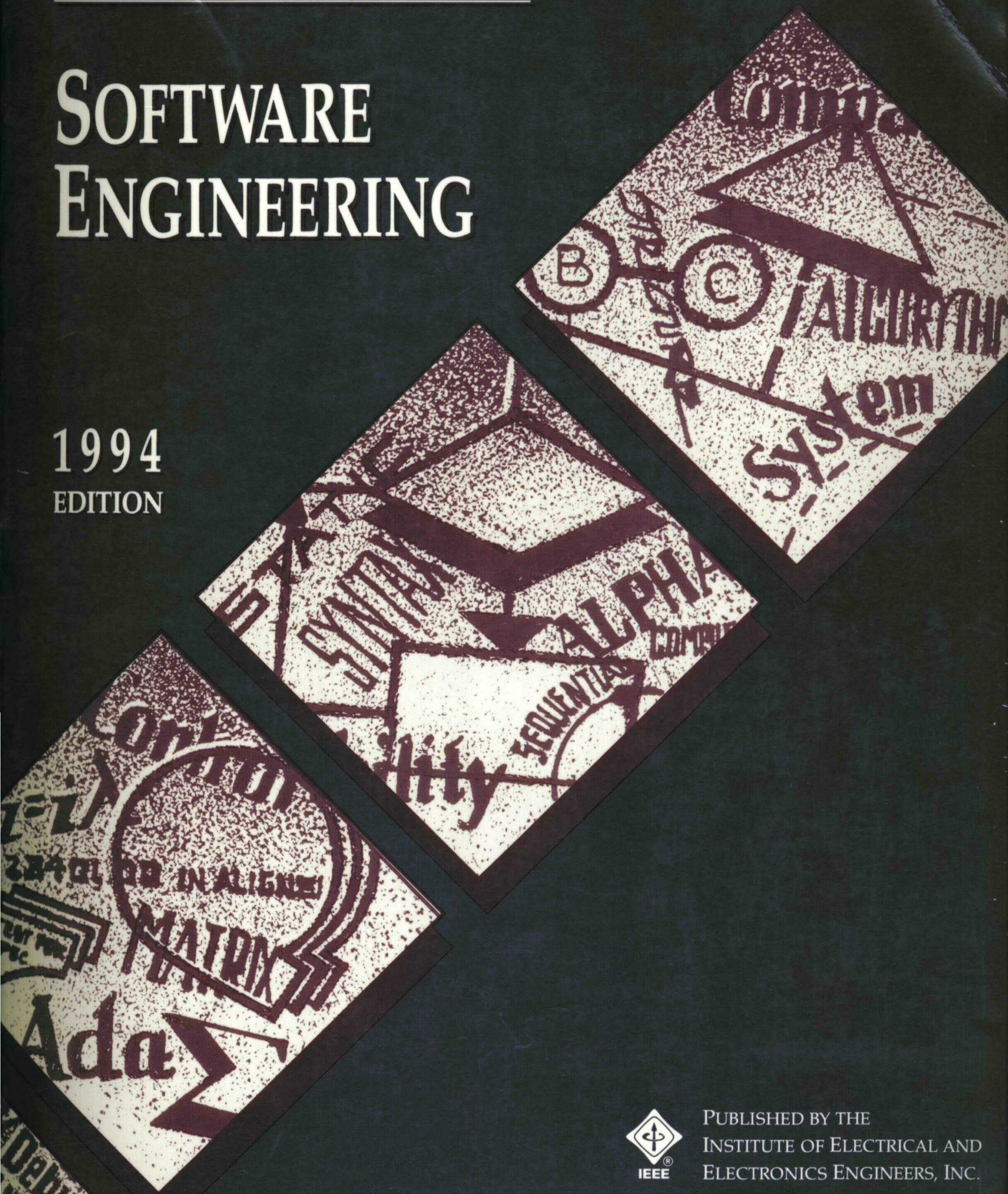
IEEE STANDARDS COLLECTION

SOFTWARE ENGINEERING

1994
EDITION



PUBLISHED BY THE
INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.

Recognized as an
American National Standard (ANSI)

IEEE
Std 828-1990
(Revision of IEEE Std 828-1983)

IEEE Standard for Software Configuration Management Plans

Sponsor

Software Engineering Standards Subcommittee of the Technical Committee on Software Engineering of the IEEE Computer Society

Approved September 28, 1990

IEEE Standards Board

Approved February 15, 1991

American National Standards Institute

Abstract: IEEE Std 828-1990, *IEEE Standard for Software Configuration Management Plans,* establishes the minimum required contents of a Software Configuration Management Plan and defines the specific activities to be addressed and their requirements for any portion of a software product's life cycle.
Keywords: configuration control board, configuration items, software configuration management, software configuration management activities.

ISBN 1-55937-064-5

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least once every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Foreword

(This Foreword is not part of IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans.)

This standard is concerned with the activity of planning for software configuration management (SCM). SCM activities, whether planned or not, are performed on all software development projects; planning makes these activities more effective. Good planning results in a document that captures the planning information, makes the information the property of the project, communicates to all who are affected, and provides a basis for ongoing planning.

SCM is a formal engineering discipline that, as part of overall system configuration management, provides the methods and tools to identify and control the software throughout its development and use. SCM activities include the identification and establishment of baselines; the review, approval, and control of changes; the tracking and reporting of such changes; the audits and reviews of the evolving software product; and the control of interface documentation and project supplier SCM.

SCM is the means through which the integrity and traceability of the software system are recorded, communicated, and controlled during both development and maintenance. SCM also supports reduction of overall software life cycle cost by providing a foundation for product and project measurement.

SCM constitutes good engineering practice for all software projects, whether phased development, rapid prototyping, or ongoing maintenance. It enhances the reliability and quality of software by

- Providing a structure for identifying and controlling documentation, code, interfaces, and databases to support all life cycle phases
- Supporting a chosen development/maintenance methodology that fits the requirements, standards, policies, organization, and management philosophy
- Producing management and product information concerning the status of baselines, change control, tests, releases, audits, etc.

IEEE Std 828-1983 was originally prepared by a Working Group of the Software Engineering Standards Subcommittee of the Technical Committee on Software Engineering of the IEEE Computer Society and approved in June, 1983, by the IEEE Standards Board. This current standard is the first revision and has been completely rewritten to

- Update the standard to recognize current software engineering practices
- Be consistent with IEEE Std 1042-1987, IEEE Guide to Software Configuration Management, which is anticipated to be reviewed and revised as Std 828.1 to maintain this consistency
- Be more flexible and easier to use for all levels of expertise

The following individuals contributed to IEEE Std 828-1990 by attendance at two or more working sessions or substantial written commentary or both:

H. R. Berlack, *Co-Chair* **M. Updike-Rumley,** *Co-Chair*

Editorial Committee

R. Frederick
D. L. Knirk
L. Roy

Working Group

B. Banerjee
F. J. Buckley
B. Conger
M. A. Daniels
N. P. Ginex
A. Hartman
R. Horner
W. M. Osborne
B. F. Rospide
D. P. Schwartz
E. Showalter
L. Siwiec
M. Swain
L. Tran
R. L. Van Tilburg
A. M. Vaughan

At the time that the IEEE Standards Board approved this revision, the Software Engineering Standards Subcommittee, which was the balloting committee that approved this document for submission to the IEEE Standards Board, included the following members:

John W. Horch, *Chairman*

K. A. Alley
R. L. Aurbach
M. Azuma
B. Banerjee
L. Beltracchi
M. Ben-Menachem
H. R. Berlack
W. J. Boll, Jr.
G. Bracon
K. L. Briggs
W. L. Bryan
F. J. Buckley
H. Carl
E. Chikofsky
F. A. Chimenti
S. V. Chiyyarata
T. S. Chow
P. H. Christensen
W. L. Chung
R. T. Close
F. Coallier
P. W. Daggett
M. A. Daniels
T. Daughtrey
P. I. Davis
P. A. Denny
B. Derganc
P. R. DeWeese
H. Dhama
K. Diggins
J. H. Dobbins
D. C. Doty
E. Dunaye
R. E. Dwyer
M. L. Eads
R. S. Euler
R. L. Evans
R. Fairley
J. Fendrich
A. M. Foley
J. J. Forman
J. Forster
M. Galinier
A. K. Geraci
Y. Gershkovitch
E. L. Gibbs
J. A. Gilmore
N. P. Ginex
S. Gloss-Soler
M. Goldberg
J. Gonzalez-Sanz
J. K. Grau
D. A. Gustafson
U. Hartmann
C. M. Hay
C. P. Hollocker
R. S. Hurst
L. Ihlenfeldt
F. Jay
D. Johnson III
M. S. Karasik
R. H. Karpinski
A. N. Kasad
R. A. Kessler
P. P. Klopfenstein
D. M. Knepper, Sr.
D. L. Knirk
S. Koenig
R. Kosinski
J. Krupinski
T. Kurihara
R. A. Lane
G. N. Larsen
F. C. Lim
B. Livson
I. S. Ludin
A. J. Maher
K. C. Majumdar
H. A. Malec
P. C. Marriott
R. J. Martin
T. Matsubara
I. Mazza
J. P. McArdle
M. C. McElvany
P. E. McKenney
G. A. Meldrum
J. Mersky
C. H. Modell
R. E. Monahan
R. C. Natale
G. Neidhart
D. E. Nickle
W. M. Osborne
D. E. Peercy
W. E. Perry
D. J. Pfeiffer
R. M. Poston
I. C. Pyle
J. Rault
M. Razy
S. T. Redwine, Jr.
B. F. Rospide
F. A. Ruhlman
D. A. Rutherford
S. R. Schach
W. A. Schnoege
R. G. Schueppert
D. J. Schultz
G. Schumacher
D. P. Schwartz
L. W. Seagren
T. Sgarlatti
G. P. Shabe
R. W. Shillato
E. Showalter
D. M. Siefert
M. W. Smith
R. Staunton
R. N. Sulgrove
W. G. Sutcliffe
C. Swain
B. J. Taute
P. U. Thompson
S. Trauth
D. L. Ulery
M. Updike-Rumley
R. L. Van Tilburg
A. M. Vaughan
H. E. Verne
D. R. Wallace
J. W. Walz
A. H. Weigel
A. T. Williams
A. F. Wilson
T. J. Wojcik
P. A. T. Wolfgang
D. L. Wood
A. W. Yonda
N. C. Yopconka
P. F. Zoll

When the IEEE Standards Board approved this standard on September 28, 1990, it had the following membership:

Marco W. Migliaro, *Chairman* **James M. Daly,** *Vice Chairman*

Andrew G. Salem, *Secretary*

Dennis Bodson
Paul L. Borrill
Fletcher J. Buckley
Allen L. Clapp
Stephen R. Dillon
Donald C. Fleckenstein
Jay Forster*
Thomas L. Hannan
Kenneth D. Hendrix
John W. Horch
Joseph L. Koepfinger*
Irving Kolodny
Michael A. Lawler
Donald J. Loughry
John E. May, Jr.
Lawrence V. McCall
L. Bruce McClung
Donald T. Michael*
Stig Nilsson
Roy T. Oishi
Gary S. Robinson
Terrance R. Whittemore
Donald W. Zipse

*Member Emeritus

Contents

IEEE Standard for Software Configuration Management Plans

1. Introduction to the Standard

1.1 Scope. IEEE Std 828-1990 establishes the minimum required contents of a Software Configuration Management (SCM) Plan (the Plan). It is supplemented by IEEE Std 1042-1987 [4][1], which provides approaches to good software configuration management planning. This standard applies to the entire life cycle of critical software; for example, where failure could impact safety or cause large financial or social losses. It also applies to noncritical software and to software already developed. The application of this standard is not restricted to any form, class, or type of software.

The Plan documents what SCM activities are to be done, how they are to be done, who is responsible for doing specific activities, when they are to happen, and what resources are required. It can address SCM activities over any portion of a software product's life cycle.

The content of the Plan is identified in Section 2 of this standard. The required information is indicated by the words "shall" and "required." Additional optional information is also identified as appropriate. The user of this standard, however, is expected to expand and supplement the minimum requirements as necessary for the development environment, specific industry, organization, and project. Tailoring of a plan in conformance with this standard is described in Section 3.

The primary users of this standard are assumed to be those planning SCM activities or performing SCM audits.

In considering adoption of this standard, regulatory bodies should be aware that specific application of this standard may already be covered by one or more IEEE Standards documents relating to quality assurance, definitions, or other matters (see [2]). It is not the purpose of IEEE Std 828-1990 to supersede, revise, or amend existing standards directed to specific industries or applications.

1.2 References. This standard may be used in conjunction with the following IEEE Software Engineering Standards:

[1] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.[2]

[2] IEEE Std 730.1-1989, IEEE Standard for Software Quality Assurance Plans (ANSI).

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

[1]The numbers in brackets correspond to those of the references listed in 1.2.

[2]IEEE publications may be obtained from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

Table 1
SCM Classes of Information

Class of Information	Description	Section in Standard	Section in Plan
Introduction	Describes the Plan's purpose, scope of application, key terms, and references	2.1	1
SCM Management	*(Who?)* Identifies the responsibilities and authorities for accomplishing the planned activities	2.2	2
SCM Activities	*(What?)* Identifies all activities to be performed in applying to the project	2.3	3
SCM Schedules	*(When?)* Identifies the required coordination of SCM activities with the other activities in the project	2.4	4
SCM Resources	*(How?)* Identifies tools and physical and human resources required for execution of the Plan	2.5	5
SCM Plan Maintenance	Identifies how the Plan will be kept current while in effect	2.6	6

1.3 Definitions and Acronyms. The definitions below describe specific terms as used within the context of this standard.

control point (project control point). A project agreed on point in time or times when specified agreements or controls are applied to the software configuration items being developed, e.g., an approved baseline or release of a specified document/code.

release. The formal notification and distribution of an approved version.

Additional terms that are relevant are defined in IEEE Std 610.12-1990 [1], and are as follows: baseline, component, configuration, configuration audit, configuration control, configuration control board, configuration identification, configuration item, configuration management, configuration status accounting, interface, interface control, software, software library, software life cycle, unit, version.

The following acronyms appear within the text of this standard:

CCB Configuration Control Board
CI Configuration Item
SCM Software Configuration Management

The term "the Plan" is used throughout this standard to refer to the Software Configuration Management Plan.

2. The Software Configuration Management Plan

SCM planning information shall be partitioned into the six classes described in Table 1. The referenced sections of the standard provide the reader with detailed requirements for each class of information.

SCM planning information may be presented in any format, sequence, or location that is meaningful to the intended users of the Plan with the following restrictions:

(1) A document with the title "Software Configuration Management Plan" shall exist either in stand-alone form or embedded in another project document.
(2) This document shall contain all SCM planning information either by inclusion or by reference to other locations, such as other documents or automated systems.
(3) A format for this document shall be defined.

The writer of the Plan shall use the sequence of sections specified in Table 1 unless a different format has been defined in the Introduction of the Plan (see 2.1).

2.1 Introduction. Introduction information provides a simplified overview of the SCM activities so that those approving, those performing, and those interacting with SCM can obtain a clear understanding of the Plan. The

Introduction shall include four topics: the purpose of the Plan, the scope, the definition of key terms, and references.

The purpose shall briefly address why the Plan exists and who the intended audience is.

The scope shall address SCM applicability, limitations, and assumptions on which the Plan is based. The following items shall be included:

(1) Overview description of the software development project
(2) Identification of the software CI(s) to which SCM will be applied
(3) Identification of other software to be included as part of the Plan (e.g., support or test software)
(4) Relationship of SCM to the hardware or system configuration management activities for the project
(5) The degree of formality, depth of control, and portion of the software life cycle for applying SCM on this project
(6) Limitations, such as time constraints, that apply to the Plan
(7) Assumptions that might have an impact on the cost, schedule, or ability to perform defined SCM activities (e.g., assumptions of the degree of customer participation in SCM activities or the availability of automated aids)

Key terms shall be defined as they apply to the Plan in order to establish a common terminology among all users of the Plan.

All references in the Plan to policies, directives, procedures, standards, terminology, and related documents shall be uniquely identified to enable retrieval by users of the Plan.

2.2 SCM Management. SCM management information describes the allocation of responsibilities and authorities for SCM activities to organizations and individuals within the project structure.

SCM management information shall include three topics: the project organization(s) within which SCM is to apply; the SCM responsibilities of these organizations; and references to the SCM policies and directives that apply to this project.

2.2.1 Organization. The organizational context, both technical and managerial, within which the planned SCM activities are to be implemented shall be described. The Plan shall identify the following:

(1) All organizational units that participate in or are responsible for any SCM activity on the project
(2) The functional roles of these organizational units within the project structure
(3) Relationships between organizational units

Organizational units may consist of a vendor and customer, a prime contractor and subcontractors, or different groups within one organization. Organization charts, supplemented by statements of function and relationships, can be an effective way of presenting this information.

2.2.2 SCM Responsibilities. The allocation of SCM activities to organizational units shall be specified. For each activity listed within SCM activities (see 2.3), the name of the organizational unit or job title to perform this activity shall be provided. A matrix that relates the organizations defined above to the SCM functions, activities, and tasks can be useful for documenting the SCM responsibilities.

For any review board or special organization established for performing SCM activities on this project, the Plan shall describe its

(1) Purpose and objectives
(2) Membership and affiliations
(3) Period of effectivity
(4) Scope of authority
(5) Operational procedures

2.2.3 Applicable Policies, Directives, and Procedures. Any external constraints placed on the Plan by other policies, directives, and procedures shall be identified. For each, its impact and effect on the Plan shall be stated.

2.3 SCM Activities. SCM activities information identifies all functions and tasks required to manage the configuration of the software system as specified in the scope of the Plan. Both technical and managerial SCM activities shall be identified. General project activities that have SCM implications shall be described from the SCM perspective.

SCM activities are traditionally grouped into four functions: configuration identification, configuration control, status accounting, and configuration audits and reviews. The information requirements for each function are identified in 2.3.1 through 2.3.4.

