards Australia procedures for reconfirmation on 17 January 2014.

The following are represented on Technical Committee IT-015:

Australian Computer Society
Australian Society for Technical Communication (NSW)
Charles Sturt University
Department of Defence (Australia)
Griffith University
La Trobe University
National Association of Testing Authorities Australia
National ICT Australia
NSW Business Chamber
Quantitative Enterprise Software Performance
Systems Engineering Society of Australia
University of Technology, Sydney

NOTES

Australian Standard[®]

**Software configuration
management**

First published as AS 4043—1992.

PUBLISHED BY STANDARDS AUSTRALIA
(STANDARDS ASSOCIATION OF AUSTRALIA)
1 THE CRESCENT, HOMEBUSH, NSW 2140

ISBN 0 7262 7599 9

 **Documents**
Document Sharing Hub

PREFACE

This Standard was prepared by the Standards Australia Committee on Software Engineering. It is identical with and has been reproduced from IEEE Std 1042, *IEEE Guide to Standard for Software Configuration Management*.

Under arrangements made between Standards Australia and the international Standards bodies, ISO and IEC, as well as certain other Standards organizations, users of this Australian Standard are advised of the following:

Copyright of the content of this Standard remains the property of IEEE. The copyright of this edition is vested in Standards Australia.

For the purpose of this Australian Standard, the IEEE text should be modified as follows:

References The references to ANSI/IEEE Standards should be replaced by references to Australian Standards as follows:

Reference to ANSI/IEEE Standard	Australian Standard
IEEE	AS
729 IEEE Standard Glossary of Software Engineering Terminology	—
828 Software Configuration Management Plans	4042 Software configuration management plans

© Copyright — STANDARDS AUSTRALIA

Users of Standards are reminded that copyright subsists in all Standards Australia publications and software. Except where the Copyright Act allows and except where provided for below no publications or software produced by Standards Australia may be reproduced, stored in a retrieval system in any form or transmitted by any means without prior permission in writing from Standards Australia. Permission may be conditional on an appropriate royalty payment. Requests for permission and information on commercial software royalties should be directed to the head office of Standards Australia.

Standards Australia will permit up to 10 percent of the technical content pages of a Standard to be copied for use exclusively in-house by purchasers of the Standard without payment of a royalty or advice to Standards Australia.

Standards Australia will also permit the inclusion of its copyright material in computer software programs for no royalty payment provided such programs are used exclusively in-house by the creators of the programs.

Care should be taken to ensure that material used is from the current edition of the Standard and that it is updated whenever the Standard is amended or revised. The number and date of the Standard should therefore be clearly identified.

The use of material in print form or in computer software programs to be used commercially, with or without payment, or in commercial contracts is subject to the payment of a royalty. This policy may be varied by Standards Australia at any time.

CONTENTS

SECTION	Page
1. Introduction	5
1.1 Scope	5
1.2 References	6
1.3 Mnemonics	6
1.4 Terms	6
2. SCM Disciplines in Software Management	7
2.1 The Context of SCM	7
2.1.1 SCM is a Service Function	7
2.1.2 SCM is a Part of the Engineering Process	7
2.1.3 SCM Manages all Software Entities	8
2.2 The Process of SCM	10
2.2.1 Management Environment of SCM	10
2.2.2 Dynamics of SCM	10
2.2.3 Role of Source Code in SCM	11
2.2.4 Different Levels of Control	11
2.3 The Implementation of SCM	11
2.3.1 Using Software Libraries	11
2.3.2 Controlling Changes to Libraries	12
2.3.3 Using Configuration Control Boards	13
2.4 The Tools of SCM	13
2.4.1 Basic Tool Set	13
2.4.2 Advanced Tool Set	13
2.4.3 On-Line Tool Set	14
2.4.4 Integrated Tool Set	14
2.5 The Planning of SCM	14
3. Software Configuration Management Plans	15
3.1 Introduction	15
3.1.1 Purpose	15
3.1.2 Scope	15
3.1.3 Definitions	16
3.1.4 References	16
3.2 Management	17
3.2.1 Organization	17
3.2.2 SCM Responsibilities	18
3.2.3 Interface Control	18
3.2.4 SCM Plan Implementation	19
3.2.5 Applicable Policies, Directives, and Procedures	20
3.3 SCM Activities	21
3.3.1 Configuration Identification	21
3.3.2 Configuration Control	23
3.3.3 Configuration Status Accounting	28
3.3.4 Audits and Reviews	29
3.3.5 Release Process	30
3.4 Tools, Techniques and Methodologies	31
3.5 Supplier Control	32
3.5.1 Subcontractor Software	32
3.5.2 Vendor Software	33
3.6 Records Collection and Retention	34

FIGURES		Page
Fig 1	Model of Change Management	10
Fig 2	Three Types of Libraries	12
TABLES		
Table 1	Characteristics of Appendixes	5
Table 2	Hierarchy of Controlled Entities	9
Table 3	Levels of Control in Sample Plans	13
Table 4	Variable Levels of Control	24
APPENDIXES		
Appendix A	Software Configuration Management Plan for Critical Software for Embedded Systems	35
Fig 1	Program Organization Chart	39
Table 1	Responsibility Assignments	40
Table 2	Baseline Objectives	42
Attachment A	System/Software Change Request	46
Attachment B	Software Change Authorization	47
Attachment C	Fig 1 CSES Procedure for Creating Initial Baseline	48
Attachment D	Fig 1 CSES Procedures for Changes to Controlled Software/Documentation	49
	Fig 2 Program Organization Chart	50
Appendix B	Software Configuration Management Plan for Experimental Development Small System	51
Fig 1	Project Organization Chart	55
Attachment A	Software Promotion Requests	59
	Table 1 Data for Software Release	59
Attachment B	IEEE Guide for Processing System Software Change Requests	60
Attachment C	System/Software Change Request Form	61
	Table 1 SCR Data Elements	61
Appendix C	Software Configuration Management Plan for a Software Maintenance Organization	62
Fig 1	SPLIT Facility Organization	66
Fig 2	Structure of CCB	67
Table 1	Hierarchy of Elements	68
Table 2	Problem Criteria	69
Attachment A	System/Software Change Request (SPLIT Form C-1049)	73
	Table 1 Definitions of Elements in SCR	73
Attachment B	Software Change Authorization	74
	Table 1 Definitions of Elements in SCA	74
Appendix D	Software Configuration Management Plan for a Product Line System	75
Fig 1	PLAS Organization Chart	80
Table 1	Processing Approved Changes	85
Appendix E	References Bibliography	88

STANDARDS AUSTRALIA

Standards Australia

Software configuration management

1. Introduction

1.1 Scope. This guide describes the application of configuration management (CM) disciplines to the management of software engineering projects. Software configuration management (SCM) consists of two major aspects: planning and implementation. For those planning SCM activities, this guide provides insight into the various factors that must be considered.

Users implementing SCM disciplines will find suggestions and detailed examples of plans in this guide. This guide also presents an interpretation of how ANSI/IEEE Std 828-1983 [2]¹ can be used for planning the management of different kinds of computer program development and maintenance activities.

The guide is presented in two parts. The first part, the main body of the guide, presents issues to consider when planning software configuration management for a project or organization. The second part of the guide presents, for those preparing SCM Plans, a series of sample Plans illustrating different concepts discussed in the body of the guide.

The text of the guide introduces the essential concepts of SCM, particularly those of special significance (for example, libraries and tools) to software engineering. It then presents the planning

for SCM in terms of documenting a Plan following the outline of ANSI/IEEE Std 828-1983 [2] so that a user who is unfamiliar with the disciplines of software configuration management can gain some insight into the issues. For those preparing SCM Plans, the second part of the guide provides sample plans for consideration.

The sample SCM Plans include a variety of software configuration management applications for different types of projects and organizations. Appendix A illustrates a software configuration management plan (SCMP) for a project developing a complex, critical computer system. It describes a Plan for managing a typical software development cycle where the development is contracted to an organization that does not have responsibility for its maintenance or use. Appendix B illustrates a SCMP for a small software development project. It describes a Plan for supporting a prototype development activity where the goal of the project is to demonstrate the feasibility of a concept. Appendix C illustrates a SCMP used by an organization where the emphasis is on maintaining programs developed by other activities or organizations. Appendix D illustrates a SCMP for an organization developing and maintaining computer programs embedded in a hardware product line. It describes a Plan for managing both software development and maintenance of a commercial product line. Some of the different characteristics illustrated are shown in Table 1.

¹The numbers in brackets correspond with those of the references in 1.2.

Table 1
Characteristics of Appendixes*

Appendix Number	Emphasis of Control (Life Cycle Phase)	Type of Project	Relative Size (Dollar/Manhour)	SCM Tools Available	Life Span of Plan	Writing for Plan
1	Development	Critical	Medium	Advanced	Short	Highly structured
2	Concept	Prototype	Small	Basic	Short	Informal
3	Operations	Support sw	Large	On-line	Full life cycle	Structured
4	All	Commercial	Small	Integrated	Full life cycle	Organizational Informal

*NOTE: The purpose of the Appendixes is not to provide an illustration for every possible combination of project characteristics but rather to show that the ANSI/IEEE Std 828-1983 [2] can be applied to a wide variety of projects.