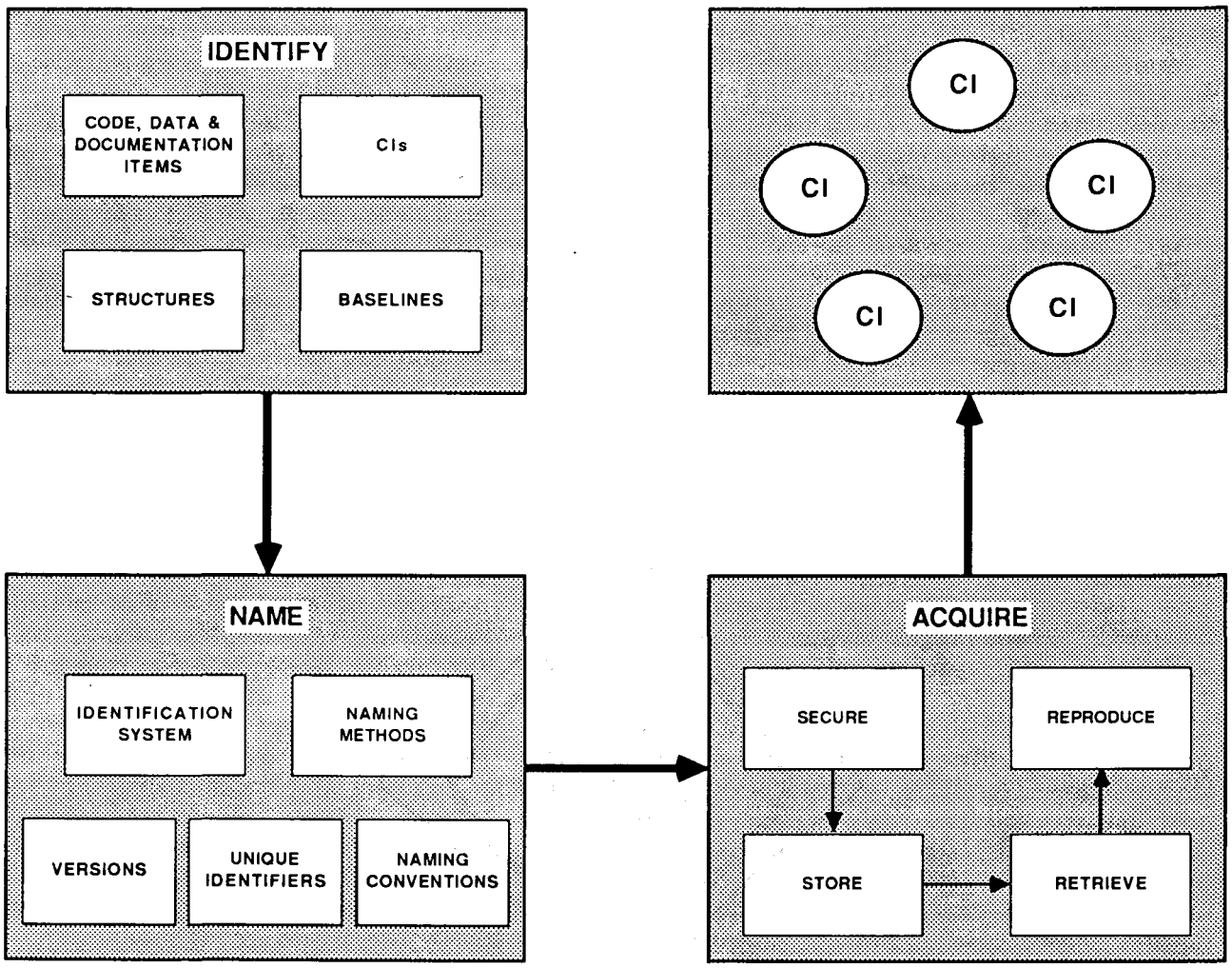


Fig 1
Configuration Identification Processes

Due to their high risk nature, the requirements for interface control and subcontractor/vendor control activities are identified separately in 2.3.5 and 2.3.6.

2.3.1 Configuration Identification. Configuration identification activities shall identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project. The documents are acquired for configuration control. Controlled items may be intermediate and final outputs (such as executable code, source code, user documentation, program listings, data bases, test cases, test plans, specifications, and management plans) and elements of the support environment (such as compilers, operating systems, programming tools, and test beds).

The Plan shall identify the project configuration items (CI) and their structures at each project control point. The Plan states how each CI and its versions are to be uniquely named and describes the activities performed to define, track, store, and retrieve CIs. The following sections specify information required for configuration identification (see Fig 1).

2.3.1.1 Identifying Configuration Items. The Plan shall record the items to be controlled, the project CIs, and their definitions as they evolve or are selected. The Plan shall also describe how the list of items and the structures are to be maintained for the project.

As a minimum, all CIs that are to be delivered shall be listed.

Appropriate baselines shall be defined at control points within the project life cycle in terms of the following:

(1) The event that creates the baseline
(2) The items that are to be controlled in the baseline
(3) The procedures used to establish and change the baseline
(4) The authority required to approve changes to the approved baselined documents

A means of identifying changes and associating them with the affected CIs and the related baseline shall be specified.

2.3.1.2 Naming Configuration Items. The Plan shall specify an identification system for assigning unique identifiers to each item to be controlled. It shall also specify how different versions of each are to be uniquely identified. Identification methods could include naming conventions and version numbers and letters.

The Plan shall describe the methods for naming controlled items for purposes of storage, retrieval, tracking, reproduction, and distribution. Activities may include version marking, labeling of documentation and executable software, serialization and altered item marking for executable code or data embedded on a microchip, and identification of physical packaging.

Subcontracted software, vendor proprietary software, and support software may require special identification schemes and labeling.

2.3.1.3 Acquiring Configuration Items. The Plan shall identify the controlled software libraries for the project and describe how the code, documentation, and data of the identified baselines are to be physically placed under control in the appropriate library. For each library the format, location, documentation requirements, receiving and inspection requirements, and access control procedures shall be specified.

The Plan shall specify procedures for the actual storage of documents and magnetic media, including the physical marking and labeling of items. Data retention periods and disaster prevention and recovery procedures may also be described.

Procedures shall describe how to retrieve and reproduce controlled items from library storage. These activities include verification of marking and labeling, tracking of controlled copies, and protection of proprietary and security information.

2.3.2 Configuration Control. Configuration control activities request, evaluate, approve or disapprove, and implement changes to baselined CIs. Changes encompass both error correction and enhancement. The degree of formality necessary for the change process depends on the project baseline affected and on the impact of the change within the configuration structure.

For each project software library identified according to 2.3.1.3, the Plan shall describe the change controls imposed on the baselined CIs. The Plan shall define the following sequence of specific steps:

(1) Identification and documentation of the need for a change
(2) Analysis and evaluation of a change request
(3) Approval or disapproval of a request
(4) Verification, implementation, and release of a change

The Plan shall identify the records to be used for tracking and documenting this sequence of steps for each change. Any differences in handling changes based on the origin of the request shall be explicitly documented.

2.3.2.1 Requesting Changes. The Plan shall specify the procedures for requesting a change to a baselined CI and the information to be documented for the request. As a minimum, the information recorded for a proposed change shall contain the following:

(1) The name(s) and version(s) of the CIs where the problem appears
(2) Originator's name and organization
(3) Date of request
(4) Indication of urgency
(5) The need for the change
(6) Description of the requested change

Additional information, such as priority or classification, may be included to clarify the significance of the request and to assist in its analysis and evaluation. Other information, such as change request number, status, and disposition, may be recorded for change tracking.

2.3.2.2 Evaluating Changes. The Plan shall specify the analysis required to determine the impact of the proposed change and the procedures for reviewing the results of the

For any CCB established to control interfaces, the Plan shall identify its responsibilities and procedures as specified in 2.2.2.

2.3.6 Subcontractor/Vendor Control. Subcontractor/vendor control activities incorporate items developed outside the project environment into the project CIs. Included are software developed by contract and software acquired in its finished form. Special attention should be directed to these SCM activities due to the added organizational and legal relationships.

For both subcontracted and acquired software, the Plan shall define the activities to incorporate the externally developed items into the project CIs and to coordinate changes to these items with their development organizations.

For subcontracted software, the Plan shall describe the following:

(1) What SCM requirements, including an SCM Plan, are to be part of the subcontractor's agreement
(2) How the subcontractor will be monitored for compliance
(3) What configuration audits and reviews of subcontractor items will be held
(4) How external code, documentation, and data will be tested, verified, accepted, and merged with the project software
(5) How proprietary items will be handled for security of information and traceability of ownership (e.g., copyright and royalties)
(6) How changes are to be processed, including the subcontractor's participation

For acquired software, the Plan shall describe how the software will be received, tested, and placed under SCM; how changes to the supplier's software are to be processed; and whether and how the supplier will participate in the project's change management process. Acquired software can come from a vendor, a subcontractor, a customer, another project, or other source.

2.4 SCM Schedules. SCM schedule information establishes the sequence and coordination for the identified SCM activities and for all events affecting the Plan's implementation.

The Plan shall state the sequence and dependencies among all SCM activities and the relationship of key SCM activities to project milestones or events. The schedule shall cover the duration of the Plan and contain all major milestones of the project related to SCM activities. SCM milestones shall include establishment of a configuration baseline, implementation of change control procedures, and the start and completion dates for a configuration audit.

Schedule information shall be expressed as absolute dates, as dates relative to either SCM or project milestones, or as a simple sequence of events. Graphic representation can be particularly appropriate for conveying this information.

2.5 SCM Resources. SCM resource information identifies the software tools, techniques, equipment, personnel, and training necessary for the implementation of the specified SCM activities.

SCM can be performed by a combination of software tools and manual procedures. Tools can be SCM-specific or embedded in general project aids; they can be standard organizational resources or ones specially acquired or built for this project. Tools can be applied to library structure and access control; documentation development and tracking; code control; baseline system generation; change processing, communication and authorization; change/problem tracking and status reporting; archiving, retention, and retrieval of controlled items; or the SCM planning process itself.

For each type of SCM activity identified, the Plan shall specify what tools, techniques, equipment, personnel, and training are required and how each resource will be provided or obtained.

For each software tool, whether developed within the project or brought in from outside the project, the Plan shall describe or reference its functions and shall identify the configuration controls to be placed on the tool.

2.6 SCM Plan Maintenance. SCM plan maintenance information identifies the activities and responsibilities necessary to ensure continued SCM planning during the life cycle of the project. The Plan shall state the following:

(1) Who is responsible for monitoring the Plan
(2) How frequently updates are to be performed
(3) How changes to the Plan are to be evaluated and approved
(4) How changes to the Plan are to be made and communicated

The Plan should be reviewed at the start of each project software phase, changed accordingly, and approved and distributed to the project team.

If the Plan has been constructed with detailed procedures documented elsewhere in appendixes or references, different maintenance mechanisms for those procedures may be appropriate.

3. Tailoring of the Plan

This standard permits significant flexibility in preparing an SCM Plan. A successful Plan reflects its project environment. It should be written in terms familiar to its users and should be consistent with the development and procurement processes of the project.

To conform to the requirements set forth in other applicable standards or to accommodate local practices, a Plan may be tailored upward, to add information, or tailored to use a specified format. The Plan may also be tailored downward, omitting information required by this standard, when specific standard requirements are identified as not applicable to this project.

3.1 Upward Tailoring. Some information requirements applicable to a particular project may not be stated in this standard due to its scope of establishing the minimum required contents of an SCM Plan. If additional requirements are applicable to the project, the Plan shall so state these additions as part of the Introduction and indicate the reason for their insertion. A cost-benefits analysis should be completed for each additional requirement. Requirements that are additional should be agreed on by all affected project functions and the parties responsible for approval of the plan.

3.2 Downward Tailoring. Some information requirements stated in this standard may not apply to a particular project due to the project's limited scope, low complexity, or unusual environment. If a requirement is not applicable to the project, the Plan shall so state this deletion as part of the Introduction and indicate the reason for removal. Requirements that are inapplicable should be agreed upon by all affected project functions and all parties responsible for approval of the Plan.

The Plan shall omit none of the six major classes of information. Detailed information may be omitted as indicated above but within the limits of the consistency criteria stated in Section 4.

If certain information has not been decided on or is unavailable at the time the Plan is initially approved, the Plan shall mark those areas or sections as "to be determined" and shall indicate, as part of Plan maintenance, information on how and when further information will be provided.

3.3 Format. The information may be presented in the Plan in any sequence or presentation style deemed suitable for the Plan's users. To achieve consistency and convenience within a single organization or industry segment, a standard format for SCM plans is desirable and appropriate. To customize this standard for a particular group of users, a supplement to the standard specifying Plan structure and standard terminology may be used.

4. Conformance to the Standard

An SCM Plan shall satisfy the following criteria in order to conform with this standard.

4.1 Minimum Information. The Plan shall include the six classes of SCM information identified in Section 2: Introduction, Management, Activities, Schedules, Resources, and Plan Maintenance. Within each class, all of the required information stated in Section 2 of this standard, as indicated by the words "shall" and "required," shall be documented within the Plan. If certain required information is not applicable, the reasons shall be so stated. If a sequence of information other than the sequence of this standard is used, an explicit cross reference

between the Plan and the standard shall be provided.

4.2 Presentation Format. One document, section title, or such reference shall exist that is specifically labeled "Software Configuration Management Plan." Within this document, each of the six classes of information shall be included. While the information may be provided in a number of presentation styles, the requirement is to provide all Plan information and references in a single document.

4.3 Consistency Criteria. The documented information shall satisfy the following consistency criteria:

(1) All activities defined in the Plan (see 2.3.1 to 2.3.6) shall be assigned to an organizational unit (see 2.2.2).
(2) All activities defined shall have resources identified to accomplish the activities (see 2.5).
(3) All CIs identified in the Plan (see 2.3.1) shall have defined processes for baseline establishment and change control (see 2.3.2).

4.4 Conformance Declaration. If the preceding criteria are met, then the conformance of any SCM planning documentation with this standard may be stated accordingly: "This SCM Plan conforms with the requirements of IEEE Std 828-1990."

Appendix
Cross Reference to IEEE Std 1042-1987

(This Appendix is not part of IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans, but is included for information only.)

Section in IEEE Std 828-1990	Section in IEEE Std 1042-1987
1. Introduction to the Standard	1. Introduction 2. SCM Disciplines in Software Management
2. The SCM Plan	3. Software Configuration Management Plans
2.1 Introduction	3.1 Introduction
2.2 SCM Management	3.2 Management
2.3 SCM Activities	3.3 SCM Activities
2.3.1 Configuration Identification	3.3.1 Configuration Identification
2.3.2 Configuration Control	3.3.2 Configuration Control
2.3.3 Configuration Status Accounting	3.3.3 Configuration Status Accounting
2.3.4 Configuration Audits and Reviews	3.3.4 Audits and Reviews
2.3.5 Interface Control	3.2.3 Interface Control
2.3.6 Subcontractor/Vendor Control	3.5 Supplier Control
2.4 Schedules	3.2.4 SCM Plan Implementation
2.5 Resources	3.4 Tools, Techniques and Methodologies
2.6 SCM Plan Maintenance	2.5 The Planning of SCM
3. Tailoring of the Plan	2.5 The Planning of SCM
4. Conformance to the Standard	2.5 The Planning of SCM

Acknowledgments

The following organizations supported working group members in the development of this standard. This support does not constitute or imply approval or endorsement of this standard.

Babcock & Wilcox
Battelle Northwest Laboratories
BTG Inc.
Compass Corporation
CTA Inc.
DSC Communications
Eaton/AIL
General Electric–MSD
Jet Propulsion Laboratory
Lockheed Missiles & Space Company
Lockheed Sanders Inc.
MACTEC
McDonnell Douglas
Mitre Corporation
Motorola/Computer X
National Institute of Standards and Technology
Northrop Corporation
Programming Environments Inc.
Rockwell Telecommunications
Texas Instruments
Unisys Corporation

Recognized as an
American National Standard (ANSI)

ANSI/IEEE
Std 1042-1987

An American National Standard

IEEE Guide to Software Configuration Management

Sponsor

Software Engineering Standards Subcommittee of the Technical Committee on Software Engineering of the IEEE Computer Society

Approved September 10, 1987
Reaffirmed December 2, 1993

IEEE Standards Board

Approved March 10, 1988

American National Standards Institute

The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017, USA

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least once every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
345 East 47th Street
New York, NY 10017
USA

Foreword

(This Foreword is not a part of ANSI/IEEE Std 1042-1987, IEEE Guide for Software Configuration Management.)

The purpose of this guide is to provide guidance in planning software configuration management (SCM) practices that are compatible with ANSI/IEEE Std 828-1983, IEEE Standard for Software Configuration Management Plans. Three groups are served by this guide: developers of software, software management community, and those responsible for preparation of SCM Plans. The developers of software will be interested in the different ways SCM can be used to support the software engineering process. The management community will be interested in how the SCM Plan can be tailored to the needs and resources of a project. Those preparing plans for SCM will be interested in the suggestions and examples for preparation of a Plan.

The introduction of this guide presents a technical and philosophical overview of the SCM planning process. Subsequent paragraphs in the body of the guide contain general statements of principles, commentary on issues to consider, and *lessons learned* for the corresponding paragraph in the outline of the ANSI/IEEE Std 828-1983 Plan. Four Appendixes illustrate how the ANSI/IEEE Std 828-1983 can be used for a variety of different projects. A fifth Appendix lists current references that may be useful in planning SCM.

This guide was prepared by a working group chartered by the Software Engineering Subcommittee of the Technical Committee on Software Engineering of the Computer Society of IEEE. This guide represents a consensus of individual working-group participants with broad expertise in software engineering and configuration management, staffed with both members within the Institute and from other groups that have expertise and interest in participating.

The following individuals contributed to the writing of this guide by attendance to two or more working sessions, or by substantial written commentary, or both.

Richard L. Van Tilburg, *Chairman*

David Schwartz, *Cochairman*

Bakul Banerjee
H. Ronald Berlack
Grazyna Bielecka
Jack L. Cardiff
Larry Cummings
Michael A. Daniels
David Gelperin
Curtis F. Jagger
Allen T. L. Jin
Dwayne Knirk
Nancy Murachanian
Sarah H. Nash
Wilma Osborne
Brian F. Rospide
Margaret Rumley
Edward Showalter
Jean Stanford
William S. Turner, III
Albert T. Williams

When the IEEE Standards Board approved this standard on September 10, 1987, it had the following membership:

Donald C. Fleckenstein, *Chairman*

Marco W. Migliaro, *Vice Chairman*

Andrew G. Salem, *Secretary*

James H. Beall
Dennis Bodson
Marshall L. Cain
James M. Daly
Stephen R. Dillon
Eugene P. Fogarty
Jay Forster
Kenneth D. Hendrix
Irvin N. Howell
Leslie R. Kerr
Jack Kinn
Irving Kolodny
Joseph L. Koepfinger*
Edward Lohse
John May
Lawrence V. McCall
L. Bruce McClung
Donald T. Michael*
L. John Rankine
John P. Riganati
Gary S. Robinson
Frank L. Rose
Robert E. Rountree
William R. Tackaberry
William B. Wilkens
Helen M. Wood

*Member emeritus

The following person were on the balloting committee that approved this document for submission to the IEEE Standards Board:

A. Frank Ackerman
Richard L. Aurbach
Motoei Azuma
H. Jack Barnard
Roy Bass
James Behm
H. R. Berlack
Michael A. Blackledge
Gilles Bracon
Kathleen L. Briggs
A. Winsor Brown
William L. Bryan
Fletcher Buckley
Lorie J. Call
Harry Carl
John Center
T. S. Chow
J. K. Chung
Won L. Chung
Antonio M. Cicu
Francos Coallier
Peter Cond, Jr
Christopher Cook
Richard Cotter
Arthur N. Damask
Taz Daughtrey
Peter A. Denny
Fred M. Discenzo
William P. Dupras
Robert E. Dwyer
Mary L. Eads
W. D. Ehrenberger
L. G. Egan
Walter J. Ellis
Caroline L. Evans
David W. Favor
Joan Feld
John W. Fendrich
Glenn S. Fields
A. M. Foley
Joel J. Forman
Julian Foster
Crespo Fuentes
F. K. Gardner
Leonard B. Gardner
David Gelperin
Anne K. Geraci
Shirley Gloss-Soler
J. G. Glynn
Andrej Grebnec
Benjamin W. Green
Victor M. Guarnera
Lawrence M. Gunther
David A. Gustafson
G. B. Hawthorne
John W. Horch
Cheng Hu
Harry Kalmbach
Myron S. Karasik
Dwayne L. Knirk
Shaye Koenig
George Konstantinow
Joseph A. Krupinski
Joan Kundig
T. M. Kurihara
Lak Ming Lam
John B. Lane
Robert A. C. Lane
Gregory N. Larsen
Ming-Kin Leung
F. C. Lim
Bertil Lindberg
Austin J. Maher
Paulo Cesar Marcondes
C. D. Marsh
Roger J. Martin
John McArdle
Russell McDowell
W. F. Michell
Manijeh Mogh
Charles S. Mooney
George Morrone
D. D. Morton
G. T. Morum
Hironobu Nagano
Gerry Neidhart
Dennis Nickle
Wilma M. Osborne
Thomas D. Parish
David E. Peercy
Michael T. Perkins
John Petraglia
Donald J. Pfeiffer
I. C. Pyle
Thomas S. Radi
Salim Ramji
Jean-Claude Rault
Meir Razy
Donald Reifer
John C. Rowe
Julio Gonzalez Sanz
Stephen R. Schach
Lee O. Schmidt
N. Schneidewind
Wolf A. Schnoege
Robert Schueppert
David J. Schultz
Gregory D. Schumacher
Leonard W. Seagren
Robert W. Shillato
David M. Siefert
Jacob Slonim
Harry M. Sneed
V. Srinivas
Manfred P. Stael
Wayne G. Staley
Franklin M. Sterling
Mary Jane Stoughton
William G. Sutcliffe
Richard H. Thayer
Bob Thibodeau
Paul U. Thompson
Terrence L. Tillmanns
G. R. Treble
Henry J. Trochesset
C. L. Troyanowski
William S. Turner III
W. T. Valentin, Jr
R. I. Van Tilburg
Tom Vollman
Dolores R. Wallace
Martha G. Walsh
John P. Walter
Andrew H. Weigel
Peter J. Weyman
G. Allen Whittaker
Patrick J. Wilson
David L. Winningham
W. M. Wong
Dennis L. Wood
Nancy Yavne
William W. Young
Janusz Zalewski
Donald Zeleny
Hugh Zettel
Peter F. Zoll

Acknowledgment

Appreciation is expressed to the following companies and organizations for contributing the time of their employees to make possible the development of this text:

Boeing
Burroughs
General Dynamics
Hughes Aircraft Co
Intel Corporation
IBM
Goodyear Atomic Corporation
GTE
National Bureau of Standards
MITRE
Motorola
Programming Environments, Inc
RCA Astro Electronics
Sperry
Telos
Texas Instruments
ZTROW Software Inc

Contents

Contents

An American National Standard

IEEE Guide to 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.

[1] The 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]

[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) baselies

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.

[2] ANSI/IEEE publications are available from IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855-1331 and from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018.

[3] A 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 entitites 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 change[4] (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 software 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

[4] Even 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.

Firmware[5] 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 by 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 assembly 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 *configuration 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.

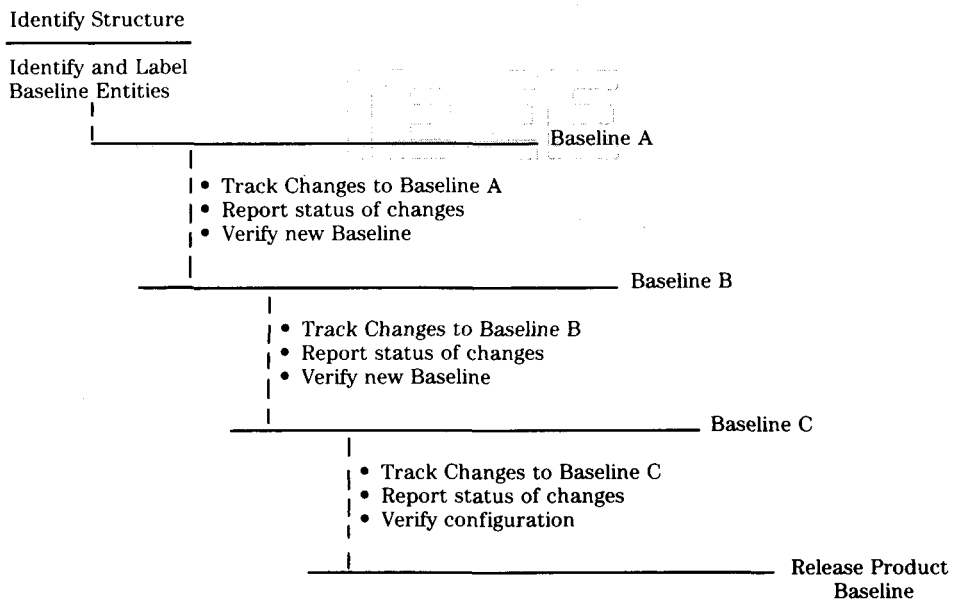


Fig 1
Model of Change Management

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, text 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

[6]There 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.

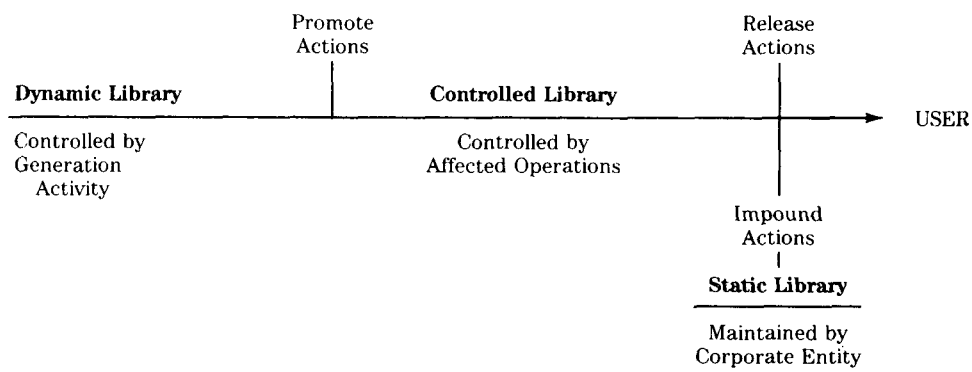


Fig 2
Three Types of Libraries

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 products: 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

[7] Reference: British Alvey Programme.

[8] Reference: *Life Cycle Support in the Ada*[9] *Environment* by Mc Dermid and Ripken.

[9] Ada 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 associates 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 Plans

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 program[10]
(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?

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.

[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.

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 forimplementing 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, subcontractor 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 throughout 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.

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 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.

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 required[11] 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?

[11] The 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.

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?
(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.

3.3.2 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.

It may be noted that the CCB concept is another one of those functional concepts of SCM. On a small project, the CCB could be the *chief programmer* and the system will function quite adequately.

Any other change approval activities, such as the SCCB that supports the CCB, also needs to be identified and their roles defined. In some installations, the CCB may need to have the technical expertise to make the final decision on whether a requested change is technically feasible. Other CCB must be supported by technical experts or be prepared to delegate a level of change authorization to qualified subordinate bodies. *In general, decision making that affects the allocation and scheduling of development or maintenance resources should be separated from decision making motivated by various technical and marketing issues.*

3.3.2.4 Interface With Other CCB. Large or complex systems can have many hardware-software interfaces (as documented in the Interface Control [2.3] subsection of the Plan) that require continued ongoing change coordination. Sometimes these boards are called program change review boards (PCRB). The Plan needs to include a description of how these interfaces are handled and documented so all of the people on the projects know how to get the job done (this will probably involve both formal and informal organizational structures and interfaces).

Issues to Consider in Describing CCB Interfaces

(1) Are there a number of CCB that have to work together, as illustrated in Appendix C, or is there only one that has total responsibility for the software configuration items?
(2) Is there a hierarchy of CCB that have authority for making business-type management decisions such as illustrated in Appendixes A and D?
(3) Who has the responsibility and authority for maintaining communications with these CCB?
(4) What body or authority has been designated to arbitrate deadlocks when two parallel CCB are unable to resolve an issue?
(5) What are the procedures for resolving differences of opinion?
(6) What needs to be done to maintain responsive communication and time limits on decision making?

3.3.2.5 Support Software. The theme of these paragraphs has to do with managing all the other software needed to build and maintain the computer program products throughout their life cycle. Specifically, this focus is on describing the necessary controls used to manage support software. Support software, which may be user-furnished, developed in-house, leased from a vendor or purchased off-the-shelf, is the class of software which may or may not be delivered with a product, but yet is necessary for designing, enhancing, or testing the changes made during the life of a delivered computer program product. The developer or maintainer needs to ensure that the support software is available for use for as long as may be necessary. For example, compilers need to be archived for use later *as is* when implementing enhancements to prevent subtle compiler dependencies from turning simple enhancements into major upgrades. Host systems, when used, and utility programs and test drivers are also needed.

Issues to Consider in Planning SCM of Support Software

(1) What is the total set of support software used to design, develop, and test the software controlled under this Plan?
(2) Is this set of software archived and maintained for the full life cycle of the computer program products?
(3) What procedures are to be followed to introduce new versions of support software that impact the software within the scope of the Plan?
(4) How are problems resolved and changes made in the support software that impact the configurability or maintainability of the software within the scope of the Plan?
(5) How is the hardware configuration used to develop and maintain the software product identified and maintained for the full life cycle of the computer program product?

It is necessary to determine the appropriate level of software support needed for maintenance of the product throughout its full life cycle. What is sufficient and necessary for the job but not prohibitive in terms

of support software costs for maintenance? In some situations, it can be very costly to actually maintain or enhance some of the support tools. For example, fixing bugs in a compiler may trigger unknown changes in production software after it is simply recompiled. *Whenever a production baseline is established, it is very important to archive all environment and support tools along with the production code.*

3.3.3 Configuration Status Accounting. The theme of this subsection is identifying what information is needed for various activities, obtaining the information and reporting it. The concern is with the acquisition of the right information at the right time so reports may be made when they are needed. In essence, this is a typical data management problem. Status accounting may be literally thought of as an accounting system; many of the concepts used to track the flow of funds through accounts may be used to track the flow of software through its evolution. Using this accounting analogy, separate accounts can be established for each CI. Individual transactions can then be tracked through each account as they occur. The configuration status accounting function, at a minimum, is basically reporting the transactions occurring between SCM-controlled entities.

The functional capabilities of the library system (or the software programming environment), in conjunction with the SCM tools, determine in a large way the capabilities of the status accounting function. As well as providing *live* information regarding the development process, the configuration of each released baseline needs to be documented, together with the exact configuration of the released system (that is, historical records). The definition of the *Build Standard* of systems is an important tool for maintenance teams. Because of its impact on maintaining operational software, support software must be addressed in status accounting.

Status accounting reports need to be addressed in detail in the Plan. The theme should be able to answer queries as to *What is the status of SCR 21, 37, 38, 39 and 50?* when one is not always sure of the query in advance. More sophisticated SCM tools that capture transaction data in the library data base can use data management systems and report generators to provide flexibility. Other systems need to anticipate common queries by capturing information in a form where it is easily accessible.

Issues to Consider in Planning Section 3.3 — Configuration Status Accounting

(1) What types of information needs to be reported?
(2) What is the degree of control required by the customer (typically management)?
(3) Who are the different audiences for each report?
(4) What is the formality required by the organization's standards and procedures for requesting or obtaining reports, or both?
(5) What kind of reports are needed to support integration of units and the tracing of error sources?
(6) What information is needed to produce reports?
    (a) Any problem report number included in a release or promotion
    (b) Units that have been delivered within a given time to integration and test activity
    (c) Changes made and released as a result of a particular problem report
    (d) Units that have been through various types of testing but have not been promoted or released
    (e) Units that have been promoted as a result of a design change
(7) For large systems, is there a need for handling rollover of identification sequences?

Many different types of reports can and do prove useful. The project's managers may, for example, make use of the status accounting data to keep track of the project's progress. Typically the report requests must evolve over a period of time. For some projects, status reporting can be extended to include the status of data items and reviews that are more strictly management scheduling information rather than just configuration management status.

The basic information needed by a CCB relates to transactions applied to the baseline(s), particularly the operational (product) baselines. The disciplines involved in controlling computer programs complement traditional CM for this process. Information needed for more detailed technical management between baseline events should also be

collected somehow. Interfaces with available software engineering tools can provide much of this information.

The procedure for tracking the status of CI should be established early enough in the software development process to allow data gathering when it is most easily generated (that is, at the decision and response points) rather than after the fact. The desirable amount of automation depends in large part on the tools available, the size of the project and the maturity of the existing procedures.

Status accounting for multiple sites represents a more complex reporting procedure. The sample Plan in Appendix B describes this problem. Other general requirements for reporting must anticipate management needs for various combinations of information. An ad hoc query capability is often most useful.

3.3.4 Audits and Reviews. The theme of subsection 3.4 of the Plan involves the procedures used to verify that the software product (executable code) matches the configuration item descriptions in the specifications and documents, and that the package being reviewed is complete. It should be noted that, as a general division of labor, the organization performing quality assurance functions also usually performs the audits that address change processing functions, operation of the library(ies), and other activities associated with the processes of software configuration management. This constrasts with the reviews and audits performed within the scope of a SCM activity or organization that verify that a software or firmware product is a consistent, well-defined collection of parts.

Audits are one means by which an organization can ensure that the developers have done all their work in a way that will satisfy any external obligations. Audits vary in formality and rigor, depending on the legal liability of external obligations. They are a check on the completeness of a computer program product. Any anomalies found during audits should not only be corrected but the root cause of the problem should be identified and corrected to ensure that the problem does not resurface.

Generally, there should be a physical configuration audit (PCA) and a functional configuration audit (FCA) of configuration items prior to the release of a product baseline or an updated version of a product baseline. The PCA portion of the audit consists of determining that all items identified as being part of the configuration are present in the product baseline. The audit must also establish that the correct version and revision of each part are included in the product baseline and that they correspond to information contained in the baseline's configuration status report.

The FCA portion is similar, in that someone acknowledges having inspected or tested each item to determine that it satisfies the functions defined in the specifications or contract(s) for which it was developed. The objectives of a PCA/FCA are for the developers to provide notice that contractual obligations are nearing completion, and to provide sufficient evidence for the clients or user organization to accept the product and initiate the transition into operational usage.

This section of the Plan should define ways to ensure that established configuration management procedures are followed:

(1) Test specifications are maintained current
(2) Test reports are properly prepared
(3) Test procedures explicitly define tests to be conducted
(4) Test results comply with acceptance criteria in the test procedure
(5) Test data package contents are complete and comply with approved formats

Issues to Consider in Planning Section 3.4—Audits and Reviews

(1) Are there needs or provisions for more than one audit of each product baseline?
(2) Is there a single, separate audit trail for each component and for the personnel working on them?
(3) How are subcontractors involved in an audit (if part of project)?
(4) Are provisions made for auditing the SCM process?
(5) Are periodic reviews held to determine the progress and technical quality of a computer program product?

Audits of a configuration as it evolves can prevent massive problems at the time of release for operational use.

A higher-level audit trail for business-type management that reflects the real-time relationships and status of CI changes, component changes and individuals responsible for development is often very useful. When addressing subcontractor audits, reference Section 5, Supplier Control, in the Plan.

When addressing internal audits, the Plan should identify who will be performing these audits and exactly what is to be audited. For example, the SQA group may audit the SCM group's adherence to change control procedures (assuming an SCM group exists—otherwise the general use of tools is audited).

Although SCM functions generally do not initiate or direct reviews, quite often the mechanisms used by SCM to process changes are used to organize and process items in a review conducted by other functions such as software quality assurance (SQA). The mechanisms of status reporting are often useful in maintaining detailed action items from reviews of complex systems. SCM supports reviews in this way as any other support provided to management.

There should always be an audit of the configuration items at the time a product is released. This will vary according to the baseline being released and the criteria for the audit stated in the Plan. At a minimum, when the product baseline is established and whenever it is subsequently changed due to the release of a new version of the computer program, the configuration should be audited. Again, the roles of the SCM organization and its participation in the audit should be established in the Plan.

3.3.5 Release Process. Major releases of software must be described so that the recipient understands what has just been delivered. Often the recipient will need installation instructions and other data concerning the use of the new system. The installation instructions should define the environment on which the software will run. This is to cover both hardware (for example, machine type, peripherals needed, and extra memory required) and software (for example, operating system version, and utilities not provided) environments. SCM verifies that the release package is complete and ready to be handed over to the user.

A short outline of the documentation (often referred to as the version description document, or VDD) typically associated with the release package is given in 3.3.5.1. It may be modified to suit the project. The more critical or the larger the application, the more complete the documentation needs to be.

3.3.5.1 Version Description Document. The version description document describes the tapes, diskettes, or other media used to provide the software.

(1) *Release Media.* List the labels on each tape, diskette or [EP]ROM and provide some guidance as to the contents of each volume. For example, *Tape FG301 contains the executable load unit library required to run FRED.*

When one has a more complex system with many CI and associated data files, it may be necessary to describe each file on the tape in this section.

(2) *Functional Description.* When the release contains any functions not previously released, describe them briefly to inform the users of new capabilities. This is not intended to be a user's manual—just the summation of the new capabilities.

(3) *User Considerations.* If there are any special actions the users must take in using the new release, describe them here. Examples may be changes in the use of a new function key, a special procedure needed to complete a certain action, hardware limitations, etc.

In this section, also list any open problem reports (SCR) against the system. Typically the open reports are listed by short title and number in this section. This is for user reference. It may prevent users from filing duplicate problem reports and will give them a better understanding of the system's status.

(4) *Closed Problem Reports.* List in this section all SCR closed out in this release.

(5) *Inventory.* If necessary, provide in this section an inventory of the source and executable load units and data objects (typically at the file level) that are contained in this release. This inventory is generally necessary for those systems that must be tightly controlled. The units are usually listed in alphabetical order by CI, with a designation of version, revision, and date changed. In some cases, the SCR initiating the change is listed against each unit.

(6) *Installation Instructions.* This section may be used when the installation is made at a remote site, at numerous sites or when there are special actions to be taken. The instructions should be specific for each site.

The most important aspect of writing installation instructions is to walk through each step that

the installer will have to perform and ensure that he/she will have the information necessary to perform it.

3.4 Tools, Techniques and Methodologies. The theme of Section 4 of the Plan is making it all happen—the easy way. A well planned project typically takes advantage of planning tools such as PERT charts and Gantt charts.

The audit trail reports should reflect directly back to milestones and other activities on the planning charts, thus giving management a tool for tracking progress on a project. An automated system for software configuration management may include some way of integrating these classical planning tools with the SCM data base to provide all parties (management, designers, developers, testers, quality assurance, etc) with an on-line tool for creating products and observing their current development status dynamically in real-time, correlated automatically with a predefined Plan to yield a quantitative performance-against-schedule measures. The group that is responsible for specific tools should be identified.

The tools, techniques, and methods used to implement SCM are usually discussed in terms of a (set of) libraries and the methods and techniques used to capture, store, and promote or release the items of each type of library in a controlled manner. The concept of *software library* varies according to the level of technology available for generating software. The degree to which all entities of the product are machine accessible is a rough measure of the level of automation for a particular project.

Issues to Consider in Planning Section 4—Tools, Techniques, and Methodologies

(1) What are the number and types of libraries to be established?
  (a) A dynamic library (or programmer's library)
  (b) A controlled library (or master library)
  (c) A static library (or software repository)
  (d) Other libraries
(2) What is the amount of change activity anticipated for the project?
(3) Can the SCM tools in the library be used to manage documentation and source code?
(4) What kinds and amounts of training (for example, orientation and learning time) are needed to make the tools and procedures an effective solution for the organization?

Definition and use of a minimal set of libraries are illustrated in Appendix C and in 2.3.1. These libraries can accomplish all of the necessary functions of baseline control but usually need to be supplemented with other kinds of libraries to provide the necessary flexibility for smooth operation on larger projects. The libraries have to be structured in such a way that the source code associated with a given executable unit is promoted at the same time that the executable unit is. The source and executable load unit libraries should **always** be kept in synchronization. There are numerous technical methods for achieving this, depending on the development environment and the tools available.

For run-time efficiency, it may be necessary to merge various CI executable units into an integrated run-time environment. When this is done, it is also advisable to maintain the source separately that created the load units.