1.2 References. This guide shall be used in conjunction with the following publications:

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

[2] ANSI/IEEE Std 828-1983, IEEE Standard for Software Configuration Management Plans.

Additional references useful in understanding software configuration management are given in Appendix E.

1.3 Mnemonics. The following acronyms are used in the text of this guide:

CCB	Configuration Control Board
CDR	Critical Design Review
CI	Configuration Item
CM	Configuration Management
CPC	Computer Program Component
CPCI	Computer Program Configuration Item
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
[EP]ROM	[Electrically Programmable] Read Only Memory
FCA	Functional Configuration Audit
OEM	Original Equipment Manufacturer
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
RAM	Random Access Memory
ROM	Read Only Memory
SCA	System/Software Change Authorization
SCCB	Software Configuration Control Board
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	System/Software Change Request
SQA	Software Quality Assurance
VDD	Version Description Document

1.4 Terms. Some terms used in SCM circles have restricted meanings or are not defined in the guide. General statements of the contextual meanings are given to aid in understanding the concepts in the guide. These are not formal definitions, subject to review and approval as in a standard, but contextual definitions serving to

augment the understanding of configuration management activities as described within this guide.

As used here, the term **baseline**³ represents the assignment of a documented identifier to each software product configuration item (CI) and associated entities. That is, the source code, relocatable code, executable code, files controlling the process of generating executable code from source code, documentation, and tools used to support development or maintenance of the software product should all be captured, labeled and somehow denoted or recorded as parts of the same baseline. As computer programs move from an initial idea to the maintenance phase, it is common for a series of developmental baselines of increasing complexity to be established during the various internal and external reviews conducted by management (and customers) to determine progress and technical suitability. The baseline concept is as useful to engineering during development as it is after release for use and maintenance.

The various SCM functions are dependent on the baseline concept. Several valuable uses of the baseline concept include

- (1) To distinguish between different internal releases for delivery to a customer (that is, successive variants of the same product baseline)
- (2) To help to ensure complete and up-to-date technical product documentation
- (3) To enforce standards (SQA)
- (4) To be used as a means of promoting (that is, internally releasing) each CI from one phase of development or test to another
- (5) To identify customer involvement in internal (developmental) baselines

Since SCM disciplines are an integral part of the engineering process they guide the management of internal developmental baselines as well as the more formal functional, allocated, and product baselines. The SCM disciplines, as applied to developmental baselines, are used (implicitly or explicitly) to coordinate most engineering activities that occur within the context of each baseline. Varying levels of formality provide flexibility and responsiveness to the engineering process, yet maintain the benefits of recognizing SCM disciplines.

³A specification or product that has been formally reviewed and agreed to by responsible management, that thereafter serves as the basis for further development, and can be changed only through formal change control procedures.

The term **promotion** is used here to indicate a transition in the level of authority needed to approve changes to a controlled entity, such as a baseline CI.

Promotions typically signify a change in a CI's internal development state. The term **release** is used to designate certain promotions of CI that are distributed outside the development organization.

In general, as the development process continues, there are more constraints imposed on the change process (coordination with interfacing hardware, user's adaptations, etc) and correspondingly higher levels of authority are needed for approving the changes. When an entity is finally released as a formal baseline, a high level of authority is needed to approve changes. When internal or developmental baselines are created as a part of the engineering process and entities are moved or released to another internal activity for additional work, integration, or testing the term **promotion** is used to distinguish this type of release from the more formal releases to users.

Promotion from one developmental baseline to another represents the visibility granted to some organizations for a given baseline. As developmental baselines are promoted within an organization, they tend to become more stable. The more stable a baseline is, the higher the level of visibility it is granted.

The term **version** is used here to indicate a software CI having a defined set of functional capabilities. As functional capabilities are added, modified, or deleted the CI is given a different version identifier. It is common and recommended practice to use a configuration identification scheme that permits easy and automatic identification of particular version labels.

The term **revision** is commonly associated with the notion of *bug fixing*, that is, making changes to a program that corrects only errors in the design logic but does not affect documented functional capabilities since none of the requirements have changed. The configuration identification scheme must provide for clear identification of revisions and versions of each specific promotion and release.

2. SCM Disciplines in Software Management

2.1 The Context of SCM. This guide discusses SCM as a set of management disciplines within

the context of the software engineering process rather than as a set of specific activities performed, or as functions within an organization. The reason for this approach is that software CM, as contrasted with hardware CM, tends to be more deeply involved in the software engineering process and, while the same general CM functions are performed, the disciplines are extended to include the process of developing a baseline.

Software CM and release processing are performed within the context of several basic CM functions: configuration identification, baseline management, change control and library control, status accounting, reviews and audits, and release processing. In practice, the ways in which these functions are performed are different for the different kinds of programs being developed (commercial, embedded, OEM, etc), and may vary in the degree of formal documentation required within and across different life-cycle management phases (research phase, product development, operations, and maintenance).

Software CM also provides a common point of integration for all planning, oversight and implementation activities for a project or product line. These functions are performed within the context of a project — providing the framework (labeling and identification) for interfacing different activities and defining the mechanisms (change controls) necessary for coordinating parallel activities of different groups. SCM provides a framework for controlling computer program interfaces with their underlying support hardware, coordinating software changes when both hardware and software may be evolving during development or maintenance activities.

Finally, SCM is practiced within the context of management, providing management with the visibility (through status accounting and audits) of the evolving computer products that it needs to perform effectively.

2.1.1 SCM is a Service Function. Software CM is a support activity that makes technical and managerial activities more effective. Effectiveness of the SCM processes increases in proportion to the degree that its disciplines are an explicit part of the normal day-to-day activities of everyone involved in the development and maintenance efforts, (as opposed to a separate SCM organization or activity). This holds true whether SCM is administered by a separate SCM group, distributed among many projects, or a mixture of both.

2.1.2 SCM is a Part of the Engineering Process. The disciplines of SCM apply to the development of programmed logic, regardless of the

form of packaging used for the application. Software engineering technology is effectively used in the generation of stored programmed logic when the complexity of the function is large. SCM disciplines assist in the identification and evolution of changes during the engineering process, even though the final package may be ROM, and managed as a hardware configuration item.

Configuration management is practiced in one form or another as a part of every software engineering activity where several individuals or organizations have to coordinate their activities. Although the basic disciplines of configuration management are common to both hardware and software engineering activities, there are some differences in emphasis due to the nature of the design activity. Software products (as compared to hardware products) are easy to change⁴ (little if any lead time is needed for parts procurement and production setup).

Software CM is a discipline for managing the evolution of computer program products, both during the initial stages of development and during all stages of maintenance. The designs of programs are not easily partitioned into independent tasks due to their complexity. Therefore, configuration management disciplines are more valuable during the design (and redesign during maintenance) phases. This is when using techniques of multiple levels of baselines and internal releases (or promotions) to a larger degree than is typically practiced by hardware CM really pays off.

Whether software is released for general use as programs in RAM or embedded in ROM, it is a form of logic. Therefore, SCM disciplines can and should be extended to include development of the computer programs' component parts (for example, source code and executable code) whereas hardware CM focuses mainly on the management of documentation.

The differences between hardware and software CM, of importance to software CM, include

- (1) Software CM disciplines are used to simultaneously coordinate configurations of many different representations of the soft-

ware product (source code, relocatable code and executable code) rather than just their *documentation*. The nature of computer programs requires this extension and the SCM disciplines and related SCM support software adapt readily to this task.

- (2) The use of interactive software development environments extends the concepts of software CM to managing evolutionary changes that occur during the routine generation of specifications, designs, and implementation of code, as well as to the more rigidly documented and controlled baselines defined during development and system maintenance.
- (3) Software development environments are rapidly becoming automated with interactive tool sets. This modifies many of the traditional methods used in hardware CM but the fundamental concepts of CM still apply.

2.1.3 SCM Manages all Software Entities.

Software CM extends the management disciplines of hardware CM to include all of the entities of the product as well as their various representations in documentation. Examples of entities managed in the software engineering process include

- (1) Management plans
- (2) Specifications (requirements, design)
- (3) User documentation
- (4) Test design, case and procedure specifications
- (5) Test data and test generation procedures
- (6) Support software
- (7) Data dictionaries and various cross-references
- (8) Source code (on machine-readable media)
- (9) Executable code (the run-time system)
- (10) Libraries
- (11) Data bases:
 - (a) Data which are processed,
 - (b) Data which are part of a program
- (12) Maintenance documentation (listings, detail design descriptions, etc)

All supporting software used in development, even though not a part of the product, should also be controlled by configuration management disciplines.

Not all entities are subject to the same SCM disciplines at the same time. When the software product is under development, the documentation entities (baselined specifications and user requirements) are the most important. When coding begins, the documentation representing the design is the most important entity to be

⁴Even what is traditionally thought of as *hard* software—that is, firmware, is becoming easier to modify. An example is card edge programming where the programs in a ROM are easily modified, though not under program control during execution.