Note that the corresponding data files are included in the various levels of libraries. When table driven software is used, it is critical to maintain that data at the same level as the corresponding code. This can be handled by carefully structuring the libraries and using appropriate naming conventions.

Manual SCM methods may be perfectly adequate for a small project. However, if the tools and equipment are already in place, they may well be cost effective. The characteristics of the project must guide tool selection. A small project may not need the detailed planning and overhead supported by a complex set of integrated SCM tools. Problems in turnover of software developers may make automation attractive even though its initial cost is high.

In the selection of SCM tools, one needs to consider the cost effectiveness of their use for the given project, product, or site. New SCM tools and methods may be good, but if the engineering staff does not trust them, understand them, or is unwilling to learn new ways of working, they may hinder rather than support the performing organizations in getting the job done.

Current commercially available SCM tools focus primarily on controlling source code. They are written by programmers and code is the important element in programming. Due consideration should be made to bring documentation under control of the same tools as the code. Good SCM systems work on files, and files can consist of paragraphs of a document as well as code.

The kinds of SCM tools recommended in a Plan should also be considered in relation to the probable availability of the tools for use within the project's environment. That is, one should not make the entire Plan dependent on a tool set that may never materialize.

3.5 Supplier Control. The theme of Section 5 of the Plan is how to place effective CM on the computer programs over which you have no direct CM control. Computer program suppliers are considered to fall into one of two classes:

(1) Subcontracted software, or those contractors that develop unique or dedicated software under contract to a developer
(2) Vendor software, or those contractors that provide privately developed and existing software, and bundled application software such as operating systems, compilers, word processing tools, software configuration management tools, and data-base management systems.

Issues to Consider in Planning Section 5—Supplier Control

(1) Is the product being procured to be used internally, delivered as part of your organization's product, or both?
(2) What post-delivery defect correction requirements and procedures need to be established?
(3) What changes is the purchaser permitted to make after delivery without invalidating the warranty or violating legal constraints?
(4) When should audits be performed?
  (a) When subcontractor or vendor releases parts to the buyer
  (b) After successful integration in buyer's system
(5) Is there a need to *pass through* SCM tools to a supplier or a vendor?
(6) Consider the use of *software in escrow*[12] as a method of enforcing SCM and quality
(7) What periodic reviews of the subcontractor's work will be needed?

3.5.1 Subcontractor Software. If a portion of a software development project is to be subcontracted to another organization, the responsibility for SCM is generally passed to that organization. However, the subcontractor can only be responsible for the portion of the work that his organization is tasked to perform, not for the integration of the subcontracted work with the final product.

Possible methods for integrating subcontractor SCM include

(1) Specifying or providing a library management tool and monitoring its use
(2) Letting the subcontractor(s) promote code to your software generation system and controlling it in the same fashion as is done in-house.
(3) Obtaining the source for all subcontractor deliveries and recompiling and relinking it using the buyer's software generation tools

To ease integration and maintenance, the subcontractor should be required to implement a system of configuration management based on the buyer's requirements—one that is compatible with the buyer's configuration management system or a subset thereof. Identification schemes should be compatible. A system for effectively managing interfaces should be developed. The subcontractor should have an internal configuration control system that is equivalent to the systems and procedures described by the buyer. The format and frequency of status reports also should be agreed upon.

Not all contractor-subcontractor relationships are easily identifiable. Sometimes, the contractual relationship does not afford the buyers any control over the subcontractor SCM processes and the buyer has to bound the relationship of the subcontracted software by alternate identification and by accepting the configuration as given, verified by testing the delivered product (as illustrated in Appendix C). Generally, it is possible to tailor the SCM requirements passed on to the subcontractor, using specifications or statements of work.

[12] If the executable code is the only code obtained, it may be advisable to have the supplier place the source code in *escrow* as a warranty that the source will be available if the supplier goes out of business.

Issues to Consider in Defining Subcontractor Relationships

(1) What SCM concerns need to be added to or removed from the contract?
(2) Who is responsible for auditing versus enforcing SCM for contractual products?
(3) What audits and procedures need to be established where the subcontractor has no documented SCM practices or procedures?

If the buyer's organization is developing the computer programs for a customer, the contract should be reviewed for any specific legal requirements that need to be passed on to the subcontractor, or special actions that have to be taken by the buyer to ensure the performance of the subcontractors' product.

Integration of subcontractor software is very difficult unless communication is kept open. One way is to allow subcontractor representatives to attend SCCB meetings to ensure that they are aware of all important technical issues. It may also be useful to accept incremental versions of the code for integration and test with the rest of the code, rather than waiting until the end of the development cycle.

In specifying delivery, identify all items that are to be a part of the deliverable. Possibilities include

(1) Source code
(2) Executable code
(3) Load units
(4) Data files
(5) Test cases
(6) Any JCL or other procedures used in running or creating the software
(7) Compilation listings or link-edit maps (for debugging)
(8) Documentation

Another concern for SCM is the subcontractor's actual performance to an agreed-upon Plan or statement of work. A preselection or purchase audit of the potential subcontractor's configuration management policies and procedures can provide an indication of the potential for the organization to perform satisfactorily. If possible, the buyer's software configuration management group should perform an in-process SCM audit of all project subcontractors to ensure satisfactory compliance. As part of this audit, a specific approved change should be traced through the subcontractor's system to the point of verifying the implementation.

A critical role for SCM is in the inspection (FCA/PCA) of the product as it is prepared for delivery to the buyer. This is most important as it determines the effort and resources that may be needed to integrate and maintain the product once it has been incorporated in the buyer's system. There are still compatibility problems and problems of error correction, and updates that have to be provided for even if the program is a stand-alone product (as for a compiler). If the program received is not well identified and documented, then the task of maintenance is generally increased.

3.5.2 Vendor Software. Warranties contained in purchase orders may be difficult to enforce. The specific criterion is that the vendor should furnish the computer program *media* as specified by a purchase order or as specified by the supplier's documentation referenced in the purchase order. Test documentation confirming compliance is desirable but often unavailable.

Issues to Consider in Defining Vendor Interfaces

(1) How is the vendor software identified?
(2) How are license agreements and data rights protected and enforced? Are there limitations on
  (a) Duplication of documentation
  (b) Your customer making copies of the program
(3) How will vendor support be provided over the life cycle of the computer program product being purchased?
(4) How will copyright interests be protected?
(5) How will legal copies of leased software be controlled?

The handling of vendor software can be very complex, particularly in a maintenance environment, such as described in Appendix C, where the vendor software is intermixed with internally developed software. More importantly, if you release the vendor product as a part of your organization's product, your organization may be responsible for ensuring its maintenance as part of your released product. An organization embedding third party software in a product delivered to a customer can be open to financial and

legal liabilities if a vendor fails to perform—that is, making required changes in a timely manner. One possible consideration is the use of an escrow account with a vendor agreement tied to performance of his product.

3.6 Records Collection and Retention. The theme of Section 6 of the Plan is to *keep the information necessary only for the time required.* This is another service aspect of configuration management. Good configuration management practices include maintaining copies of released material for backup and disaster protection. Also the liability and warranty provisions and responsibilities make considering the retention of test and approval records a necessity. If a master disaster recovery plan exists for the company, the Plan needs to disclose all information regarding the location of backups and records that are impounded in relation with that plan.

Records collection can also be a part of risk management. Part of the trade-off must consider whether personnel will be available to recover lost software. Trade-offs can be made concerning the cost of capturing and maintaining records versus the potential cost savings for

(1) Recovering programs in the event of a disaster for
  (a) Software developed for internal use
  (b) Delivered products for which warranty is still in effect
  (c) Support software necessary for maintaining computer program products under warranty
(2) Liability for not being able to certify the reliability of delivered products
(3) Information gathered that may lead to performance or productivity improvements in development or maintenance activities.

Record keeping begins in planning the capture of all data that needs to be maintained. In addition to all other considerations, archiving the software should be done in a manner acceptable to any legal contracts that may affect the computer programs. Static libraries, disaster planning, and storage should consider the legal status of the software involved (for example, whether it has trade secret status) and the impact on the provisions made for the care and storage of the software components. Special attention should be given to the retention of support software associated with software on target machines.

Issues to Consider in Planning Section 6—Records Collection and Retention

(1) What type of information needs to be retained?
(2) What data need to be maintained over a period of time for trend data analysis?
(3) Is all the information, support software, and equipment needed to recreate the product available from archives?
(4) Is media protected from disaster?
(5) Is there a need to maintain copies of software licensed for use and distribution?
(6) What activities need to be recorded (and data captured) for maintaining a product after the project is completed?
  (a) Copyright records
  (b) Distribution records
  (c) Benchmarks
  (d) Change history (CCB activity, SPR, etc)
  (e) Audits, reviews, status reports
(7) For whose use are the records being maintained?
  (a) Engineering
  (b) Management
  (c) Software Quality Assurance
  (d) Legal, customer
(8) How are the records to be kept?
  (a) On line versus off line
  (b) Media and format (hard copy document versus electronic media, deterioration rate versus time needed)
  (c) Location (preservation conditions, accessibility both off site and on site)
  (d) Tools used on the project that affect data capture
(9) How long will the data be kept?

The information collected need not mirror that collected for hardware bit for bit. For example, serial information on production that is kept to identify configurations in hardware may not be necessary for software. Plan to keep only the data that will be of use in maintenance, disaster recovery, or which has other justification. Is the deterioration rate of the storage medium sufficient for the needed time span? Media can deteriorate in storage; also, work in-progress should be backed-up at specific intervals to protect the investment for projects that have long development periods or are of high cost.

Appendixes

(The following Appendixes are not a part of ANSI/IEEE Std 1042-1987, IEEE Guide to Software Configuration Management, but are included for information only.)

Appendix A

Software Configuration Management Plan for Critical Software for Embedded Systems

Version 1.0

Approved

_____

Mgr, SCM Dept

_____

Project Mgr

_____

Contracts

_____

Customer

Date: ___ /___ /___

Synopsis

This example contains a discussion of a hypothetical contract to provide a medium-sized real-time control system for the management of advanced vehicles. Sensors are used for input of information to the system; displays are used to support a man-machine interface. The contract for the system consists of eight software configuration items being developed concurrently with five new and seven off-the-shelf hardware configuration items. The project is expected to have at most three hundred and fifty-six personnel, with an average of thirty-four and peak of fifty software development personnel over the estimated three and a half year development cycle.

Most of the development work is performed in the contractor's main facility with some work being performed at a nearby subsidiary. Testing and acceptance is performed at the mock-up in the contractor's facility. Some commercial software is procured from a vendor for the support software and the firmware for the vehicle is subcontracted to the builder of the vehicle. This is a turnkey contract. The customer takes over all maintenance of the software after delivery of the first system.

The customer's procurement organization has a large staff for monitoring the contract and is expected to perform frequent audits. The contractor's project office wishes to minimize friction with the customer and is willing to perform most, but not all, of the necessary record keeping and in-process inspections. The configuration management department of the contractor has a long history of involvement in projects with the customer and there is a general familiarity and comfortableness in *doing business* in this manner. The software configuration management activity is relatively new but is strongly supported by the old line configuration management department.

In this environment, the software configuration management activity will be a very disciplined operation, logging and maintaining accurate records of *all* transactions against established baselines.

Contents

ATTACHMENTS PAGE

Appendix A

Software Configuration Management Plan for Critical Software for Embedded Systems

1. Introduction

This document is the Software Configuration Management (SCM) Plan for the *Critical Software for Embedded Systems* (CSES). The CSES system performs functions critical to the life and safety of human beings. The configuration management of this software during development is essential to the delivery of error-free and reliable software configuration items.

1.1 Purpose. This plan provides information on the requirements and procedures necessary for the configuration management activities of the CSES project. It identifies the software configuration management requirements and establishes the methodology for generating configuration identifiers, controlling engineering changes, maintaining status accounting, and performing audits and reviews during the design and development of software configuration items.

1.2 Scope. This plan applies to all software and associated documentation used in the production of computer programs produced under the critical software for embedded systems contract including, but not limited to, source, object, and executable load images. Software configuration items referenced in the contract and controlled by this plan include

CSES Operational System
CSES Training Program
CSES Test Program
CSES Hardware Acceptance Programs
CSES Diagnostic Software
CSES Software Support System
CSES Simulation System
CSES Utilities

The organizations involved in this project are identified in Fig 1.

This plan applies to all phases of the software development life cycle, up to and including the time of delivery to the customer. Maintenance of the software after delivery is covered by another contract.

1.3 Definitions and Mnemonics

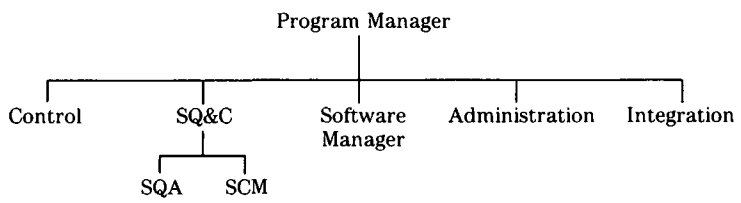
1.3.1 Definitions. The definitions used in this plan conform to the company standards as set forth in Vol II of the company *Configuration Practices Manual*. Other definitions will conform to those found in ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology. See specifically: baseline, configuration item, configuration management, configuration control, configuration control board, configuration audit, configuration identification, configuration status accounting, and software library. Unique definitions used in this document include:

interface control. The process of

(1) Identifying all functional and physical characteristics relevant to the interfacing of two or more configuration items provided by one or more organizations.
(2) Ensuring that proposed changes to these characteristics are evaluated and approved prior to implementation.

1.3.2 Mnemonics. The following mnemonics are referred to within the text of this standard:

Fig 1
Program Organization Chart



CCB	Configuration Control Board
CDR	Critical Design Review
CI	Configuration Item
CM	Configuration Management
CSES	Critical Software in Embedded System
ECN	Engineering Change Notice
FCA	Functional Configuration Audit
I&T	Integration and Test
PCA	Physical Configuration Audit
SCA	Software Change Authorization
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	Systems/Software Change Request
SQ&C	Software Quality and Control
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SRR	System Requirements Review
SSR	Software Specifications Review

1.4 References. The standards listed here will be considered when applying this plan. The latest revisions apply:

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

[2] ANSI/IEEE Std 730-1984, IEEE Standard for Software Quality Assurance Plans.

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

[4] ANSI/IEEE Std 829-1983, IEEE Standard for Software Test Documentation.

[5] Company Standard *Configuration Management Practices Manual*, Vol II.

[6] CSES Software Development Plan

Reference documents are available for use in the company library.

2. Management

2.1 Organization. The critical software for embedded systems program organization is designed to ensure clear lines of authority and to provide a framework within which administrative and technical control of software activities can be cost-effectively integrated into a quality product.

Primary responsibilities for various configuration management tasks are assigned as shown in Table 1. Within the CSES project organization, the program manager has total responsibility for the project. With this project, the program manager will have overall responsibility for configuration management of this project. The program manager serves as the project configuration control board (CCB) chairperson. The SCM project authority from the SCM organization cochairs the CCB. The SCM authority assists the program manager with planning and tailoring of the software configuration management plan (SCMP) and related CM procedures and is responsible for overseeing their implementation. The software configuration management authority reports functionally to the critical software for embedded systems program manager for the implementation of this plan. Administratively, the SCM authority reports to the SCM Department, which performs the necessary activities for the project.

Table 1
Responsibility Assignments

Responsibilities	Program Manager	Software Engineer	SCM Authority	SQA	Drafting
Configuration identification			Originate		
Approve/release tech documentation	Approve	Originate	Review	Review	
Change preparation		Originate			
Change control		Approve			
Change implementation		Approve	Review		Originate
Documentation maintenance		Approve			
Status accounting			Originate	Review	
Formal SCM audits		Approve	Originate	Review	
Baseline definition	Approve	Originate	Review	Review	Review

2.2 SCM Responsibilities. The software configuration management authority has the authority to require changes in practices and procedures that do not meet contract requirements. The general responsibilities of the software configuration management authority are outlined in Table 1. The software configuration management authority's functions include, but are not limited to the following tasks:

(1) Configuration control
(2) Status accounting
(3) Configuration identification
(4) Implementation and maintenance of the software configuration management plan
(5) Configuration control board cochairperson
(6) Establishment and maintenance of engineering baselines
(7) Cochairperson for formal audits
(8) Participation in reviews

2.2.1 Configuration Identification. Configuration identification is applied to all critical software for embedded software, both code and associated documentation. Associated documentation (that is, specifications, design documents, and program/procedure listings) along with the actual produced software makes up the configuration item. The software configuration management authority originates the identification scheme, with the approval of program management.

Configuration identification of computer programs and documentation during the development effort consists of established baselines and releases that are time-phased to the development schedules as described in the CSES software development plan.

2.2.1.1 Baselines. Baselines are established for the control of design, product, and engineering changes and are time-phased to the development effort. Baselines are established by the authority of the program manager. The software configuration management authority administers application of the baselines. Baselines defined for CSES include

(1) Functional baseline
(2) Allocated baseline
(3) Developmental baseline
(4) Product baseline

More details on baselines are presented in 2.4.2.

2.2.1.2 Releases. Throughout the development life cycle, at the discretion of the program manager, software manager, and SCM, baseline releases are performed. The releases fall into one of three categories

(1) Developer release (engineering release)
(2) Release to SCM (preliminary release)
(3) Final release (formal release to customer).

It is the responsibility of SCM to establish the release, version, and update number identifiers.

2.2.1.3 Documentation. All relevant specifications and documentation are given an identifier by SCM.

2.2.2 Configuration Control. All documentation and software entities are released to and maintained by software configuration management in a controlled library. SCM administers the change control process.

2.2.2.1 Systems/Software Change Request (SCR). The SCR is the mechanism by which change requests are presented to the CCB. This action allows a developer to check out software/documentation from SCM controlled libraries. The mechanism for requesting authorization is to present the SCR to the CCB and request approval for work to begin. The SCR form shown in Attachment A is used.

2.2.2.2 Software Change Authorization (SCA). The SCA is used to request SCM to place a new version of software/documentation into the controlled libraries. The approvals necessary are as follows: software manager, software quality assurance, and SCM. The SCA form shown in Attachment B is used.

2.2.3 Status Accounting. A software change authorization data base is used for generating reports that track changes to all of the controlled baselines. At project request, SCM generates reports that track the status of documentation and the software.

2.2.4 Audits. The SCM authority is responsible for cochairing, with the customer, all formal audits.

2.2.4.1 SQA Audits. It is the responsibility of SCM to assist SQA with their audit of the development effort. SCM maintains all documentation and software under strict controls to minimize the effort required by SQA to perform their function.

2.2.5 Configuration Control Board (CCB). The CSES project CCB is established by the program manager and SCM authority.

The program manager is the CCB chairperson and has the final responsibility for CCB actions relative to program SCM policies, plans, procedures, and interfaces. The software configuration management authority acts as cochair. In addition to the chairpersons and the CCB secretary, the CCB may include: development personnel; hardware representative; drafting representative; testing representative; customers; and always will

include a representative from software quality assurance. CCB meetings are held on a regular basis determined by the CSES program manager, or when required at the call of the CCB chairperson. The system/software change request that is generated is reviewed by the CCB and one of the following actions taken: approved, disapproved, or tabled.

2.3 Interface Control. Interface control is handled in the same manner as other types of hardware, software, or documentation. Any differences between the SQAP and the SCMP must be resolved prior to the establishment of any baselines.

2.4 SCMP Implementation. The SCMP is implemented as soon as it is signed off by the CSES program manager but prior to holding any formal reviews with the customer. Any unresolved issues found once the SCMP is written must be resolved as soon as possible during the development period and prior to any baselines being established.