NOTE: While the time to change a design may be the same for hardware engineering as for software engineering, implementation and installation time is greater and consequently more expensive for hardware configuration items.

managed. Finally, when the product is ready for general use, the source code is the most accurate representation of the real product and the documentation is related so that representation is most important. These transitions of disciplinary focus over time are common to all SCM disciplines and need to be recognized in planning systems for effectively supporting project management.

Firmware⁵ raises some special considerations for configuration management. While being developed, the disciplines of software CM apply; but when made a part of the hardware (*burned* into [EP]ROM), the disciplines of hardware CM apply. Testing may vary but the SCM requirements are generally the same. The packaging of [EP]ROM versus RAM code also introduces and necessitates different identification procedures, which are noted in 3.3.1.

2.1.3.2 The issue of what entities are to be managed often arises in the practical context of what gets captured in each library, and when. Consideration need also be given to the hierarchy of entities managed during this process. There are several different ways of looking at this hierarchy of entities; one, for example, is a three-level hierarchy:

- (1) Configuration item (CSCI, CPCI, System, System Segment, Program package, module)
- (2) Component (CPC, CSC, Subsystem, Unit, Package, Program function)
- (3) Unit (Procedure, function Routine, Module)

The configuration control boards (CCB) that are oriented to business type management decisions usually select one level in this hierarchy as the level at which they will control changes. Other CCB may focus on more technical issues and would each select other levels, the module for example, as the control level for reviewing changes. See 2.2.5 for further discussion of control levels.

⁵**Firmware.** Computer programs and data loaded in a class of memory that cannot be dynamically modified by the computer during processing. Used here to generically refer to any programmed code implemented in nonvolatile memory such as [EP]ROM, regardless of its function; contrasts with code designed to execute out of volatile memory, such as RAM. There are differences between software intensive firmware and hardware intensive firmware. The key is ease of adaptability or degree with which programmed instructions are used, and the size of the program. Software intensive firmware denotes an activity that has available a set of tools commonly used in software engineering. Hardware intensive firmware denotes a development activity that has available a minimum of tools necessary for creation (*burn in*) of the firmware.

Another way of looking at entities to be managed is in terms of the interrelationships between the computer programs being developed and the other software entities used during development and testing of that program. This hierarchy is illustrated in Table 2.

Table 2
Hierarchy of Controlled Entities

Entity	Layer
Released entities	Product Layer
Promoted entities	Test layer
Modifiable unique entities and support software	Invocation layers
Product development environment	Support software layer
Operating system	Run-time software layer

The SCM process should support each of these layers.

2.1.3.3 Still another way of viewing the entities is in terms of the intermediate products generated in the process of building the computer program product. Each of these intermediate products may be viewed as:

- (1) *Modifiable entities.* These items are the individually modifiable units that are required to produce the deliverable entities. They are the source code, control files, data descriptions, test data, documents, etc, that constitute the focus of SCM. The entities at this level are referenced as *units* or *components* in this guide.
- (2) *The compilation or assembles entities, such as compilers.* These are needed to develop, test and maintain the program throughout the life cycle of the product. These entities are referenced as *support software* in this guide.
- (3) *Application-specific entities.* These are the different representations that are created in the process of producing the deliverables. Examples are the results produced by the compilation and assembly entities, and link/load entities, such as a link editor/locator. These culminate in the product that is released for general use. These entities are referenced as *configurations items* (CI) in this guide.

2.2 The Process of SCM

2.2.1 Management Environment of SCM.

Software engineering, and therefore SCM, takes place within an organizational business environment. To be effective, SCM must blend in with and reflect the organization. It must take into account the management style—entrepreneurial, very disciplined, etc. The technical skills of the implementing organization must be taken into account as well as available resources when specifying whether SCM is to be performed by a single organization or distributed among several. The organization must also be responsive to the kinds of controls needed by the organization that will ultimately be using the product.

SCM management provides support to the organization by working within it to define implement policies, techniques, standards, and tools that facilitate their control of the SCM process. These processes assist other managers (and customers as required) by supporting effective configuration identification, change controls, status accounting, audits, and reviews.

2.2.2 Dynamics of SCM. The cornerstone activity of SCM is managing the change process and tracking changes to ensure that the configuration of the computer program product is accurately known at any given time. The change management is accomplished by completely identifying each baseline and tracking all subsequent changes

made to that baseline. This process is used whether the baseline represents preliminary documentation, such as requirements, or a fully documented program including source and object code. All entities (specifications, documents, text data, and source code) are subject to this change management discipline.

Effectively managing baseline changes requires that a scheme for identifying the *structure* of the software product must be established. This structure relates to the hierarchical organization of the computer program and is extended to include all entities or work-products associated with the program. This identification scheme must normally be maintained throughout the full life of the computer program product. Usually a numbering scheme or file name scheme is associated with the structure, and unique and appropriate labels are assigned to each entity of the product.

As new baselines are created in transition by a promotion or release, the aggregate of entities is reviewed or audited to verify consistency with the old baseline, and the identification labeling is modified to reflect the new baseline. Changes to the different versions and revisions of each baseline are maintained. The history of changes to baselined configurations is maintained and made available to engineering and management in status reports. Figure 1 illustrates a model of the SCM process.

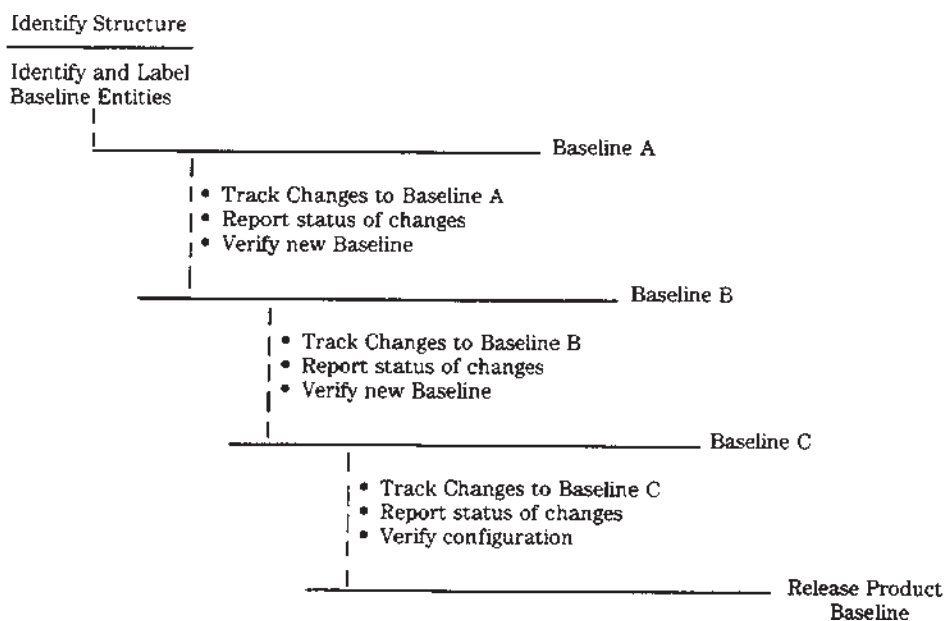


Fig 1
Model of Change Management

2.2.3 Role of Source Code in SCM. A key entity to be managed is the source code, since it is the basic representation in readable form of the product being controlled. Other forms of documentation and data are verified by comparison to this entity. At different phases in a development cycle, source code may not be available and different baselined entities may be defined as the basic representation. However, for most of the software life cycle, the source code provides the key entity for verification. The creation of executable code for the machine is directly derived (in the majority of computer systems) from the source code by various mechanized tools, such as assemblers, compilers, link/loaders, and interpreters. Recreation of source code (and object code) from design documentation can be costly. Therefore, to control only design documentation does not usually fully capture the implementation of the software. If the source code were to be lost because of improper, unreliable, or insufficient controls, the cost of recreating all of the source code would (in the majority of cases) be very expensive because of the typically incomplete state of the documentation.

Design documentation is verified against the product represented by the source code. The test entities (test design, test cases and procedures), test data (including data generation procedures) and test reports, are used to verify that the executable code (produced by the source code) matches the documentation. Documentation needed for maintenance (programming notebooks, etc) and user documentation are also verified against the source code.

Depending on the difficulty of rebuilding a complete set of executable code, the relocatable code may also be identified and considered an entity. However, the source code is generally considered to be the primary, if not sole source in establishing the product configuration.

Since source code can be interpreted differently by different compilers, it is necessary to control the versions of the support software used for a specific released product so as to have full control over the computer program product.

2.2.4 Different Levels of Control. Management delegates the authority for action downward to and including the work done by nonmanagement personnel. Management also selectively delegates aspects of control to nonmanagement personnel. In this guide, the term **levels of control** includes all control exercised by both management and nonmanagement. The term **authority** refers to control reserved by management for

management decisions relative to allocation of resources: schedule, production cost, customer requirements for product cost, performance, delivery, etc. Nonmanagement provides technical data to support these evaluations. Since the SCM Plan must identify all software CI (or classes thereof) that will be covered by the Plan, it must also define the level of management needed to authorize changes to each entity. As the software product evolves, it may be wise or necessary to increase the management authorization level (that is, level of control) needed. This can be accomplished through the internal part promotion hierarchy.

A general-use facility, which has many released software CI as well as CI under development, will often require many separate levels of control, and possibly different levels of authority for approving changes. For example, software CI that are used by several organizations may require change approval by management that is in charge of all those organizations. Not only the CI that will be delivered by the development group but also the level of authority for all vendor-supplied or internally developed software tools, utilities, operating systems, etc, used in the development need to be identified. Software CI used within any intermediate organization may usually require change approval by that organization's management. These intermediate organizations may have unique design or analysis tools for their own use on the project and can have change control authority over these tools.

2.3 The Implementation of SCM

2.3.1 Using Software Libraries. The techniques and methods used for implementing control and status reporting in SCM generally center around the operation of software libraries. Software libraries provide the means for identifying and labeling baselined entities, and for capturing and tracking the status of changes to those entities.

Libraries have been historically composed of documentation on hard copy and software on machine readable media, but the trend today is towards all information being created and maintained on machine-readable media.⁶ This trend, which encourages the increased use of automated tools, leads to higher productivity. The trend also

⁶There may still be valid *legal* needs for maintaining hard copy versions of all baselined materials. The ability to eliminate hard copy media should not be construed as a necessary or even wise thing for an organization to do.

means that the libraries are a part of the software engineering working environment. The SCM functions associated with the libraries have to become a part of the software engineering environment, making the process of configuration management more transparent to the software developers and maintainers.

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

The names of libraries may vary, but fundamentally three kinds should be considered, as outlined in Fig 2.

The **dynamic library**, sometimes called the *programmer's library*, is a library used for holding newly created or modified software entities (units/modules or data files and associated documentation). This is the library used by programmers in developing code. It is freely accessible to the programmer responsible for that unit at any time. It is the programmers' workspace and controlled by the programmers.

The **controlled library**, sometimes called the *master library*, is a library used for managing the current baseline(s) and for controlling changes made to them. This is the library where the units and components of a configuration item that have been promoted for integration are maintained.

Entry is controlled, usually after verification. Copies may be freely made for use by programmers and others. Changes to units or components in this library must be authorized by the responsible authority (which could be a configuration control board or other body with delegated authority).

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

2.3.2 Controlling Changes to Libraries. Several possible methods for controlling access to libraries are illustrated in the Appendixes. Appendix B prescribes formal change control of several configuration items at the component level within established baselines. Another approach is having rather informal methods for authorizing changes to configuration items. This method is used for fast integration of changes in a research type environment, as in Appendix B. For libraries having several configuration items including both external (third-party software) and internal (in-house developments) sources of supply, a mixture of formal methods for authorizing changes is applicable, as illustrated in Appendix C. Externally developed computer programs may be controlled at CI levels, whereas internally developed computer programs may be controlled at more discrete component levels. The procedures for authorizing changes may be integrated with the software tools in an integrated environment, as illustrated in Appendix D.

In summary, the levels of control described in each appendix are illustrated in Table 3.

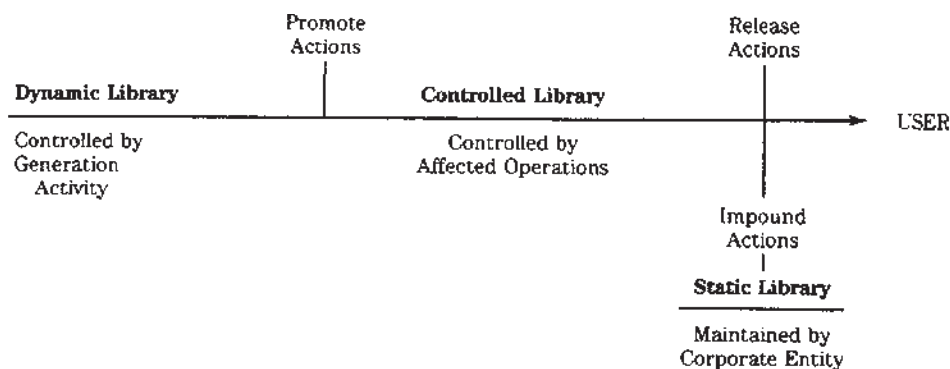


Fig 2
Three Types of Libraries

Table 3
Levels of Control in Sample Plans

	Appendix A	Appendix B	Appendix C	Appendix D
Number of CI	Several CI (internal)	3 CI (internal)	Internal CI External CI	2 CI (internal)
Components (CSC)	All components	NA	Internal components	Unit
Type of control	Formal	Informal	Formal	Formal (automated)

2.3.3 Using Configuration Control Boards.

Another functional concept of SCM is the extended use of configuration control boards (CCB). This concept provides for implementing change controls at optimum levels of authority. Configuration control boards can exist in a hierarchical fashion (for example, at the program, system design, and program product level), or one such board may be constituted with authority over all levels of the change process. In most projects, the CCB is composed of senior level managers. They include representatives from the major software, hardware, test, engineering, and support organizations. The purpose of the CCB is to control major issues such as schedule, function, and configuration of the system as a whole.

The more technical issues that do not relate to performance, cost, schedule, etc, are often assigned to a software configuration control board (SCCB). The SCCB discusses issues related to specific schedules for partial functions, interim delivery dates, common data structures, design changes and the like. This is the place for decision-making concerning the items that must be coordinated across CI but which do not require the attention of high level management. The SCCB members should be technically well-versed in the details of their area; the CCB members are more concerned with broad management issues facing the project as a whole and with customer issues.

2.4 The Tools of SCM. The SCM software tools selected for use by a project and described in a Plan need to be compatible with the software engineering environment in which the development or maintenance is to take place.

SCM tools are beginning to proliferate and choices have to be made as to the tool set most useful for supporting engineering and management. There are many different ways of examining available SCM tools. One way is to categorize them according to characteristics of their prod-

ucts: a filing system, a data-base management system, and an independent knowledge-based system.⁷ Another way is to examine the functions they perform: clerical support, testing and management support, and transformation support.⁸ A third way of categorizing the SCM tools is by how they are integrated into the software engineering environment on the project. The current set of available SCM tools is classed in terms of the level of automation they provide to the programming environment on a project.

2.4.1 Basic Tool Set. This set includes:

- (1) Basic data-base management systems
- (2) Report generators
- (3) Means for maintaining separate dynamic and controlled libraries
- (4) File system for managing the check-in and check-out of units, for controlling compilations, and capturing the resulting products

This set is compatible with a programming environment that is relatively unsophisticated.

The tools control the information on hard copy regarding a program product. They assume a capability for maintaining machine processable libraries that distinguish between controlled and uncontrolled units or components. The tools simplify and minimize the complexity, time, and methods needed to generate a given baseline. Appendix B illustrates a project using such a tool set.

2.4.2 Advanced Tool Set. This set includes:

- (1) Items in the basic tool set
- (2) Source code control programs that will maintain version and revision history
- (3) Compare programs for identifying (and helping verify) changes

⁷Reference: British Alvey Programme.

⁸Reference: *Life Cycle Support in the Ada⁹ Environment* by Mc Dermid and Ripken.

⁹Ada is a registered trademark of the US Government, AJPO.

- (4) Tools for building or generating executable code
- (5) A documentation system (word processing) to enter and maintain the specifications and associated user documentation files
- (6) A system/software change request/authorization (SCR/SCA) tracking system that makes requests for changes machine readable

This set provides a capability for a SCM group to perform more efficiently on larger, more complex software engineering efforts. It assumes a programming environment that has more computing resources available.

It provides the means of efficiently managing information about the units or components and associated data items. It also has rudimentary capabilities for managing the configurations of the product (building run-time programs from source code) and providing for more effective control of the libraries. Appendix A illustrates use of such a tool set.

2.4.3 On-Line Tool Set. This set includes:

- (1) Generic tools of the advanced tool set integrated so they work from a common data base
- (2) An SCR/SCA tracking and control system that brings generation, review, and approval of changes on-line
- (3) Report generators working on-line with the common data base, and an SCR/SCA tracking system that enables the SCM group to generate responses to on-line queries of a general nature

This set of tools requires an interactive programming environment available to the project. It also provides an organization with the minimal state-of-the-art SCM capabilities needed to support the typical interactive programming environment currently available in industry. It assumes on-line access to the programming data base and the resources necessary for using the tools. Appendix C illustrates use of such a SCM tool set.

2.4.4 Integrated Tool Set This set includes:

- (1) On-line SCM tools covering all functions
- (2) An integrated engineering data base with SCM commands built into the on-line engineering commands commonly used in designing and developing programs (most functions of CM are heavily used during design and development phases)
- (3) The integration of the SCM commands with on-line management commands for building and promoting units and components

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

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

SCM defines the interaction between a number of activities extending throughout the life cycle of the product. The SCM Plan functions as a centralized document for bringing together all these different points of view. The cover sheet of the Plan is usually approved by all of the persons with responsibilities identified in the Plan. This makes the Plan a living document, to be maintained by approved changes throughout the life of the computer programs.