2.4.1 Configuration Control Board. The CCB is established at the time of SCMP approval but prior to the establishment of any baselines.

2.4.2 Configuration Baselines. Baselines are established by the following events:

2.4.2.1 Functional Baseline. The functional baselines are established by the acceptance, or customer approval of the CSES system/segment specification. Normally this occurs at the completion of the CSES system requirement review (SRR).

2.4.2.2 Allocated Baseline. The allocated baseline is established with the customer approval of the CSES software requirement specification. Normally this corresponds to the completion of the software specification review (SSR). The specification(s) and associated documentation define the allocated configuration identification.

2.4.2.3 Developmental Baseline. The developmental baseline is established by the approval of technical documentation that defines the top-level design and detailed design (including documentation of interfaces and data bases for the computer software). Normally, this corresponds to the time frame spanning the preliminary design review (PDR) and the critical design review (CDR).

2.4.2.4 Product Baseline. The product baseline is established upon customer approval of the product specification following completion of the last formal audit (FCA).

2.4.3 Schedules and Procedures for SCM Reviews and Audits. Reviews and audits are held as defined by CSES software development plan.

2.4.4 Configuration Management of Software Development Tools. The configurations of all support software used in development and test on the CSES project software is controlled in the same manner as the critical software. Nondeliverable support software baselines do not need customer approval.

2.5 Applicable Policies, Directives, and Procedures. The complete SCM policies, directives, and procedures that apply to this program are included as part of the procedures section of this document or are part of the referenced documents or one of the appendixes.

3. SCM Activities

3.1 Configuration Identification

3.1.1 Documentation. All supporting documentation generated for this project is identified by the use of the following convention: CSES, an abbreviation for the document nomenclature, a unique four digit number assigned by the CSES software configuration manager, and the product's version-revision-update number.
EXAMPLE: CSES-SDP-0024-1.2.1

Table 2
Baseline Objectives

Baseline	Purpose	Reviews & Audits
Functional	Functions established	SRR
Allocated	Requirement defined	SSR
Developmental	Top level design complete	PDR
Developmental	Detailed design complete	CDR
Product	Approval of product spec	FCA/PCA

Document Nomenclature	Mnemonic
Software Configuration Management Plan	SCMP
Software Detailed Design Document	SDD
Software Development Plan	SDP
Software Test Procedures	SPP
Software Product Specification	SPS
Software Quality Assurance Plan	SQAP
Software Requirements Specification	SRS
Software System Specification	SSS
Software Top-Level Design Document	STD
Software Test Plan	STP
Software Test Report	STR

3.1.2 Software Parts. The software configuration items, components, and units are identified by unique identification labels.

3.1.3 Configuration Identification of the Functional Baseline. The functional baseline is identified by the approval of the CSES system segment specification.

3.1.4 Configuration Identification of the Allocated Baseline. The allocated baseline is identified by the approval of the software requirement specification.

3.1.5 Configuration Identification of the Developmental Baselines. The developmental baselines are identified by the approved technical documentation that defines the top level design and detailed designs. The process by which the initial developmental baselines are established is shown in Attachment C, Create Initial Baseline.

3.1.6 Configuration Identification of the Product Baseline. The product baseline is identified by the approval of the CSES software product specification. This baseline specification is made up of the top level specification, detailed design specification, and the computer listings.

3.2 Configuration Control. Software configuration management and change control is applied to all documents and code, including CSES critical operational software and support software. Control is effected through the implementation of configuration identification, the CCB, change control, and status accounting functions.

3.2.1 Function of the Configuration Control Board. The configuration control board reviews proposed changes for assuring compliance with approved specifications and designs, and evaluates impacts on existing software. Each engineering change or problem report that is initiated against a formally identified configuration item is evaluated by the CCB to determine its necessity and impact. The CCB members electronically sign the document to indicate that they have reviewed the changes and provided their recommendations to the chairperson. The CCB approves, disapproves, or tables all changes. The mechanism for submitting changes to the software or documentation is the systems/software change request.

3.2.2 The System/Software Change Request. The SCR system is one of the major tools for identifying and coordinating changes to software and documentation. The SCR system is a minicomputer based tool used to track the status of a change from its proposal to its eventual disposition and assist in documenting important information about the change. The SCR form (Attachment A) contains a narrative description of the change or problem, information to identify the source of the report and some basic information to aid in evaluating the report. SCR is submitted only against baselined software or documentation. SCR may be submitted by anyone associated with the project effort or its products, but usually is submitted by a member of the software development team. SCM provides the single point for receiving and processing SCR. SCM, using the report writer feature of the SCR system, is capable of producing reports that provide change control tracking. A SCR is *closed* when

(1) Integration testing has shown that the changes have been correctly made
(2) No unexpected side-effects have resulted from making the change
(3) Documentation has been updated and reviewed

3.2.3 Software Change Authorization. The software change authorization form (Attachment B) is used to control changes to all documents and software under SCM control and for documents and software that have been released to SCM. The SCA is an on-line form that the software developers use to submit changes to software and documents to SCM for updating the master library. Approvals required for baselining or updating baselined software are as follows. The developer(s) first obtain the manager's signature, I&T signature, and an SCM signature. These approvals can either be written or added electronically. SCM signature testifies that the action has occurred. SQA signature signifies that they have verified that the change has been incorporated. SCM notifies the software developer through the electronic mail system that the change has occurred so the developer can delete extra copies of the changed parts. The SCA data base, along with the SCR data base, is used for status accounting needs.

The process by which changes are made is shown in Attachment D, change procedure.

3.2.4 Change Control Automated SCM Tools. The libraries of the CSES system are used to control all textual files containing the specifications, documentation, test plans and procedures, and source code. The support software (listed below) is also under configuration management by SCM. The library structure that is used is as follows:

(1) The CSES master library
(2) The program library
(3) The development library

3.2.4.1 For this mini-computer based development effort, the change control tools are as follows:

(1) *The Source Management System* The mechanism for creating and maintaining delta files (changes only) for the CSES master library. Only SCM has access to the CSES master libraries. The CSES master library data base is accessible by the SCM status accounting system.
(2) *The Package Management System* is used to automate the build process and is used to assist SCM with the generation of software.
(3) *SCM Get* is the function invoked by software developers to acquire software modules, or parts from the program libraries.
(4) *SCM Send* is the function invoked by software developers to impound a software module into the SCM program libraries. The use of this function implicitly and automatically generates an SCA.

3.3 Configuration Status Accounting. The status accounting system is capable of generating the following reports:

(1) *Report 1.* A list of all SCR with a status of *not closed* (that is, the same as *open*)
(2) *Report 2.* A cross-reference of SCA, engineering change notices (ECN), and drawings, per SCA
(3) *Report 3.* A monthly summary of the SCR and SCA data bases
(4) *Report 4.* A total of all SCR submitted per unit within a user-selected range of submittal dates
(5) *Report 5.* A list of all SCR which are *open*, *closed*, or *all* (selected by the user)
(6) *Report 6.* A summary of all SCR submitted by unit
(7) *Report 7.* A summary of the current approval status of all SCR with a status of *not closed*
(8) *Report 8.* A short summary of all SCR within a particular software component with a status of either open, closed, or all
(9) *Report 9.* A version description document
(10) *Report 10.* A report that gives the status of all documentation under SCM control
(11) *General Report.* Allows the user to define his/her own reports. The user must first specify which fields to include in the report.
(12) *On-Line Inquiry.* Allows the user to interactively view fields within the SCM data bases. The user specifies the fields that he/she wishes displayed and conditions for searching the data base

3.4 Audits and Reviews. The SCM authority co-chairs, with the customer, the formal CM audits: the functional configuration audit (FCA) and the physical configuration audit (PCA).

3.4.1 Functional Configuration Audit. The functional configuration audit is performed on the software configuration items when the acceptance tests have been completed. Both the functional baseline and the allocated baselines have previously been approved by the customer.

The audit is made on the formal test plans, descriptions, and procedures and compared against the official test data. The results are checked for completeness and accuracy. Deficiencies are documented and made a part of the FCA minutes.

Completion dates for all discrepancies are clearly established and documented. An audit of both draft and final test reports is performed to validate that the reports are accurate and completely describe the development tests.

Preliminary and critical design review minutes are examined to assure that all findings have been incorporated and completed.

3.4.2 Physical Configuration Audit. A physical examination is made of the CI to verify that the first article conforms *as-built* to its technical documentation. The SCM authority assembles and makes available to the PCA team at the time of the audit all data describing the item configuration. This includes a current set of listings and the final draft of the product baseline specifications. Customer acceptance or rejection of the CI and the CI product specification presented for the PCA is furnished to the project manager in writing by the responsible customer representative after completion of the PCA.

3.4.3 Reviews. The SCM authority participates in all formal reviews with the customer.

In addition, the SCM activity conducts two informal audits of the developing CI during the

development cycle. The first informal audit is just prior to CDR. The second informal audit is performed at the discretion of the SCM authority midpoint between the CDR and final acceptance test.

4. Tools, Techniques, and Methodologies

4.1 Configuration Control Tools. An integrated set of SCM tools is used for configuration control and status accounting for this project. The particular tools are as follows:

(1) *Source Management System* (SMS). This tool is a file system for checking out vendor-supplied and internal software. A license agreement has been purchased from the vendor of this tool for use on this project.

(2) *Package Management System* (PMS). This tool is a vendor supplied data management tool used to automatically generate software. A license agreement has been secured from the vendor for use on this project.

(3) *Systems/Software Change Request Tool.* This is a proprietary piece of CSES software. This tool has two parts: the input form and its data base.

(4) *Software Change Authorization* (SCA) *Tool.* The SCA is a proprietary piece of CSES software. This tool has two parts: the input form and its data base.

(5) *Status Accounting Report Generator Tool.* This is a proprietary piece of CSES software. This is a report generation tool that gathers input from the following subsystems:
   (a) Source management system
   (b) Package management system
   (c) System/software change request
   (d) Software change authorization

5. Supplier Control

5.1 Vendor-Provided Software. Vendor-provided software that is to be used by this project must conform to *good business practice* SCM. The vendor provides to this project a copy of its SCM Plan for evaluation. This project must ensure that the vendor SCM system is adequate. If the vendor system is found to be inadequate, or if no vendor SCM Plan is available, then at the program manager's discretion, the vendor can be disqualified from providing software for this project.

5.2 Subcontracted Software. Any subcontractor wishing to do business with this project must provide a copy of its SCMP for evaluation by project SCM or agree to follow and abide by this SCMP. If the subcontractor SCMP is found inadequate, all other provisions of this SCMP apply. Any subcontractor not willing to abide by the above provision may be disqualified at the program manager's discretion.

5.3 Vendor and Subcontractor Software. All vendors and subcontractors are audited for compliance with *good business practice* SCM. The frequency and methods of audits are determined by the size, dollar value, and critical nature of the software.

6. Records Collection and Retention

All formal documentation produced for and by this project is retained and safeguarded for 20 years. A second copy of all software and documentation is stored in an off-site facility. This off-site facility is 21 mi from primary storage.

Attachment A
System/Software Change Request

SCR NUM.: _____

1. Submitted by: _____ DATE: ____/____/____

   Project Name: _____

2. Software Program/Document Name: _____

   Version/Revision _____

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

4. Short Task Description:

5. Detail Description:

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

7. CCB Action: _____ CCB Priority [ ]

8. Assigned to: _____ Target Release Date _____

9. Solution Comments:

10. Software Programs affected:

11. I&T Approval _____ Date: ____/____/____

    SCM Approval _____ Date: ____/____/____

12. Actual Release _____ Date: ____/____/____

13. Closed by: _____ Date: ____/____/____

    SCA Reference No: _____

14. SQA Approval: _____ Date: ____/____/____

Attachment B
Software Change Authorization

SCA Number: XXXXXX
Sheet Number: 1

Submitter: _____ System: _____ Date: ____/____/____ Time: ____/____/____ 00:00:00

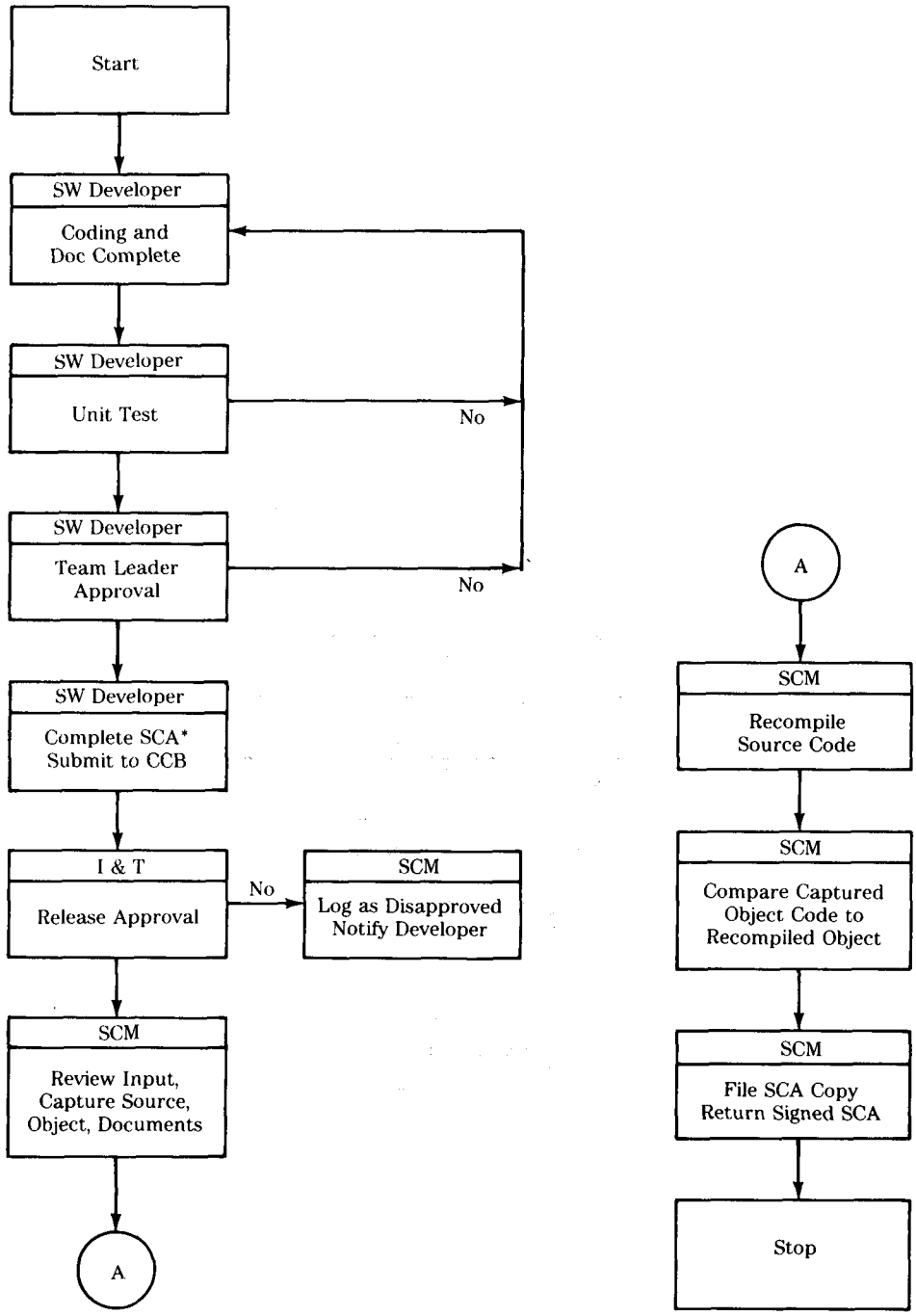
Product Version ID: _____ Computer Name _____

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

Comments: _____

Approvals	I & T	SCM	SQA
Signature			
Date			

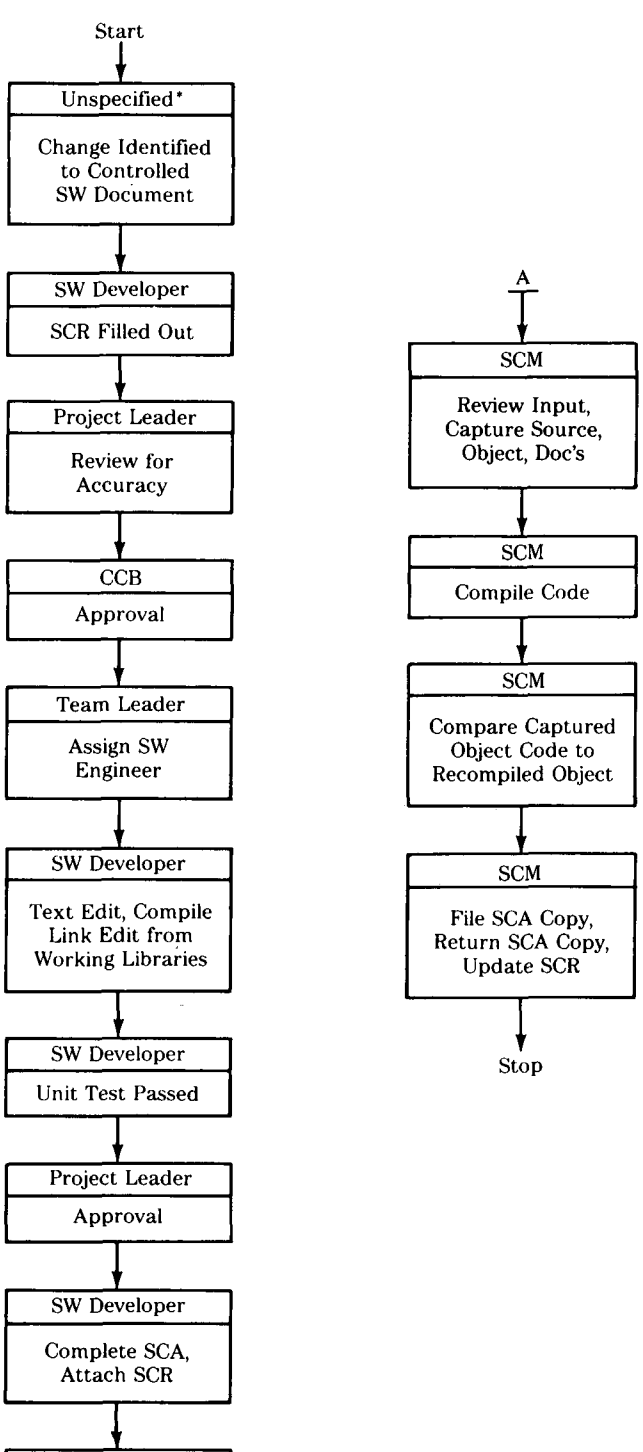
Attachment C



*NOTE: Developer's work files are retained until an approved SCA is received from SCM.

Fig 1
CSES Procedure for Creating Initial Baseline

Attachment D



NOTE: If type of change is unspecified, submit SCR to SW Development.