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

It should be clear from the information given above, but it is stated explicitly here, that the application (and thus the planning) of SCM is very sensitive to the context of the project and the organization being served. If SCM is applied as a corporate policy, it must not be done blindly, but rather it should be done in such a way that the details of a particular SCM application are reexamined for each project (or phase for very

large projects). It must take into consideration the size, complexity, and criticality of the software system being managed; and the number of individuals, amount of personnel turnover, and organizational form and structure that have to interface during the life of the software system being managed.

This guide provides suggestions as to how ANSI/IEEE Std 828-1983 [2] can be interpreted for specific projects, and items to be considered in preparing a plan. The objective of the planner is to prepare a document that

- (1) Clearly states the actions to be performed by software engineering and supporting activities that are required to maintain visibility of the evolving configuration of the computer program products
- (2) Supports management in the process of evaluating and implementing changes to each configuration
- (3) Assures that the changes have been properly and completely incorporated into each computer program product.

3. Software Configuration Management Plaits

3.1 Introduction. Because SCM extends throughout the life cycle of the software product, the SCM Plan is the recommended focal point for integrating and maintaining the necessary details for software CM. Projects do differ in scope and complexity and a single format may not always be applicable. ANSI/IEEE Std 828-1983 [2] describes a minimum format for plans with a maximum amount of flexibility. If a section of the format is not applicable, the sentence *There is no pertinent information for this section* should be inserted to indicate that the section has not been overlooked.

It is desirable to provide a synopsis for users of the Software Configuration Management Plan and for the managers who must approve it. In each Appendix to this guide, a synopsis has been prepared to set the context surrounding the generation of the sample SCM Plan. For purposes of this guide, the viewpoint of each synopsis in the Appendixes is directed towards the user of the guide.

3.1.1 Purpose. The theme here is to inform the reader of the specific purpose of the SCM activity(ies) to be defined in the SCM Plan. It is sufficient to write a brief paragraph identifying

the system to which the particular SCM Plan applies, noting any dependencies on other SCM or CM Plans. For example, Appendix A emphasizes thoroughness of audits and reviews to assure conformance to contractual requirements for a computer program product; the theme is rigorous control of the configuration during development. Appendix B is directed towards low cost, quick response to changes, and documentation of the as-built versions of the computer programs. In Appendix C the theme is maintaining configuration control of many computer program products after development and while they are in use. This is complicated by the necessity to manage third-party software and subcontracted software along with internally developed software. Appendix D is directed towards the complex process of generating computer programs, and includes third-party software and subcontracted software in an environment where changes to configurations are driven by marketing, engineering, vendor changes, and customer demands, as well as the normal iteration of engineering changes.

3.1.2 Scope. The scope of the Plan encompasses the tasks of SCM. The function of the subsection is to

- (1) Identify the specific SCM concerns
- (2) Define what the Plan will and will not address
- (3) Identify the items to be managed.

3.1.2.1 It is also important to identify the

- (1) Lowest entity in the hierarchy (the control element) that will be reviewed by the top level project or system management CCB
- (2) Smallest useful entity that will be reviewed (a module, a unit, a line of code) by technical management (SCCB)
- (3) Deliverable entities or configuration item(s) to be released for use as separate entities

The definition and scope of each entity of the configuration item and the kind of control to be considered for each type of entity is also needed.

A short description of relationships among configuration items may be appropriate. The boundary of the SCM activities may be described here with the help of graphics (block diagrams, tables, engineering drawing) as necessary.

Issues to Consider in Planning Section 1.2—Scope

- (1) What are the characteristics of the configuration items to be controlled?

- (a) Only one application program¹⁰
- (b) Many separate small application programs
- (c) An integrated set of application and support programs embedded in a system
- (d) Computer programs as an integral part of a hardware system
- (2) What are the different high-level interfaces to be managed?
 - (a) People, organization interfaces
 - (b) Subcontractor interfaces
 - (c) Specification interfaces
 - (d) Contractor interfaces
 - (e) Hardware interfaces
 - (f) Life cycle phase interfaces
 - (g) Software interface
- (3) What are the time frames of the project
 - (a) Life cycle phases
 - (b) Calendar time
- (4) What resources will be available or required for the SCM activities?
 - (a) Machine resources
 - (b) Space resources
 - (c) People resources
 - (d) Schedule dependencies
- (5) What are the software engineering entities to be controlled?
 - (a) Contractual documents
 - (b) Specifications
 - (c) Other documentation
 - (d) Test procedures, data, verification reports
 - (e) Source code
 - (f) Support software

3.1.3 Definitions. Subsection 1.3 of the Plan is used to capture all definitions needed for understanding the Plan or helpful for communication.

Issues to Consider in Planning Section 1.3 - Definitions

- (1) Are the definitions easily understood?
- (2) Is there a list of definitions that can be easily referenced?
- (3) Do you really need to define a new term?
- (4) Can a glossary of acronyms be used?

¹⁰ Throughout the guide, when lists are added to questions in the *issues to consider* NOTES, the lists are to be considered as suggested items, not an exhaustive checklist as in a standard.

It is best to use standard definitions that are common to the industry. For example, terms defined in ANSI/IEEE Std 729-1983 [1] have been arrived at by a consensus of professionals in the industry; it is a good source to use. Numerous new definitions tend only to make understanding the Plan more difficult. Define only those new terms that have to be defined—usually specific to the computer program product. Also duplicating definitions used elsewhere leads to unnecessary work to maintain them current—another configuration management task.

3.1.4 References. Subsection 1.4 of the Plan lists the documents cited elsewhere in the Plan. References made here refer to existing documents and, when customers are involved, the contractual standards and directives cited in the Plan. Having all the references in one place eliminates duplication of citing different sources. This makes a Plan that is more readable and supports general standardization of work instructions.

Issues to Consider in Planning Section 1.4—References

- (1) Can policies, practices, and procedures that already exist within the organization be referenced?
- (2) Is each reference necessary for the Plan?
- (3) Are some references a part of the organization's directive system?

Large, critical software developments, such as illustrated in Appendix A, tend to rely on a set of standards that are shared with other projects. This makes for better communication among those using the same general system but at the cost of some flexibility. Smaller projects, such as cited in Appendix B do not need the cross-checks and redundancy of these generalized standards and tend to rely on fewer documented standards.

Referencing helps to reduce the bulk of the document that must be maintained. Care should be taken to reference only those documents that are directly applicable to the Plan. Excessive references will lessen the effectiveness of the more important references. A distinction should be made between references that are necessary for *execution* of the Plan and those documents that are included as *general or supplementary* information.

3.2 Management. Section 2 of the Plan has the theme of relating the elements of the SCM discipline to specific activities of the project's or company's management organization. It also provides an opportunity to specify budgetary, schedule, and resource requirements necessary to carry out the Plan.

3.2.1 Organization. In 2.1 of the Plan, functions are allocated to organizational entities. Interfaces between organizations are handled in a separate section (2.3). The functions of the SCM department itself (if it will exist) are defined in more detail in 2.2. It is not necessary or desirable in most cases to allocate all SCM functions to an SCM department; SCM is a part of the entire software engineering process and as such may best be accomplished by the various organizations actually performing the systems engineering or integration. Software Development, Systems Engineering, Test and Quality Assurance departments all may assume significant roles in carrying out SCM functions. The *Issues to Consider* listed below are designed to provide a starting point in looking at the project's work-flow in relation to the current management structure and to support consideration of how the SCM activities can be best allocated or coordinated.

**Issues to Consider in Planning
Section 2.1 — Organization**

- (1) What kind of product interfaces have to be supported within the project itself?
 - (a) Software—hardware
 - (b) Software—software
 - (c) Software maintained at multiple sites
 - (d) Software developed at different sites
 - (e) Dependencies on support software
 - (f) Maintenance changes generated from different sites
- (2) What are the capabilities of the staff available to perform CM specific activities?
- (3) What is the management style of the organization within which the software is being developed or maintained?
- (4) Who will be responsible for maintaining the support software?
- (5) What organizational responsibilities are likely to change during the life of the Plan?
 - (a) Project management organization

- (b) Organizational interfaces
 - (c) Configuration management organization
- (6) Who has the authority to capture data and information and who has authority to direct implementation of changes?
- (7) What are the plans for maintaining current organization charts(s)?
- (8) What level of management support is needed for implementing various portions of the SCM discipline?
- (9) Will the project management be confined to a single organization or will it be distributed among several organizations?
- (10) Are responsibilities for processing changes to baselines clearly indicated, including who
 - (a) Originates changes
 - (b) Reviews changes
 - (c) Signs-off changes
 - (d) Approves changes
 - (e) Administers the process
 - (f) Validates and checks for completion?
- (11) Who has the authority to release any software, data, and associated documents?
- (12) Who has the responsibility for various SCM activities?
 - (a) Ensuring the integrity of the software system
 - (b) Maintaining physical custody of the baselines
 - (c) Performing product audits (versus quality audits)
 - (d) Library management
 - (e) Developing and maintaining specialized SCM tools
- (13) How is authority vested for handling exceptional situations and waivers?