Fig 1
CSES Procedures for Changes to Controlled Software/Documentation

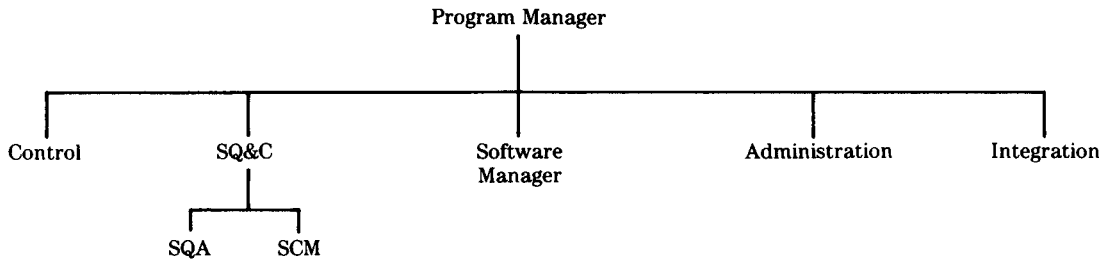


Fig 2
Program Organization Chart

Appendix B

Software Configuration Management Plan for Experimental Development Small System

Version 1.0

Approved

_____

Project Manager

_____

SCM Manager

_____

Customer

Date: ___ /___ /___

Synopsis

This example contains a hypothetical contract to provide a prototype minicomputer-based system for a research-oriented customer. The system consists of three software programs, to be developed by a project team of twenty persons (of which ten are programmers) and is considered a prototype for installation in one field site. The software is written in COBOL. If the system is considered successful at that site, it will be expanded to an additional two sites for further evaluation. These sites may be supported by different hardware (for example, a transition may have to be made from hardware configuration A to hardware configuration B) or by different versions of the code (for example, one site may be a data input-processing installation and another a centralized data-gathering installation; each may use slightly different logic or data elements). The development time frame for the prototype system is two years. The expected life of the system is not known as the production system becomes part of a major procurement sometime in the future when the management of the first three sites agree on the requirements.

The contracting company and the customer are very end-user oriented, willing to sacrifice rigor in configuration management and specifications in the interest of speedy delivery of software to the sites and rapid response to changes. Because of this orientation, the configuration control board functions are administered by the project manager alone. All change requests are reviewed by the manager and an immediate ruling is made as to whether and when to implement them. The project manager meets with the customer technical representative regularly to review change requests that require consultation, making disposition of the requests quite rapid.

In this environment, the software configuration management (SCM) activity must be very supportive of the customer and manager or all SCM records will be lost. The SCM coordinator attends meetings between users and the project staff and prepares change requests on the spot. These are provided to the project manager and customer technical representative for resolution. The project emphasis is on intensive support to management in performing SCM—it is literally transparent to management since the SCM organization completes all of the required paperwork. The managers' and customers' responsibility is to review and authorize the resulting documentation.

Contents

Appendix B

Software Configuration Management Plan for Experimental Development Small System

1. Introduction

This document describes the software configuration management activities to be performed in support of the Experimental Development Small System (EDSS) Project. The EDSS project is charged with developing and demonstrating an advanced data processing concept, which, at a later date, may be converted to a fully functional system for processing special data. The project is considered to be a research/development program.

1.1 Purpose of the Plan. The software configuration management plan (SCMP) for the EDSS system describes how the software development activity supports EDSS management in the rapid iteration of software builds necessary for efficient development of the prototype demonstration software at site A. It also describes how this demonstration baseline is to be captured to provide for adaptation of the operational program to sites B and C and for subsequent up-grade of the software to full production quality for support of operational sites.

1.2 Scope. Three software configuration items (CI) are being developed as part of this contract:

(1) The Operational Program
(2) The Data Reduction Program
(3) The Test Generator Program

The development of these three CI is the responsibility of the contractor's software engineering organization. The internal build testing, the conduct of integration testing and demonstration of the prototype at site A is the responsibility of the contractor's test and control organization. The test and control organization is also responsible for demonstrations at sites B and C under this contract and possible subsequent upgrade testing of the software during later contracts.

The configuration of the operational program is managed at the unit level with all changes reviewed and approved as each unit comes under configuration management in the master library. The configuration of the data reduction program and test generator program is managed at the component level after being released for use with the operational program.

This SCMP specifically covers the configuration management support provided by the software configuration management department to the EDSS project office for

(1) The development of software used for different builds in test
(2) The prototype demonstration at site A
(3) The demonstrations at sites B and C.

1.3 Definitions and Mnemonics

1.3.1 Standard Definitions. Definitions used are found in ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology. Specifically, attention is called to definitions of

configuration item
configuration identification
configuration status accounting
master library
software library

1.3.2 Other Definitions

prototype system. The software developed for demonstrating the feasibility of the system concept.

1.3.3 Mnemonics. The following mnemonics are used within this document:

AXCESS	Vendor Software Company
CCB	Configuration Control Board
CI	Configuration Item
CM	Configuration Management
DRP	Data Reduction Program
EDSS	Experimental Development Software System
OP	Operational Program
SCA	Software Change Authorization
SCI	Software Configuration Item
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	System/Software Change Request
SDG	Software Development Group
SPR	Software Promotion Request
T & CG	Test and Control Group
TGP	Test Generation Program

1.4 References

[1] ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology.[13]

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

[3] Contractor Software Engineering Organization Labeling Standards for EDSS System.

[4] Contractor Test and Control Malfunction Reports.[14]

[5] EDSS Software Development Plan

2. Management

2.1 Organizations. All authority for managing the EDSS system is vested in the EDSS project office. The software engineering organization and the test and control organization provide personnel on loan to the EDSS project office for the duration of the project. The configuration management department provides qualified personnel to the EDSS project office to perform the necessary SCM coordination. Figure 1 illustrates the major organizations.

The working organization is divided into two main groups

(1) EDSS software development group (SDG)
(2) EDSS test and control group (T & CG)

Both groups report to the EDSS project manager. The SCM coordinator is provided by the CM department to the EDSS project office to help support both the software development group and the test and control group.

The EDSS project office has full responsibility for program management functions, including configuration management, until the demonstrations at all three sites are concluded. The SDG has responsibility for preparing and maintaining requirements specifications, designing the software, and performing the unit testing needed for all builds. The T & CG is responsible for integration tests, field installations, and all demonstrations. The SCM coordinator is responsible for processing all changes affecting the documentation (including test data and test procedures) and programs after their release to the T & CG.

The EDSS project manager is responsible for approving/denying all changes to the program, whether originating from the T & CG or from the customer. The project manager functions as the configuration control board (CCB).

2.2 SCM Responsibilities. The general responsibilities of the SCM coordinator are to process the information needed to control changes in the prototype software as it develops and to capture the as-built documentation, test data and reports, and code that represent each successful site demonstration. The emphasis is placed on supporting the project change activities by independently handling all of the required paperwork —making the CM process transparent to management.

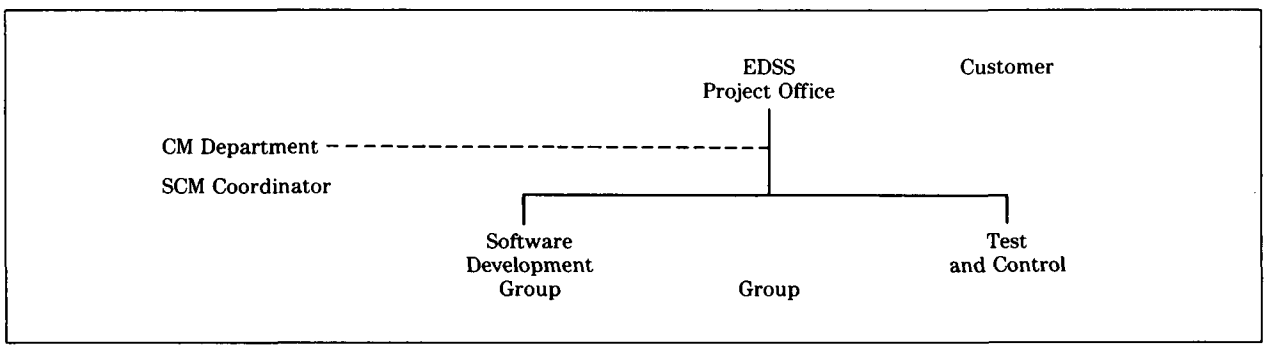
Specific organizational responsibilities of the SCM coordinator are as follows:

2.2.1 Identification. Naming conventions are established for

[13] IEEE publications are available in the company technical library.

[14] Organizational standards are available from the EDSS project office secretary.

Fig 1
Project Organization Chart



(1) *Unit Names.* These are designed so that unique identification of each item is possible. In addition, the unit naming conventions are structured so that it is possible to determine which SCI each unit belongs to by simply looking at the unit name.
(2) *File Names.* These are designed with the same mnemonic capability as the units.
(3) *Component Names.* These are given unique names so the source code can be matched to the supporting documentation.
(4) *Configuration Item Names.* These are defined in the same manner as in the contract statement of work.

2.2.2 Control. Control of all changes is maintained by
(1) Preparing and tracking approved system/software change requests (SCR), including all problem reports originatingfrom the customer, throughout implementation and testing
(2) Acting as software librarian, controlling the release of code to
  (a) The integration library for integration and testing by the T & CG at the contractor's development facility
  (b) The master library for installation and demonstrations at the site(s)

2.2.3 Status Accounting. The SCM coordinator provides the necessary status reports to the groups and project management. Typically, the reports cover
(1) SCR opened during period XXXX-XXXX[15]
(2) SCR closed for period XXXX
(3) Major SCR remaining open for three or more weeks
(4) SPR made during period XXXX
(5) SCR included in SPR, by date of promotion

2.2.4 Audits and Reviews. There is no pertinent information for this section.[16]

2.3 Interface Control. The EDSS system interfaces with the *AXCESS* software being developed by the *AXCESS Company.* The interface with this software is defined in an interface specification developed jointly by representatives from the software development activities of each company. The specification is approved by the responsible project managers of each company.

The EDSS interfaces with the hardware configurations found at customer sites are defined in a memorandum of agreement between the customer and EDSS project manager. Where agreement is not mutual, resolution is reached by contract negotiations. For changes to the EDSS system, the EDSS project manager initiates all change requests. For necessary changes to the *AXCESS* system, the *AXCESS* project manager initiates the change requests.

2.4 SCMP Implementation. The CM Department supports the EDSS project office with the services of a qualified SCM coordinator on the basis of 50% of one person's services per month.

One four-drawer file cabinet in the library is used for storage for the period of time specified in Section 6.

One workstation for execution of the data management system is used for the duration of the project.

Key events in the SCM planning phase are
(1) Establishing the integration library upon release of the first unit to T & CG for integration
(2) Establishing the EDSS master library upon release of the first software system configuration for demonstration at site A
(3) Impounding the master libraries from the three sites, along with the associated documentation and test data and reports, at the end of the final site demonstrations

2.5 Applicable Policies, Directives, and Procedures. The following standards and procedures apply for the duration of the contract:
(1) Labeling standards used for documentation, test data, and software media are in accordance with the standards in the software engineering organization's standards and procedures manual, modified as necessary in the EDSS software development plan.
(2) Version level designations are sequential numbers following a dash, appended to the documentation/media label.
(3) Problem report (SCR) processing is done according to the flow diagram in Attachment B.
(4) Procedures for operating the integration library and the EDSS master library are documented and distributed as a part of

[15]The period *XXXX* is left to the discretion of the program manager but is no less frequent than three-week intervals.

[16]No audits are performed as there is no contractual requirements. All reviews are informally conducted. Since there is no formal delivery, the software quality assurance activity is not involved in the configuration management of the software.

the EDSS software development plan prior to establishing the integration library

3. SCM Activities

3.1 Configuration Identification

3.1.1 EDSS Project Baselines. The requirements baseline (functional baseline) is established as the list of functional capabilities set forth in Addendum 2 of the statement of work in the contract.

The design baseline (allocated baseline) is established as the source code and associated design documentation, and all test procedures of the *as built* configuration items are successfully demonstrated to the customer at site A.

The prototype system baseline (product baseline) is established by the current design baseline of site A and versions of the configuration items for sites B and C at the end of the final demonstration.

Integration baselines are used to maintain successive builds during the development of the prototype demonstration at site A. A significant number of software builds at site A can be expected.

3.1.2 EDSS Project Labeling. The basis for labeling is by mnemonic labels assigned to each unit. In addition, each source unit shall have a prologue embedded in it. This prologue shall contain the following elements:

(1) Unit name
(2) Component name or identifier
(3) CI identifier
(4) Programmer name
(5) Brief description of the function of the module
(6) Change history
  (a) Date of each change
  (b) Reason for change (see SCR)
  (c) Change level (version being changed)

For example, the initial version of a unit is A-0, the second is A-1, etc. The change level is incremented each time the code is revised. The change level and the unit name are used to uniquely identify the version of the source code that incorporated a given problem correction, for example, ABC (3) for revision 3 of the unit ABC.

3.2 Configuration Control

3.2.1 Configuration Control Board. The EDSS project manager performs the functions of the change control board. The SCM coordinator supports the project manager by preparing SCR for the manager's review and processing the SCR subsequent to the manager's decision.

3.2.2 Processing SCR. The procedure for handling SCR is described in Attachment B.

3.2.3 CCB Interfaces. The EDSS project manager performs all of the coordination necessary with the customer in reviewing and in accepting, rejecting and negotiating changes. The manager also performs the liaison with the AXCESS vendor. Changes originating from the EDSS project are processed by the SCM coordinator. The two project managers provide coordination between projects and mutually resolve differences.

Changes to a system that result from these agreements are initiated by the responsible project manager.

3.3 Configuration Status Accounting. Status accounting is accomplished by tracking the changes to units through the use of the SCR form. This manually generated form (reference Attachment C) is updated (upon release) with the version number of each release.

Status of each CI is reported periodically to the project manager or at the manager's request. The status of the revisions to the units and components is reported weekly to the managers of the SDG and T & CG. When a software system is released to a site, the release and version are recorded and the units contained in the system are listed, along with their current change level.

3.4 Audits and Reviews. No audits are scheduled to be held for the EDSS system. Instead, the system is verified through the customer's functional testing. Parallel operation using the site's previous manual system and the new system is maintained until the users are confident that the system is producing accurate reports and displays. The SCM coordinator attends performance/functional reviews to record action items and change status.

Functional reviews are held periodically during the software development cycle. The principle document used is the *User Interface Guide*. This document contains the layouts of each of the displays and reports the users have available from the system. Each data element in each display or report is defined there, along with the method by which the element is derived (if any).

4. Tools, Techniques, and Methodologies

The primary technique for SCM is the manual processing of SCR, SCA, and SPR. The SCR form (see Attachment C) is used to record all customer requests for changes, their disposition, and eventual implementation. The same form is used to record enhancements or changes requested by the SDG. These forms later become the basis for updating the requirements specifications and for resolving questions concerning the origin of a change or the status of a requirement that arise during implementation, during integration, installation and checkout at site(s) and during the demonstrations.

A set of basic SCM tools is available for use. A data management system is used for recording and reporting status of the units, components, and CI.

The integration library uses a file system to check in and check out units for revision and test. The project master library uses the same system to impound master copies of the units and components.

Release of code to the integration library is made through software promotion requests (SPR) shown in Attachment A. The software is compiled and built into a protected integration package owned by the T & CG. Software successfully demonstrated at the sites is placed in the EDSS master library for future demonstrations and upgrades.

5. Supplier Control

There is no pertinent information for this section.

6. Records Collection and Retention

Copies of each status report are maintained as a historical record for the EDSS project until the project is terminated or the prototype demonstration system is replaced by the production system. These records are transferred to microfiche as they age over six months.

The prototype system baseline code, test data and reports, and documentation shall be maintained at the termination of the project for a period of two years or until replaced by a production system. The software media for retention of this baseline code is magnetic tape. The documentation for this is retained on microfiche.

Test procedures and test data resulting from the successful demonstrations shall be retained as a part of the data for use in defining the production system.

Attachment A

Software Promotion Requests

Table 1 defines the list of data elements included in the Transaction file for release of each unit.

Table 1
Data for Software Release

Element Name	Definition
CI number	Number assigned for CI identification
Sub-application	The name or number assigned to the unit or portion of the CI being released
Release request	Data release was requested
Action requested	The control action requested by the development activity (builds, move to test library)
Members	Names of modules, units to be included in the release
Change level	The change level or version number of the units being released
Justification	The number or the statement of justification concerning the reason for release
From library	The library location of the units before the release
Member type	The type of the unit being released (procedures, macros, test drivers)
Load module	The load module with which the units are linked
Time tag	The time tag for the version of the unit being promoted

Attachment B

IEEE Guide for Processing System Software Change Requests

System/Software Change Request (SCR) forms are used to document three types of situations:

(1) Requests for changes to the software by the customer (whether these requests result from tests, demonstrations, or from experience at the sites).

(2) Requests for changes by the designers or coders (generated within the company) that affect code already in use at the sites.

(3) Problems or errors in the code in test or at the sites that were clearly not requests for new or different functions—documented *bugs* in the released code

1.1 Processing steps are

(1) All SCR are logged in by the SCM coordinator and assigned a number on receipt. After logging, the SCR is forwarded to appropriate manager for action or resolution.

(2) When an SCR results in a software change (whether a correction or a new function), the software manager annotates the SCR form at the time of release of the new software to the sites and forwards the SCR to the SCM coordinator who then updates the master file.

1.2 SCM Coordinator. The SCM coordinator attends all customer/designer meetings and acts as the recorder of change requests. Signature approvals are obtained at that time.

When a release of software to sites is being prepared, the SCM coordinator meets with the project manager and review all outstanding SCR against the released software.

The SCR closed at that time are documented by the SCM coordinator.

Attachment C

System/Software Change Request Form

The following data elements are included on the SCR form.

Table 1
SCR Data Elements

Element	Values
CI	The name of the configuration item involved
Environment	The hardware site involved (may be more than one as project uses three different types of minicomputers)
Change type	Legal values: new function, error correction, design change
Date requested	DD/MM/YR
Narrative description	Description of the change desired in language as explicit as possible; description of the problem in the case of error reports
Disposition	Final disposition: fixed, accepted but delayed, rejected.
	If fixed, description of changes made are included here
Requester	Person making the request for the change
Requester site	Location of the person making the request
Release and version	The release and version number in which the problem existed
Implementation data	List of modules involved in the change on system/software change request form
Implementation release and version	Release and version number in which change appears
Implementation ship date	Date on which the change is shipped to the sites
Responsible manager signature	
Customer approval signature	(Used only for changes to software already released for field use)

Appendix C

Software Configuration Management Plan for a Software Maintenance Organization

Version 1.0

Approved

_____

Mgr SPLIT Facility

_____

Mgr SCM Dept

Date: ___/___/___

Synopsis

This example contains a discussion of a hypothetical programming facility that manages the support software systems used in the design, development, test and maintenance of the software systems for a large software engineering company. The company has approximately twenty-seven hundred employees of whom nine hundred are professional software engineers with degrees in computer science, computer systems or electrical engineering. The average experience of the professional engineers is five and one-half years. The software products they build and maintain are primarily real-time systems for many applications, some critical and some not. The company has an extensive investment in software engineering facilities. There are software engineering work stations for a third of the professional programming staff and terminals available for the support staff. The work stations are attached to a local area network that is integrated with a large number of mini-computers and two mainframes.

The programming facility, SPLIT, is staffed with one hundred and thirty-five people. Fifty are systems and maintenance programmers. There is a software configuration management department within the company that performs all of the configuration management activities for the facility and the software engineering groups. Special emphasis is placed on the management of the products in the SPLIT facility since the productivity and reputation of the company directly depends on the efficiency and reliability of the support software used by the engineering groups. A special software configuration management group is permanently assigned to the SPLIT facility with the responsibility for controlling the company's support software. The company management supports this focus — as long as the software engineering activities do not complain too loudly about the service they receive.

In this environment, the software configuration management group in the facility has a direct role in the control of the support software. This group processes all changes made to the support software by the system programmers, builds the run-time systems and performs all the other normal configuration management activities. The role of configuration management in maintenance makes this group a major part of the facility's management team.

Since the company has a considerable investment in the support software and data records, the disaster control practice requires that the support software in the production library have copies in the software archival repository. The company maintains the software repository in a protected shelter thirty-five miles from the main facility.