If the plan for maintaining organizational charts shows a certain organization or management group (such as the program office or the business management office) assuming this responsibility, it may be wise to reference those charts in the Plan rather than placing the actual chart in the document, which must then be maintained every time another group of charts is updated. Alternatively, the organizational chart may be shown in the initial version of the Plan with a footnote directing readers to the proper official source for updates. It is usually best to include organizational charts that refer only

to functional names (such as department names) rather than to individuals responsible for managing them. This information is quite dynamic in most organizations, and it is probably not worth updating a Plan every time a department is assigned a new manager.

Consider advantages of alternative forms of organizing activities. Appendix A illustrates a complex, critical software development where there is a strong need for independence and centralization of SCM duties in a functional type organization. Appendix C also illustrates a functional type organization but for a different reason: in a software maintenance environment, SCM plays a stronger role in managing the change processing, even to the scheduling of work—more so than in a typical development environment.

Another point to consider is the management support for the various SCM disciplines. Note, for example, in Appendix B the management supported some concepts of SCM but wanted the process to be as *painless* as possible for the software developers and customers. The SCM administrator established a method of collecting information necessary to achieve the purpose without interfering with the flow of changes to the sites. Similarly, the other Appendixes illustrate SCM practices that are tailored to the reality of the situations in which they are found.

For ease of reading, organize the tasks and the owners in terms of the classical set of CM functions: identification, configuration control, status accounting, and audits and reviews. The matrix in Appendix A, Table 1 illustrates how this kind of information can easily be presented.

3.2.2 SCM Responsibilities. If a specific SCM department or group is identified in the management structure, this section provides a specific description of the role this organization will play in the overall SCM process.

Issues to Consider in Planning Section 2.2—SCM Responsibilities

- (1) Are there any special considerations for this project that require the SCM department to change its standard method of doing business?
- (2) What explicit assumptions is the SCM group making in planning their part of the project?

- (3) Are there specific expectations on the part of the customer or client (such as contractual requirements) for an SCM group that need to be taken into account?

While the major considerations may center on responsibilities of the configuration control boards (CCB), there is the need to consider the responsibilities of other activities such as software quality assurance (SQA), users of the system, other system or hardware configuration control boards, and other management activities.

3.2.3 Interface Control. The theme of subsection 2.3 of the Plan is how SCM disciplines are coordinated and how they are used to manage interfaces throughout the project's life. This is the place to define the roles and composition of the various CCB, SCCB, and other activities and practices used for interface control. All types of interfaces should be considered.

The scope of the SCM Plan (1.2) specifies the boundaries of the CI and the jurisdiction of the Plan, but this boundary is often not as clear as it should be and the control mechanisms are even fuzzier. The definition of interfaces is one of the most important planning elements for ensuring a smooth operation. Every possible effort should be made to reach a common agreement regarding each organization's responsibility regarding the interface(s), and then document them in this subsection. The basic types of interfaces to consider here include organization, phase, software, and hardware.

Organizational interface elements include interfaces between various organizations involved with the product; for example, vendor to buyer, sub-contractor to contractor, and co-developer to co-developer. It is typical that different organizations have different views of a product and will apply different expectations to it. Effective SCM disciplines can help minimize and resolve these differences whenever and wherever they may arise.

Phase interface elements include transition interfaces between those life cycle phases of the product that are included in the Plan. They are often coincident with a transition in control of the product between different organizations; for example, promotion from a development group to a formal testing group. Effective SCM disciplines can support these transitions with all the documentation, code, data, tools, and records that are needed for management to smoothly continue SCM on the product.

Software interface elements are the agreements shared between the computer program product and other software entities (for example, operating system, utilities, communication system). These agreements involve the structure and meanings assigned to data passing and operational coordination of the data and the results. The other software may already exist or may be concurrently developed. Effective SCM disciplines can make these agreements generally known and assist management in maintaining the integrity of the product(s).

Hardware interface elements are the agreements shared between the computer program product and characteristics of any hardware in the environment with which the program product interacts. These agreements involve capabilities provided by the hardware and operations defined by the computer programs. Effective SCM disciplines help make these agreements known and support their evaluation for consistency through out the evolution of both hardware and software.

Issues to Consider in Planning Section 2.3—Interface Control

- (1) What are the organizational interfaces?
- (2) What are the important interfaces between adjacent phases of the life cycle?
- (3) What are the interfaces between different entities of the computer programs?
- (4) What are the dependent hardware interfaces?
- (5) Where are the documents defined and maintained that are used in interface control?
- (6) What are the procedures for making changes to these interface documents?

Interface control should be extended to include more than just documentation. If the hardware configuration and its supporting software interfaces are complex, then the Plan must also include or reference controls for hardware drawings and equipment as well. The sample Plan in Appendix D illustrates the interface between multiple kinds of computer programs in a variable hardware configuration. In real-time system environments, the interface controls may involve tracking changes to configurations of external sensors, valves, etc. Typically, in a software modification and maintenance situation, human operator interface controls may play a significant role in this section. In

some organizations, [EP]ROM are considered hardware, yet the programs residing in them must be explicitly dealt with in this section of the Plan. The guiding principle of SCM is that any proposed changes to the product or to its expected use be considered and evaluated in an open, documented, deliberate, and mutually acceptable fashion.

3.2.4 SCM Plan Implementation. Subsection 2.4 of the Plan has the theme of providing details concerning the implementation of the key SCM milestones identified in the Plan. These details include:

- (1) Identification of prerequisites or required activities that affect the Plan and the sequencing of events in the Plan
- (2) Schedules for accomplishing these items
- (3) Resource requirements (for example, machine time, disk space, specialized tool availability, and staff support)

The implementation section's level of detail and complexity are dependent on the level of complexity of the system being controlled. Small software development activities, particularly those that focus primarily on software and are not currently tied to hardware systems development, may need relatively simple implementation schedules. SCM Plans that support more complex activities, such as software maintenance (Appendix C) or development and maintenance of product line software (Appendix D), will have more complex implementation schedules but will focus more on events such as release for use, new product baselines, audits, and reviews.

Issues to Consider in Planning Section 2.4—SCM Plan Implementation

- (1) Are the resources planned for SCM commensurate with the size and complexity of the system being controlled?
- (2) How will the SCM activities be coordinated with other project activities?
- (3) How will the different phases (development, maintenance) be managed throughout the software life cycle?

Resource requirements should be carefully considered and included here only when they are important factors in implementing the Plan. If there are any separate project documents that contain the necessary information (for example, department budgets,

development laboratory implementation Plans), include them here by reference to avoid unnecessary document maintenance.

Items to include are:

- (1) People resources
- (2) Computer and computer-related resources
- (3) Library space
- (4) Storage space (including electronic media)

It is usually impractical to put actual dates in the Plan for events. In general, it is better from the maintenance perspective to put actual dates in a schedule chart kept in an appendix or a separate document. In this section it is more appropriate to refer to significant events in terms of their relationships to other milestones (for example, a controlled library for source code will be established following the completion of the critical design review), or in terms of their relationship in time to other events (for example, the physical configuration audit will be held 90 days after the functional qualification test).

Requirements for implementation should be discussed in the same sequence in this section as they are discussed in the body of your Plan (for example, configuration identification is followed by product baselines). This should make correlating the Plan with the implementation considerations easier for the user.

Keep in mind that this section should be updated as the project continues. Consider reviewing this section and making any necessary additions or changes upon the achievement of each major milestone in the system development life cycle (for example, completion of functional design) or on a periodic basis (for example, once per quarter).

Project managers are often asked to provide a budget for SCM separate from the development budget. Little historical data are reported in the literature, primarily because every SCM activity has a slightly different organizational structure. In the example given in Appendix B, the project defined 0.5 full time equivalent man-months. Other types of projects, such as illustrated in Appendix A, will require a larger portion of dedicated SCM personnel. In general, however, as more effective automated tools are deployed and used, the need for dedicated personnel will diminish.

3.2.5 Applicable Policies, Directives, and Procedures. Subsection 2.5 of the Plan has the theme of identifying and defining the degree to which existing and future SCM policies and procedures apply to the Plan. The actual identification of referenced documents, and information on how to obtain them should be cited in Section 1.4 of the Plan. Subsection 2.5 provides the opportunity to *interpret the use* of reference document(s) and to describe any new document(s) that may be planned or are under development (which, obviously, cannot be cited in Section 1.4 of the Plan).

Issues to Consider in Planning Section 2.5—Applicable Policies, Directives and Procedures

- (1) Are any standard identification procedures available?
 - (a) Standard labels for products
 - (b) Identification of the hierarchical structure of computer programs
 - (c) Component and unit naming conventions and limitations
 - (d) Numbering or version level designations
 - (e) Media identification methods (including [EP]ROM)
 - (f) Data-base identification methods
 - (g) Documentation labeling and identification standards
- (2) Are any specific procedures existing for interacting with the dynamic libraries?
 - (a) Promoting from one type of library to another
 - (b) Documentation releases
 - (c) Releasing computer program products
 - (d) Releasing firmware products
- (3) Are there standard procedures for managing the change process?
 - (a) Handling change or enhancement requests
 - (b) Provisions for accepting changes into a controlled library
 - (c) Processing problem reports
 - (d) Membership in CCB
 - (e) Operating CCB
 - (f) Capturing the audit trail of changes
- (4) Are any status accounting procedures available?
 - (a) Reporting procedures for summarizing problem reports

- (b) Standard reports and other formatted management information
- (c) Distributing status data
- (5) Are there procedures for audits?
 - (a) Procedures for functional configuration audits
 - (b) Procedures for physical configuration audits
- (6) Are there procedures for other general SCM activities?
 - (a) Standards for accessing and controlling libraries, including security provisions, change processing, backups and long-term storage
 - (b) Forms or file definitions for problem reports, change requests, documentation change notices, etc

The set of procedures need not be developed at one time; but effort consistently applied over a period of time can generate an adequate set of policies and procedures that are effective. The kinds of policies, directives, and procedures that are part of an organization's general practices and procedures might also be considered a part of the Plan.

documentation. This is the level associated with released programs. The second level, the labeling of files (parts), is more unique to software and is constrained by the support software used in generating code. File nomenclature must support the structure of the product. Typically, these files are identified with mnemonics unique to a project and need to be correlated back to the identification scheme. This is the level associated with the parts of a released program. SCM not only must set identification schemes for both of these levels, but also must devise a method for relating the two different views of the same product.

Project management generally determines the criteria for identifying CI and subordinate control level items. SCM then devises the identification numbering or labeling structure for tracking those entities.

Other kinds of problems that should be considered include legal responsibilities. Some contracts require that all new code added to a program belongs legally to the owner of the original computer programs. Problems of third-party software acquisition must also be considered. The legal status of each program should be accurately identifiable before the computer programs are released for use. Usually some controls must be placed on the number of copies of third-party software *passed through* and delivered to customers as royalty payments might even be required.

3.3 SCM Activities. The SCM organizational descriptions in Section 2 of the Plan describe who has what responsibilities for software configuration management. Section 3 of the Plan describes how these groups accomplish their responsibilities.

3.3.1 Configuration Identification. The theme of this subsection is to document an identification scheme that reflects the structure of the product. This is a critical task of SCM, a most difficult task but one that is necessary for a smoothly running SCM operation. It is critical because the flow of management control must follow the structure of the software being managed. It is important because the identification scheme carries forth through the life of the computer program(s). It is difficult because at the time the identification scheme is constructed, the structure of the product is rarely known to the level of detail required during the development process.

Relating the identification scheme to the structure of the computer programs is complicated because there are generally two levels of identification that SCM has historically kept separate. The first level, the identification of configuration items and components recognized by management and users, is identified traditionally by

Issues to Consider in Planning **Section 3.1 — Configuration Identification**

- (1) What scheme is to be used to relate the identification of files to the formal (document based) identification scheme?
- (2) How does one relate the software identification scheme to the hardware identification scheme when the computer programs are deeply embedded in the system (for example, device controller firmware, code and data split between ROM firmware and loadable RAM image code)?
- (3) How does one identify computer programs embedded in [EP]ROM?
- (4) What specifications and management plans need to be identified and maintained under configuration management?
- (5) What timing is involved in naming documents as CI?

- (a) When does a document enter into controlled status (for example, when presented by author, when reviewed, when rework is verified, or when the document is formally distributed)?
- (b) When and how does a document get removed from the CI status?
- (6) Is a separate identification scheme needed to track third-party software?
- (7) Is a special scheme needed to identify reusable/reused software as different from other software parts?
- (8) Are there differences in identification across projects that have different fiscal accounting?
- (9) How does one identify support software such as language translators, linkers, and cross-support tools?
- (10) Is a special identification scheme needed to identify test data (transaction files, data-bases, etc) that must be kept for regression testing?
- (11) Is there a need to identify tables and files for data driven systems?

One practice for identification of parts of a CI (as illustrated in Appendix A) is to use a version description document to relate the different files to the component or configuration item scheme. A suggested practice for embedding computer programs into hardware systems is illustrated in Appendix D where the system index type of project identification is used.

The management of firmware changes can become difficult when the package becomes a part of the hardware item. The problem remains to relate functional capabilities to physical part identifiers, especially when changes to the firmware are closely coupled to changes in the system or application software (for example, boot loaders, device controllers, and high-level ROM-resident system debuggers).

Third-party software needs to be tracked even though it is not changed in the same manner as other software. This is especially important if you, as a reseller, accept responsibility of collecting and dealing with problem reports generated by your customers for these products. It may be necessary too for compliance with legal restrictions on copies and distribution accounting. Appendix C describes this identification situation.

The successful reuse of pieces of software

in a (controlled) production library requires a standardized identification scheme to retrieve packages or units and account for their use in different configurations. Appendix C, 3.1 references identification of reused software. It should be noted that it is important to control the test procedures and test cases needed for regression testing in an environment that maintains such software or has extensive dynamic libraries of reusable software.

The identification scheme needs to reference dependent supporting software. Therefore, provisions must be made for identifying the internal documentation, data, and programs used in the generation of the computer program product(s).

3.3.1.2 Identify Project Baselines. Baselines are an effective mechanism to allow many people to work together at the same time. They are a way of synchronizing people working on the same project. The SCM discipline, as in all CM, focuses its activity around the construction and maintenance of baselines. The modifiable units need an identifying mechanism, and a way of describing what is contained in their aggregates is needed. Even if the program is small, a baseline is used to let the other, nonprogramming people, know what is taking place.

Issues to Consider in Defining Baselines

- (1) Are baselines other than, for example, the traditional three required¹¹ to support the project?
- (2) Who is needed to authorize the creation of new baselines?
- (3) Who approves a baseline for promotion?
- (4) How and where are the baselines created and who is responsible for them?
- (5) How will the numbering system account for different baselines?

¹¹The traditional baselines used in CM (functional, allocated, product) are defined in ANSI/IEEE Std 828-1983 [2] along with the minimal requirements for identifying and establishing those baselines. Additional internal or developmental baselines can be defined and included in the Plan when necessary. For example, in making multiple builds, it is useful to define separate baselines for each build to keep the status of changes straight. The sample SCM Plan in Appendix B illustrates the use of multiple builds. These developmental baselines are very helpful for integrating and testing large software systems.

- (a) Different versions (functional changes)
- (b) Different revisions (something to make the existing functions work)
- (6) How are baselines verified?
 - (a) Reviews
 - (b) Customer approval
 - (c) Developer test reports
 - (d) Independent verification and validation
- (7) Are baselines tied to milestones?
 - (a) Developmental milestones
 - (b) New versions

Baselines tie documentation, labeling, and the program together. Developmental baselines define a state of the system at a specific point in time, usually relative to an integration level, and serve to synchronize the engineering activity and documentation that occurs at that time.

Promotions are basically a change in the informal authority required to effect changes in developmental baselines. The new authority commonly represents a higher level of engineering management. The programmer cannot change a unit that has been promoted and integrated with other programmer's units without notifying the others involved, and gaining their (explicit or implicit) approval by way of an SCCB (or CCB).

The more formal baselines (functional, allocated, and product) define a product capability associated with performance, cost, and other user interests. These baselines relate the product to contractual commitments.

- (5) Does the identification scheme need to identify the hierarchy of links between modifiable entities?
- (6) Are there constraints on unit and file names?
 - (a) Compiler and file system limitations on name length and composition
 - (b) Mnemonic requirements
 - (c) Names that cannot be used

It is often useful to have the identification label indicate the level (that is, release, version, and revision) of the product it identifies. Labeling the components or units of computer programs can be accomplished in several ways. Numbering schemes can be devised to identify the components. A hierarchy of names can be devised that organizes and identifies parts using mnemonic or English labels. Naming conventions that are associated with the compilation system and are significant for a project are most easily used.

[EP]ROM labeling has special problems and will require a different scheme than that used for RAM-based packages shipped on disk or tape. In developing embedded computer programs, there is the additional consideration of labeling the media ([EP]ROM) with the correct version of the programs. This means that the identification scheme of some computer program packages must somehow relate to the hardware identification scheme. One possible solution is to use the version description document (VDD) form for relating the computer program identification documents to the altered item drawings conventionally used for identifying the [EP]ROM parts.

3.3.1.3 Delineate Project Titling, Labeling, Numbering. This part of the Plan defines the procedures and labels for identifying the CI, components, and units. This is important for identifying and retrieving information, reporting status and for legal protection of data rights.

Issues to Consider in Labeling and Numbering

- (1) Is there a (corporate) standard for labeling that must be used?
- (2) Does the identification scheme provide for identification of versions and revisions for each release?
- (3) How can or will the physical media be identified?
- (4) Are specific naming conventions available for all modifiable entities?

3.3.2 Configuration Control. Subsection 3.2 of the Plan describes how the configuration control process is managed. The theme here deals with identifying the procedures used to process changes to known baselines. An appropriate level of authority for controlling changes must be identified or delegated for each baseline. The organizations assigned responsibilities for control in Section 2 of the Plan have to manage changes made to the entities identified as defined in Section 3.1 of the Plan. Procedures for processing the requests for changes and approvals must be defined.