Contents

Appendix C

Software Configuration Management Plan for a Software Maintenance Organization

1. Introduction

This plan describes the standard operating procedures for managing the configuration of all the support software available to the users of the SPLIT facility. The SPLIT facility provides the supporting software used in the design, development and maintenance of software products produced by the company. All of the support software products available to the users of the SPLIT facility are maintained under configuration management to ensure that users have continual and reliable service from the software products in the run-time environment, and that errors in the support software and requests for enhancements are handled accurately, completely, and in a timely manner.

1.1 Purpose of the Plan. This operating plan specifies procedures whereby software configuration management supports the entire software change/enhancement process.

1.2 Scope. This plan defines the SCM activities necessary for maintaining all support software items being procured, tested, sustained and kept in the production environment in the facility. The list of the software configuration items will vary over time. The consolidated list of configuration items and their status is maintained by the SCM group within the SPLIT facility and published monthly in the SPLIT configuration summary.

1.3 Definitions and Mnemonics

1.3.1 Definitions. The terms used in this plan conform to the definitions found in ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology.

1.3.2 Mnemonics. The following mnemonics are used within this document:

CCB	Configuration Control Board
CCM	Configuration Change Management [system]
CI	Configuration Item
CM	Configuration Management
COMM	Communications Software
EWS	Engineering Work Stations
HW	Hardware
LAN	Local Area Network
SCA	Software Change Authorization
SCM	Software Configuration Management
SCMG	Software Configuration Management Group
SCMP	Software Configuration Management Plan
SSQAG	SPLIT Software Quality Assurance Group
SCR	System/Software Change Request
SQA	Software Quality Assurance
SQAG	Software Quality Assurance Group
STEG	SPLIT Test and Evaluation Group
SDT	Software Development Tools
TFR	Transfer File Request

1.4 References[17]

[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.

[3] GP:25, Software Configuration Management.

[4] GP:26, Software Change Request Processing.

[5] SF:39, Vendor License Identification and Accountability.

[6] SF:27, Inspection and Test of Support Software Products.

[7] SF:15, Test and Evaluation Group Activities.

[8] CMP:13, Identification and Labeling of Software.

[9] CMP:25.3, Unit Naming Conventions.

[10] CMP:25.4, Version Level Designation.

[11] CMP:37, Computer Program Media Identification and Marking.

[12] CMP:12, Software Auditing.

[13] SP:17, Support Software Status Reporting.

[17] Referenced documents are available for use in the SPLIT software reference library.

[14] SP:12, Operation of SPLIT Configuration Control Board.

[15] SP:5, User Documentation Maintenance.

[16] SP:7, SPLIT Production Library Maintenance.

[17] SP:95, Work Station Request and Allocation.

[18] SCMG-WP:19, Data Retention—SCR/SCA.

[19] SCMG-WP:1, Software Release Procedures.

2. Management

2.1 Organization. The vice-president managing the SPLIT facility reports to the company president along with the vice-president in charge of the product effectiveness group and the vice-president in charge of the operations division (engineering). The configuration management (CM) department is part of the product effectiveness group. The software configuration management group (SCMG) is administratively a part of the CM department and their activities are responsive to the policies set by the CM department; but, functionally, they report to the manager of the SPLIT facility.

The organizational structure of the SPLIT facility is shown in Fig 1.

2.1.1 Operations Group. The operations group maintains the processing and communications systems, installs and reconfigures hardware installations, and performs the day-to-day operations of the processing environments.

2.1.2 Systems Software Programmers. The systems software programmers perform the maintenance on the support software developed in-house (generally by the engineering division) and subcontracted software acquired by the facility. Third party software acquired from vendors is not maintained by the SPLIT facility.

2.1.3 Test and Evaluation Group. The test and evaluation group performs the acceptance tests for vendor and subcontracted software and also all new releases for in-house support software maintained by the systems software programmers.

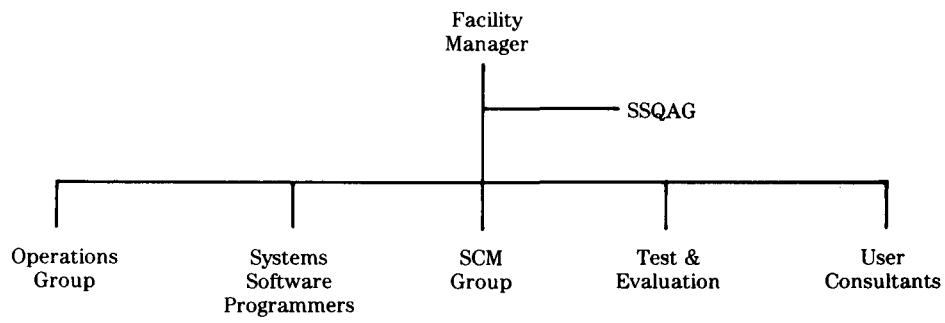
2.1.4 User Consultants. The user consultants provide training to that portion of the company that does not include Section 2.1.3 in the use of the support software systems, and consulting services to the software engineers as needed. They are the primary source of change requests for support software.

2.1.5 SPLIT Software Quality Assurance Group. The SSQAG is functionally a part of the product effectiveness group. They perform evaluations of new software as an incoming QA function, and periodic audits of the operations of the facility.

2.1.6 Multiple Configuration Control Boards. There are multiple configuration control boards (CCB) within the facility. The senior CCB, called the SPLIT CCB, has overall responsibility for managing the hardware and software configurations in the facility. This responsibility includes

(1) Allocating SPLIT resources for use on company projects
(2) Setting overall schedules for support software updates and new version releases
(3) Allocating resources to update configurations of mainframe processors, the minicomputer nodes, and the LAN/Hi-Speed data bus configurations.

Fig 1
SPLIT Facility Organization



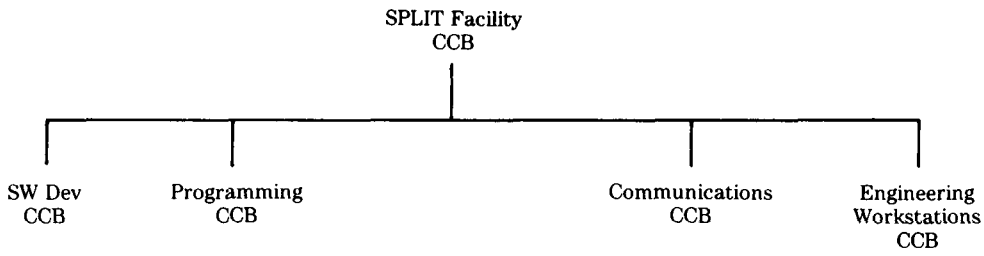


Fig 2
Structure of CCB

(4) Providing resources to subordinate CCB that manage software product lines

The manager of the SPLIT facility chairs the SPLIT CCB. The head of the SCMG is the alternate chairman and attends all meetings of the SPLIT CCB.

The in-house software is grouped by function into three separate CCB

(a) Software development (SWDEV) tools
(b) Programming environments (PROG)
(c) Communications (COMM) software

These subordinate CCB have configuration management responsibility for support software developed in-house, and managing the changes approved for software acquired from outside sources. Individual product line CCB are assigned to software products developed by the company, but their operation is independent of the SPLIT facility CCB. When these company software products are used in the SPLIT facility, they are controlled in the same way any product purchased from an outside vendor is controlled. The software used in the engineering work stations has a separate work station CCB for tracking the volatile hardware and software configuration.

Each SPLIT facility CCB is responsible for allocating resources needed for maintaining their assigned software products. Where changes affect interfaces with other hardware or software within the facility, or both, the issue must be brought before the SPLIT facility CCB. The head of the SCMG cochairs the SPLIT facility CCB and work station CCB with their respective managers.

Each project making use of a software product has representation on the CCB controlling that product.

2.2 SCM Responsibilities. The primary SCMG responsibilities involve supporting the change process as it affects existing software product baselines; maintaining an accounting of the status of *all* the software configuration items in the facility; and auditing physical configurations (CI) received from subcontractors, vendors of commercial software used in the facility, and support software from the company engineering division.

2.2.1 Identification. The SCMG is responsible for maintaining the identification (numbering, labeling, and integrity of documentation) for all the support software in the facility. Responsibility also extends to identifying the configuration items that are acquired from commercial vendors.

2.2.2 Configuration Control. The SCMG is responsible for supporting the change process for all of the support software used in the SPLIT facility.

2.2.3 Configuration Status Accounting. The SCMG maintains the data base used to prepare reports on the status of all support software products and hardware configurations used in the facility.

2.2.4 Audits and Reviews. Audits are performed by two groups

(1) The SCMG performs physical configuration audits of all support software acquired by the facility. Periodic inventory audits of the support software are also performed as directed by the SPLIT facility manager
(2) The SCMG supports SSQAG in performing functional configuration audits of incoming subcontracted and vendor-provided support software. The SCMG also provides SSQAG with summary data on probable causes of failure

The SCMG works directly with the STEG in evaluating software changes being released to the production library.

2.3 Interface Control. One of the most critical activities is controlling the interfaces between the different software systems in the facility and between the software and changing hardware configurations.

The SCMG supports the interfaces between the multiple CCB by recording action items affecting each interface and following up on them to see that they are accomplished in a timely manner.

The SCMG maintains configuration control of the specifications and standards controlling the interfaces between the software elements of the workstations. The workstation configuration must include both hardware and support software for each installation. This includes accounting for leased and licensed software used on personal computers and in workstations.

The SCMG maintains the operating system configuration used in the SPLIT facility as a means for enforcing control of the interfaces with the applications programs.

2.4 SCMP Implementation. The staff of the SCMG is composed of one group head, who acts as coordinator, and one qualified SCM administrator for each separate SPLIT facility CCB (one per CCB). One additional person has the function of tracking the EWS configuration(s).

Computer resources and work space are provided by the SPLIT facility manager for the SCMG.

Milestones for SCMG activity are set by the manager of the SPLIT facility and reflect the ongoing continuous support activities required for managing the various support software configurations.

2.5 Applicable Policies, Directives, and Procedures

2.5.1 Policies

(1) Company Policy
  (a) GP:25, Software Configuration Management
  (b) GP:26, Software Change Request Processing
(2) SPLIT Policies
  (a) SF:39, Vendor License Identification and Accountability
  (b) SF:27, Inspection and Test of Support Software Products
  (c) SF:15, Test and Evaluation Group Operations

2.5.2 Directives

(1) Company Bulletin, GB:87, *Use of Licensed Software*
(2) Company Directive, CD:34, *Copyright Protection*
(3) Company Bulletin(s), GB:(various), *CCB Membership*

2.5.3 Procedures

(1) Company Procedures
  (a) CMP:13, Identification and Labeling of Software
  (b) CMP:25.3, Unit Naming Conventions
  (c) CMP:25.4, Version Level Designations
  (d) CMP:37, Computer Program Media Identification and Marking
  (e) CMP:12, Software Auditing
(2) SPLIT Procedures
  (a) SP:17, Support Software Status Reporting
  (b) SP:12, Operation of SPLIT Configuration Control Board
  (c) SP:5, User Documentation Maintenance
  (d) SP:7, SPLIT Production Library Maintenance
  (e) SP:95, Work Station Request and Allocation
(3) SCMG Procedures
  (a) SCMG-WP:19, Data Retention—SCR/SCA
  (b) SCMG-WP:1, Software Release Procedure

3. SCM Activities

3.1 Configuration Identification. Each support software product in the facility is identified by configuration item title, specifications, user documentation, and media labels in accordance with established company procedures.

Since the software being managed has already had a product baseline established, the identification schema is already set. The SCMG uses the identification and labeling standards in the product baseline. In-house software identification follows company procedures CMP-13; 25.3; 25.4; and 37. Third-party software is labeled with company-defined labels for record-keeping purposes.

The elements of software (programs, documentation, test data, etc) in the production library (the library of software released for running on hardware in the facility) is organized as in Table 1.

Table 1
Hierarchy of Elements

Generic Term	Alternate Terms
Configuration item	Package, product
Component	Segment, program
Unit	Module, routine

The level of control applied by the SCMG will generally be to the component level. Components are considered to be the *controlled item* in managing the operation of the SPLIT facility. A given programming library used by systems programmers may have a system for managing configurations of software units previously used in the development and maintenance of other programs. Sometimes the units in these libraries are referred to as packages, following the concepts of *reusable software* being advocated.

3.1.1 Baseline Identification. Support software product baselines are established during incoming inspections of the product at the facility. New releases to a product baseline are labeled in accordance with 2.5.3(1)(c). New releases include changes or updates as necessary to the product package—specifications, user documentation, design documentation (listings), test procedures, and associated test and inspection reports. The procedure 2.5.3(1)(a) is followed for each new release of a support software product.

A new release of a support software product is made in accordance with 2.5.3(3)(b).

The scheduling of a new release is determined by the SPLIT CCB.

3.1.2 Product Baseline Cataloging. Labeling of product CI is in accordance with 2.5.3(1)(a). The SCMG reviews each request to be released for conformance to company procedures. The SCMG then checks the release package against the transfer file and the CCB authorization for completeness and STEG/SSQAG approvals.

3.2 Inspection and Receiving. New products entering into the facility for use are inspected for conformance to 2.5.3(1)(a) by the SCMG. Vendor software parts (configuration items) are given company CI part numbers in the 7000 series for maintaining separate accountability within the status accounting system.

3.3 Configuration Control

3.3.1 Levels of Authority for Approvals. All software is tested by the STEG prior to its promotion into the integration library or the production library. Both STEG approval and SCMG approval is required before the software is promoted to the integration library or production library.

The promotion of changes into the integration library is authorized by the SPLIT facility and work station CCB and approved by the SCMG after design checks by the STEG.

The release of changes to the production library is authorized by the SPLIT CCB. Prior to entering changes into the production library, each change is tested and verified as correct by the STEG, checked for conformance to packaging standards by SSQAG, and administratively approved by the head of the SCMG before being placed into the library.

3.3.2 Change Proposal Processing

3.3.2.1 SCR Processing. Software change requests are prepared using the form C-1049, software/system change request, or use of the SCR ENT command in the interactive configuration change management (CCM) system. Manually prepared forms (C-1049) are entered into the CCM system by the SCMG librarian. The same form used to initiate a problem report is used for requesting an enhancement to the system. All changes are concurrently routed to the SCMG files in the CCM system for administrative checks and to the appropriate product line manager for verification. Each SCR is reviewed by technical personnel and their evaluation is forwarded to the appropriate SPLIT CCB for action.

The SPLIT CCB can approve, reject, or table (with an action date) a request pending further information.

Action in response to a SCR is scheduled by the CCB in response to the severity of the problem reported or the need for enhancement. Problem reports are given priority over change requests not associated with an operating problem. Problem reports (as indicated on the SCR form) are processed on an expedited basis.

Problem reports that are determined to be valid errors in the performance of the system and given priority for solution with temporary fixes are incorporated into the subject system—along with publication of a bulletin notifying all users of the

Table 2
Problem Criteria

Category	Symptom
"C"	A software item cannot be executed by a user
"M"	Users have problems with a program but can work around with temporary fix
"S"	Minor irritation but users can still accomplish work

change in the system. Permanent modifications to correct the error are incorporated with the next upgrade released to all users.

Requests for system enhancements that are valid and within the scope and resources allocated to the software product are scheduled for incorporation in the next scheduled upgrade to that product.

Approvals are incorporated in the maintenance schedule and a release date tentatively identified for a scheduled upgrade or correction to the affected support software product(s). Status of these SCR is indicated as *approved.*

The SCR may be returned to the user when additional clarification is needed or when the results of the design review may necessitate additional design analysis or even modification to the change request. The SCR is held with the status *pending* until a course of action has been determined.

Testing for promotion to the integration library or release to the production library may result in additional design changes or recoding. In that event, the status of the SCR reflects *approved* and the status of the SCA reflects *in-work.* The status of the SCR/SCA action is changed to *implemented* only if the change has been completed, verified, and released into the production library.

3.3.2.2 SCA Processing. Approved SCR are forwarded by the CCM system to the appropriate programming activity for implementation. Similar changes that are grouped together for an upgrade are worked on at the same time. Emergency changes (needed to keep the system in operation) are expedited through the system. The programming activity extracts necessary files for work from the production library and makes the changes. When the supervisor is ready to integrate the file, the SCA and the code are completed and passed to the CCM_HLD/INT area for administrative checks by SCMG before being released to the integration library for integration and test.

STEG performs the integration and testing, requesting modifications from the programming activity as appropriate. When it has been demonstrated that the change package is correct and introduces no additional errors into the system, the SPLIT CCB is informed of the pending update whereby STEG initiates a transaction file request (TFR). Upon approval by the SPLIT CCB, the SCMG enters the change into the production library. The status of the SCR/SCA is then changed to *closed.*

The SCMG performs the systems generation of the run-time programs used in the facility, and loads, after verification by the STEG, into the necessary hardware configurations.

Failure of the users to accept the changes in the support software system may result in it being returned to a previous step or cancellation of the task.

3.3.2.3 Changes to EWS. The processing of changes to work station support software is the same as the above procedure except that the run-time software generation and allocation to HW configuration is controlled by the EWS CCB network manager.

3.3.2.4 Changes to Supplier Software. Change processing for subcontracted software is performed in the same manner described above when the source code is in-house and maintenance is being performed by the SPLIT systems software programmers. When the software product is under subcontractor warranty, the SCR is passed to the subcontractor and the new version is accepted into the production library in the usual manner. In the event where the subcontractor has a maintenance contract for the product, the SCR is passed on to them for processing.

3.3.2.5 Licensed Software. Licensed software is given a company label with a unique identifier to indicate limited use. Periodic audits are conducted by the SCMG to determine adherence to the license limitations by users.

3.3.2.6 Purchased Commercial Software. Purchased commercial software is relabeled with company identifying numbers and released for use and configuration management in the same manner as in-house developed software.

3.3.3 CCB Roles. The CCB evaluation takes into consideration, among other things, the staff resources available versus the estimated workload of the request; the estimated additional computing resources that are required for the design, test, debug, and operation of the modified system, and the time and cost of updating the documentation.

An essential function of each CCB is to coordinate the flow of information between the users of the software product and the maintenance organization supporting the product. This function is executed when the CCB representatives of the project use the products and monitor the evaluation of the significance of problem reports and requests for enhancements. The result of the CCB review is the assignment of a priority to each request.

3.3.4 Control of Interfaces. There is no pertinent information for this subsection.

3.4 Configuration Status Accounting. The SCMG supports the following reports:

(1) *SPLIT Software Configuration Report.* An accounting of the software and hardware configurations of all the systems within the SPLIT facility. This report is kept current at all times. Weekly reports are made to the SPLIT facility manager, including changes just completed and changes scheduled for the next week.

(2) *SPLIT Performance Summary.* A monthly summary of the up-time of all systems and an analysis of all problems causing unscheduled down-time.

(3) *SCR/SCA Summary.* For each configuration item, a summary of the current status of SCR/SCA activity is given on a weekly basis to the SPLIT facility manager. The SCR summary includes problem type and severity, priority given by the CCB, activity or programmer assigned, and target release date for either the fix or new release.

(4) *EWS Configuration Status.* This configuration status is maintained in a data base for general access. Status and configuration summaries are presented to the SPLIT facility manager on a weekly basis.

3.5 Audits and Reviews. The SCMG performs a physical configuration audit on all incoming third-party software.

The SCMG performs functional and physical configuration audits on each new release of software in the system.

The SCMG performs periodic audits of the software and hardware configurations in the facility to ascertain that no unauthorized changes have been made. Particular attention is paid to licensed software.