3.3.2.1 Levels of Authority. The levels of authority required to make changes to configuration items under SCM control can vary. The system or contract may often dictate the level of authority needed. For example, internally controlled software tools may require less change controls than man-rated or critical software developed under contract. The levels of authority may vary throughout the life cycle. For example, changes to code in the development cycle usually require a lower level of control to be authorized than changes to the same code after it has been released for general use. The level of authority required can also depend on how broadly the change impacts the system. A change affecting specifications during the requirements analysis phase has less ramifications than a change affecting software in operational use. Likewise, changes to draft versions of documents are less controlled than changes to final versions. Changes to a product distributed to several sites and used by many different users requires a different level of authority than products with a very restricted or minimal user base.

The level of control needed to authorize changes to developmental baselines depends on the level of the element in relation to the system as a whole (for example, a change in logic affecting one or a few units usually has less impact on the system than an interface between CI, especially if the CI are developed by different organizations.

Issues to Consider in Defining Levels of Authority

- (1) Is the level of authority consistent with the entities identified in subsection 3.1 of the Plan?
- (2) When are levels of control assigned to the modifiable units (parts) of the computer programs during top level and detail design stages (technical engineering phase) for developmental baselines?
- (3) Do control levels assigned for developmental baselines (for both components and configuration items) need to be reviewed by management.
- (4) Are there significant increases in levels of control for transitions between developmental baselines?
 - (a) During design
 - (b) For promotion from design to implementation
 - (c) For unit testing

(d) For integration

- (5) Does management need to know specifically who requested a change?
- (6) Do changes originating from outside the organization, such as customers or general users, require different authority for approval than changes from a technical development group?
- (7) Do changes that do not impact formal baselines require coordination and approval by a CCB or can they be authorized by a SCCB?

The Plan should clearly define the level of authority to be applied to the baselined entities throughout the life cycle of the system, and should distinguish between controls applied to processing technical changes that do not impact formal baselines and the authority needed to approve changes to formal baselines. For example, during maintenance or in the latter stages of preparing multiple builds for a project, authority for making changes to all entities at all levels is typically restricted. However, when beginning development of a new version or build, the controls on the dynamic library and testing with the controlled library can be relaxed.

Table 4 suggests some ideas for assigning different levels of change authority to different SCM elements during the life cycle.

3.3.2.2 Processing Changes. The theme of these paragraphs is to describe the methods to be used for processing change requests. Generally, no single procedure can meet the needs of all levels of change management and approval levels. These paragraphs must concentrate on

- (1) Defining the information needed for approving a change
- (2) Identifying the routing of this information

Table 4
Variable Levels of Control

Element	Internal Coordination	Developmental Baselines	Formal Baselines
Specifications	Supervision	CCB	CCB
Test data	Supervision	SCCB	CCB
Unit code	Supervision	SCCB	CCB
Configuration Item code	SCCB	CCB	CCB

- (3) Describe the control of the library(ies) used in processing the changes
- (4) Describe or refer to the procedure for implementing each change in the code, in the documentation, and in the released program (for example, field upgrades).

The change initiator should analyze the proposed change to assess its impact on all configuration items (documentation, software, and hardware) and the CCB should satisfy themselves that this has been done and interface with the CCB in the impacted areas (if any).

Another area that is often overlooked (and not specifically covered) is that of the maintenance of design documentation. The documentation hierarchy should be fully defined and a change to any level should be analyzed to ensure that the higher levels of documentation have been considered and that the change is *rippled through* the lower levels to implementation in the code.

Source code changes, and indeed hardware changes, should first be implemented in the highest level of documentation and the change implemented through the subsequent levels. Provisions for backing up of changes and maintaining their history need to be considered.

A more critical issue centers on managing controlled libraries. This configuration management concept grew out of the SCM experience with managing source code and has been expanded to include all of the baseline items (including associated documentation and reports) that relate to the computer programs. One can observe that as the interactive programming environments continue to evolve, most of the procedural controls associated with SCM will probably be integrated into the programming environment. The procedures for processing changes are the same, whether for approval by a designated management authority or approval by a control activity (SCCB) delegated by management. The procedure needs to distinguish the proper channels for making the decisions, defining the flow for changes made to an established formal baseline, and the flow for changes made to developmental baselines. Most of this capability is now available in SCM and software engineering tools in one form or another.

Issues to Consider in Processing Changes

- (1) What is the information necessary for processing a software/system change

request (SCR) or authorizing a change (SCA)?

- (2) What kind of information will a CCB or SCCB need in order to make a decision?
- (3) What is the overall processing cycle of changes?
- (4) What SCM support is provided by automated tools available in the environment?
- (5) Will changes in procedures be required to support different kinds of reviews during each of the phases of the life cycle?
- (6) When there are multiple CCB in a hierarchy, what are the procedures for information exchange and approval chains?
- (7) Is there a need for dynamic libraries and controlled library interfaces?
- (8) Is there a need for controlling all access to a library or just controlling changes made to the information in the libraries?
- (9) Does the library system provide an audit trail, such as change histories?
- (10) Are back-up and disaster files taken into account?
- (11) Are there provisions for archive procedures to provide the static library support to the full life cycle?
- (12) How are source items (source code) associated with their derived object (executable code) programs?
- (13) What are the provisions for *draw down* or *check out* to get units from the controlled library?
- (14) What are the provisions for keeping the data files synchronized with the program(s) using them?
- (15) How does the change process itself support or accommodate the development of new versions or revisions?

Some library tools maintain *deltas* to base units of source code. Procedures for maintaining version histories of units as well as derived configuration items need to be established along with archiving maintenance.

A CCB concerned with project management may need information regarding estimated cost and schedules for a change, as illustrated in Appendix B. Other CCB may be interested only in the technical interfaces affected by a change, as illustrated in the sample Plan in Appendix C. Still others may need, in addition, information on proprietary

rights and copyrights affected, as illustrated in Appendix D.

Some CCB review a proposed change to validate it (approve it as a necessary change; to expend time and resources for investigating feasibility of the change) while others may simply want completed (programmed and documented) changes to be approved prior to inclusion in released computer programs. There are different functions of SCCB responsibility, extending from coordinating engineering technical changes to allocating the work to a work group. Some organizations design, code and test all proposed changes with preliminary CCB approval before submitting them for final CCB approval. This technique may reduce total time to produce a change. The process for granting change approvals must guarantee that unauthorized changes do not contaminate baselined software.

Some advanced tools provide capabilities for formatting change requests, routing to different sets of individuals for approvals, and authorizing work to be done; reviewing changes and tests while in a holding area; and releasing a baseline to a controlled library for operational use. Others provide only for the recording of change information and a history of past versions of source code.

If secure procedures are not in place or feasible for controlling a library system, the library may necessarily be divided into physical entities that control access.

3.3.2.3 The Configuration Control Board.

The theme of these paragraphs is identifying the authorities needed for granting change approvals. Subsection 2.2 of the management section of the Plan identifies the general role(s) of each CCB. These paragraphs go into detail on the roles and authority. It should be remembered that the CCB has traditionally been concerned with managing changes to established baselines of documented configuration items and the components of those configuration items. There may be other change control bodies (SCCB) that authorize changes subordinate to the CCB described here. The CCB described in these paragraphs of the Plan have the role of authorizing changes to baselined configuration items and components from the point of view of entrepreneurial management. They reflect concerns over the costs, schedules, and resources available to implement changes in response to user desires for change.

Issues to Consider in Identifying Configuration Control Boards

- (1) Can the limits of authority be defined?
 - (a) Limited to contractual baselines as in Appendix A
 - (b) Limited to developmental baselines (noncontractual) as in Appendix D
- (2) Will the project mix computer programs that are controlled by other CCB?
- (3) Is there a need to limit the CCB tabling actions by setting time limits?
- (4) Are there contractual requirements imposed on a CCB that must be reflected in the Plan?
- (5) How are the different levels of authority determined?
- (6) How are different organizational bodies phased in when transitioning from one phase of the life cycle to another?
- (7) How are changes to a baselined product to be batched together for release?
 - (a) For a new version
 - (b) For a revision
- (8) Does the CCB membership reflect the management style of the organization?
 - (a) For a functional organization
 - (b) For a matrixed organization

Large, complex systems require ongoing configuration control authorities to coordinate the technical work involved in generating specifications and code, and in continuing the work of technical coordination required for maintaining interacting software systems (such as defined in Appendix C). Such projects use the same principles of configuration management and perform the same generic approval and scheduling functions as the CCB concerned with smaller-scale entrepreneurial management, particularly where automated SCM tools are used to support both types of activities.

Large software systems are frequently not completely new. They are often mixtures of software in public domain, vendor-supplied products, vendor supplied but modified by a contractor, subcontracted software, proprietary software, and software paid for on another project but reused or adapted. The procedures of how the CCB handles the special nature of proprietary software and reusable software are important and need to be specifically addressed in a Plan.