4. Tools, Techniques, and Methodologies

4.1 Use of the CCM System. The CCM system is used to manage and track all changes to the software in the SPLIT facility. The system provides for initiating changes, review and approval by management, assigning and monitoring work status, and the testing and releasing of all changes. Status reporting is provided as an output from the CCM data base. This configuration management tool is one of the set of software tools used in the SPLIT facility by all of the operating activities.

4.2 Inspections. Releases to the production library are inspected to confirm inclusion of scheduled SCR/SCA.

4.3 Library Management. The SCMG makes disciplined use of programming libraries to manage the changes to support software configuration items. The SCMG and the STEG cooperate in promoting software modifications from the development library into the integration library and from there releasing them to the production library.

4.3.1 Development Library. The development library is used by the systems software programmers as they develop their code. The units and components are controlled by the individual programmers. Criteria for allowing promotions into the integration library includes the successful completion of unit testing and approval by the group's supervisor.

4.3.2 Integration Library. The integration library is used by the SCMG to capture and build the code that is designated for promotion to the STEG for integration and test. This library contains the source code and executable load modules created as a result of a system build. The source code is placed in a special controlled library in preparation for a build. Then the code is recompiled and link edited before it is placed in the integration library. Criteria for releasing to the production library includes

(1) Submission of a software release request by the SPLIT CCB

(2) Completion of status accounting audits and resolution of issues by SCMG

(3) Acknowledgment of regression and integration test completion by the STEG and SQA

All test data and routines used to verify software released for use are also maintained under configuration control in the integration library.

4.3.3 Production Library. The production library contains the master copies of all the support software configuration items used in the SPLIT facility. Copies are made from the masters by the SCMG for use on other systems. The production library acts as backup for the run-time configurations used on the systems. Only current master copies of support software configuration items are maintained in the production library.

4.3.4 Software Repository. Current copies of all support software configuration items from the

production library are maintained in the software repository. Historical copies of support software released for use outside the facility are maintained in the repository for a period of ten years after release.

5. Supplier Control

Since the SCMG does not have responsibility for supporting the development of subcontracted software, the SCMG has no interface with the support software developed in this way.

The SCMG does participate with the STEG in the receiving inspection of commercial software and subcontracted software to ascertain that

(1) All physical items are available as required by contract
(2) The proper labels are on the media to be placed in the integration library, and subsequently, in the production library

The SCMG is responsible for the physical configuration audit of subcontracted and vendor-supplied software. The SQA activity performs the functional configuration audit.

6. Records Collection and Retention

Records of SCR/SCA processing are retained for a period of five years to support fiscal standards of records. Status reports of the SPLIT facility configurations are also maintained for a period of five years.

Records defining the product baselines of all support software products released for use outside the facility (in conjunction with engineering division sales) are maintained for a period of twenty years to protect product warranties. The product baselines of all other support software products developed in-house but not released for use outside the facility are maintained for a period of ten years.

Records of licensed vendor software integrated, or otherwise used, with internal configurations are maintained for a period of five years after their removal from the system.

Biweekly backups of the systems are archived for a period of six months to protect the data files of the ongoing engineering division development activities. Backups of the systems processing company financial records are archived for a period of seven years, as required by law.

Attachment A

System/Software Change Request
(SPLIT Form C-1049)

Table 1
Definitions of Elements in SCR

Element	Values
Originator	Name of the person making the request
Product	Originator's subject support software product
Date	Date of change request (option: date of anomaly detection for the SCR
SCR number	Sequential number assigned for the product in question
SCR title	A concise descriptive title of the request
SCR type	One of the following types: AR—Anomaly Report SCN—Specification Change Notice ECR—Engineering Change Request ER—Enhancement Request IR—Impound Request
Program	Identification of the support software product for which the change is requested
System version	Version identifier of the system for which the change is requested
Description of change	Originator's description of the need for a change
Disposition	CCB indicates one of the following dispositions: Approved—Date approved and assigned for implementation _____ Deferred—Date deferred to _____ Rejected—Date rejected _____
User class	Indicates organization/activity using the software
Date needed	Indicates date the change is needed in the production system

For those SCR referencing anomalies detected in a product baseline, the CCB must verify that the problem exists and the following data should be added:

Optional Data for Anomaly Reports	
Item	Data
1	System configuration on which the anomaly was detected.
2	Performance effect—The effect the anomaly has on the performance of the system [c] critical; [m] major; or [s] small

Attachment B

Software Change Authorization

Table 1 defines the list of data elements included in the SCA file for releasing each unit. The SPLIT facility CCB may add to the list of elements. Deletions are made only with explicit approval of the SPLIT CCB.

Table 1
Definitions of Elements in SCA

Element Name	Definition
CI number	Number assigned for CI identification
Date	(1) Date change was released to the integration library
	(2) Date change was released to the production library
SCR number	The SCR number of the request/authority for making the change
Subapplication	The name or number assigned to the unit or portion of the CI being released
Release request	Date release was requested
Action requested	The control action requested by the development activity (builds, move to integration library, etc)
Programmer(s)	The names of the programmer(s) making the changes
Members	Names of modules, units affected by the change in the release
Change level	The change level or version number of the units being released
Justification	The number or the statement of justification concerning the reason for release
From library	The library location of the units before the release
Member type	The type of the unit being released (procedures, macros, test drivers)
Load module	The load module with which the unit will be linked
Verified by	Name of the person approving the verification
Verified system name	Identification of system used for testing change
Time tag	The time tag for the version of the unit being released

Appendix D

Software Configuration Management Plan for a Product Line System

Version 1.0

Approved

_____

Director, Engineering

_____

PLAS Program Manager

Date: __/__/__

Synopsis

This example Plan contains a discussion of a hypothetical project in a microelectronics company that makes microprocessors and microprocessor-based systems that are later embedded within other hi-tech electronic systems. The company has approximately nineteen hundred employees, of which one hundred and thirty-four are in the engineering division and the remainder are in the production division, marketing, and administration group. There is an extensive investment in hardware CAD/CAM to make the operation productive and a lesser investment in computer-aided engineering (CAE). Office automation is used to minimize the costly handling of paper; therefore, most of the communication within the company uses electronic media. Customers buy hardware or systems — receiving software products only as part of a system.

There is no independent software development activity. Software technology is considered a basic skill that electronic engineers and system designers use in their day-to-day work. The engineers design software for execution within their system's RAM or ROM with the same ease as they use the silicon compilers to design chips. There are two focal points where the different engineering design technologies interact with the configuration management discipline. The first focal point is in the system's computer aided engineering system where the engineering libraries (where functional logic and piece/part information is maintained) or data bases and VLSI design systems are maintained. The second focal point is in the production computer aided manufacturing system where the programmed logic is transformed from compiled into deliverable products. The two focal points are separate as the mode of implementation demands different interfaces — the production system interfaces directly with the hardware CAD/CAM systems in production; the engineering system with the software/firmware development stations and prototype-testing stations. The configuration management software to support management of these data bases is largely embedded within the program management system, which schedules work and manages the changes to baselines.

In this environment, the software configuration management disciplines are just another one of the tools used by engineering and production management for performing their daily tasks. The software configuration management plan focuses primarily on establishing unique project data base structures in the engineering systems, routing the change management materials to named organizational positions for approvals, and defining data-base baselines. Software configuration management is a service provided by the engineering, production, and management systems to help management more effectively perform their tasks.

Contents

Appendix D

Software Configuration Management Plan for a Product Line System

1. Introduction

This guide describes the plan for managing the configurations of stored program logic used in manufacturing the product line analysis system (PLAS) module. This module performs the computational, communications, and device-controller functions of a larger system — *The Quick Stretch*, which performs stress analysis for mechanical structures. This system is sold as a proprietary company product to customers and is maintained by field representatives of the company. The company intends that the PLAS module have functional flexibility through its use of computer programs to make the module adaptable to other company proprietary systems and possibly for sale to other systems manufacturers.

1.1 Purpose. This plan identifies the procedures for managing the configurations of the PLAS computer programs during their development and for maintenance of the programs throughout the time period the company sells and has warranty responsibility for the products that incorporate the PLAS as an embedded system.

1.2 Scope. This plan is applicable to the development and maintenance of all the computer programs embedded in ROM, loaded into EPROMS, or loaded into RAM for use in the PLAS module. Configuration management of the hardware associated with the PLAS module is covered in a PLAS hardware configuration management plan — PLAS-CMP. These computer programs, packaged in different media, are collectively managed under the single configuration item *PLAS software configuration item* regardless of their function. The computer programs packaged for ROM or EPROM are managed as hardware components, identified under their prime hardware configuration item identification. The support software used in production and test of the PLAS module components (both hardware and computer programs) is also controlled by this plan.

1.3 Definitions and Mnemonics

1.3.1 Definitions. The terms used in this plan conform to the definitions found in ANSI/IEEE Std 729-1983, *IEEE Standard Glossary of Software Engineering Terminology.*

hard logic. Programmed logic that is embedded as circuit logic in a chip. The logic is developed using the general software engineering tools and disciplines. Packaging of the logic uses silicon compilers for generating the geometry of the chip.

P-CAMS. The product computer aided manufacturing system (P-CAMS) environment that contains

(1) The engineering data bases of hard logic and stored-programmed logic defining the products in the production environment (the controlled libraries)
(2) The support software used in converting the controlled engineering data bases into instructions and data for
  (a) Production of chips, software and firmware
  (b) Test programs and data for verifying that the produced entities have been correctly implemented

User documentation is also produced using P-CAMS. Configuration management disciplines relating to product serialization, change labeling and tracking, and verification tests are a part of this environment.

project-management system (PMS). The PMS provides the capability for management to

(1) Define an identification schema for projects at start-up time and to make changes to the different schemes
(2) Authorize and control the release of project drawings and engineering data bases from the dynamic libraries in systems computer-aided engineering system (SCAES) to the controlled project libraries in P-CAMS

(3) Schedule the production and release of product changes, and coordinate the production schedules within the production division

In general, this system supports the configuration management change control board (CCB) and production scheduling activities.

stored program logic. Computer program instructions and data that are executed out of RAM, ROM, and EPROM in the PLAS module. The instructions and data are developed using general software engineering tools and disciplines. Packaging of the instructions and data uses technology appropriate for the media.

systems computer aided engineering system (SCAES). The SCAES environment is composed of

(1) A variety of engineering support software including different simulators, prototyping tools, modeling programs, engineering design aids, documentation tools, test generators, test simulators, utilities and compilers
(2) Engineering libraries (the **dynamic** libraries) that contain general algorithms that have widespread utility, reusable stored-programmed logic, reusable hard-logic functions, and access to selected product designs
(3) Design data bases representing the dynamic working libraries for product developments that are currently in progress (such as the PLAS module development)

The commands that are used in SCM disciplines for supporting identification of entities relating to a specific project and for tracking current versions of those entities are an integral part of SCAES.

1.3.2 Mnemonics. The following mnemonics are used within this document:

APM	Associate Program Manager
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacturing
CAE	Computer-Aided Engineering
CAES	Computer-Aided Engineering Systems
CCB	Configuration Control Board
CI	Configuration Item
CM	Configuration Management
CMP	Configuration Management Plan
CMS	Change Management System
CSCI	Computer Software Configuration Item
DP&S	Data Processing and Support
EPROM	Erasable Programmable Read Only Memory
EWS	Engineering Work Stations
EWSW	Engineering Work Stations Environment
LSI	Large Scale Integration
MSI	Medium Scale Integration
P-CAM	Product Computer-Aided Manufacturing System
PLAS	Product Line Analysis System
PMS	Project Management System
QC	Quality Control
ROM	Read Only Memory
SCA	System Change Authorization
SCR	System Change Request
SCAES	Systems Computer-Aided Engineering System
TD	Technical Director
VDD	Version Description Document
VLSI	Very Large Scale Integration

1.4 References

[1] PLAS Functional Requirements.[18]

[2] Engineering Work Station and Environment User's Manual.

[3] Programming Standards Manual.

[4] Product Line Identification Numbering Standard.

[5] Software Quality Assurance Policy.

[6] Production Test Standards.

2. Management

2.1 Organization. The PLAS program manager of the product line has financial and administrative responsibility for all PLAS module engineering and production. He is part of the administration and reports directly to the general manager of the company. The company uses a matrix organization for managing projects.

The PLAS program manager has final responsibility for the business success of the program. The project staff consists of the financial staff, the technical director (TD), an associate program manager (APM), and a quality representative from the quality control (QC) department. The PLAS APM is functionally a part of the production division and attends all PLAS project meetings.

[18] All referenced documentation is available from the SCAES library.

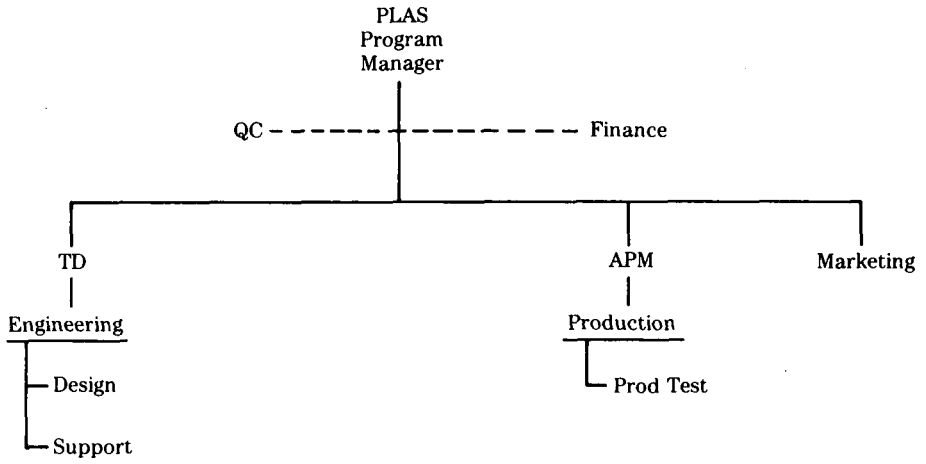


Fig 1
PLAS Organization Chart

The major elements in the administration, engineering division, and production division that support the PLAS product line include

(1) Marketing (administration) provides the sales and marketing support to
  (a) Perform market analyses and prepare functional requirements for the Quick Stretch System that indirectly determine the functional requirements for engineering the PLAS module
  (b) Maintain customer liaison for product maintenance and improvement and
  (c) Sell the Quick Stretch Systems

(2) The PLAS engineering design group (engineering division) is an ad hoc organization under the direction of the PLAS technical director, which provides engineering expertise to
  (a) Manage the overall system design activity
  (b) Develop the hard logic, the stored program logic, and drawings for PLAS assemblies
  (c) Review all proposed changes for feasibility, cost, and design integrity
  (d) Perform all necessary engineering design and logic changes

(3) The PLAS engineering support group (engineering division) provides the technicians and technical resources to
  (a) Maintain the SCAES_PLAS engineering data base
  (b) Perform product engineering based on design prototypes to be released for production
  (c) Support TD and PLAS program manager in verifying design changes prior to release to P-CAMS.

(4) The PLAS production group (production division) activity, under direction of the PLAS APM, provides the capability to
  (a) Manufacture hardware in accordance with PLAS drawings released for production
  (b) Compile, verify, and package programmed logic released as software for PLAS RAM
  (c) Compile, verify, and burn-in programmed logic released as firmware for PLAS EPROM
  (d) Compile, verify, and coordinate mask production or programmed logic released as firmware for PLAS ROM
  (e) Compile, verify, and coordinate production of hard logic released as VLSI chips
  (f) Test complete assemblies of PLAS modules
  (g) Maintain inventories
  (h) Ship PLAS modules to customers as directed by the PLAS program manager

The functions that are generally performed by a separate SCM activity and not supported by SCAES and PMS are shared between the quality control representative and the APM. This is possible because most of the detailed SCM processing activities and library interface management are accomplished by the PMS.

The PLAS technical director is the chairperson of the configuration control board (CCB). The PLAS identification scheme implemented in the SCAES control system is approved by the chairperson of the CCB. The responsibility for reviewing and approving all changes to established baselines and scheduling releases belongs to the CCB chairperson. Release of PLAS engineering data base(s) to P-CAMS and all changes to the P-CAMS data base for PLAS is authorized by the CCB chairperson.

2.2 SCM Responsibilities

(1) The PLAS program manager provides general direction to the TD for establishing the identification scheme, to the APM for production scheduling, and authorizes the establishment of baselines. The PLAS program manager also provides general direction to the TD for CCB actions and issues requests for QC to audit and review the integrity of the SCAES_PLAS engineering data base and the P-CAMS_PLAS production data base.

(2) The PLAS TD establishes the contract identification schema used by the PLAS project engineers and performs (or delegates to engineering support group) the duties of updating the P-CAMS_PLAS production data base when authorized by CCB actions. All changes to the P-CAMS_PLAS production data base are approved by the TD.

(3) The PLAS associate program manager (or a delegated assistant, such as a librarian) has overall responsibility for maintaining the P-CAM PLAS data base, PLAS unit and module tests, and production schedules.

(4) The production test group is responsible for testing the hardware assemblies, including the units containing the packages of programmed logic (ROM and EPROM), and verifying that the correct version of the logic is embedded in the device. The group also verifies that the diskettes containing the dynamically loadable software for the PLAS module is the correct version for shipment. Final assembly tests of these units along with VLSI chips are also conducted by this group.

(5) The PLAS quality control representative is responsible for reviewing the production test group's verification activities, and auditing the integrity and use of SCAES_PLAS engineering data base and P-CAMS_PLAS production data base. The QC representative verifies the physical configuration of the PLAS module, its associated user documentation, and its functional capabilities (review of module acceptance testing) as a part of the quality review prior to shipment.

(6) The engineering support group provides special extractions from the SCAES, P-CAMS and PMS systems data bases showing status of the various baselines when information other than that provided by general project status commands is required.

(7) The marketing organization provides the functional requirements for the system and is the major source of high-level system changes and improvements. In effect, this organization defines the functional baseline. Customers purchasing a PLAS module for their own use or for the PLAS module integration in the Quick Stretch System have no direct interface or review authority over PLAS baseline activities or product capabilities.

2.3 Interface Control. The data bases for the PLAS module are maintained in two different library systems: the SCAES_PLAS engineering data base and the P-CAMS_PLAS production data base. The interface between these two data bases is controlled by PLAS CCB authorizations.

The SCAES_PLAS engineering data base is made up of several parts representing

(1) Top-level drawing of the PLAS module

(2) Detail design representations of the programmed logic as it is to be packaged for implementation in ROM, EPROM, and RAM based software

(3) Detail designs for implementation in chips (LSI and VLSI designs)

(4) Electrical engineering drawings for cards and assemblies and

(5) Mechanical drawings for the module assemblies

The interfaces between these subdata bases are managed as developmental baselines during the engineering development phase of a PLAS module.

The interface with the Quick Stretch System, or with customer defined systems using the PLAS module, is defined by the top-drawing design data base. Changes in this interface are made only with the authority of the TD. In case of conflict, the PLAS program manager negotiates the changes with the appropriate system representative.

Interfaces with the P-CAMS_PLAS production data base and the computer-aided manufacturing software are managed by the APM in production division, as long as the changes do not affect the CAES_PLAS engineering data-base interface.

2.4 SCMP Implementation

2.4.1 PLAS Configuration Baselines. The functional baseline is established when the system level description for the PLAS module is approved by the general manager for prototype development. Marketing surveys and analyses of potential customer applications provide a description of the desired functional capabilities of the proposed system. The functional baseline is documented with the marketing analysis report, supplemented by a preliminary top-level drawing of a proposed system. This baseline is considered obsolete after acceptance of the preproduction baseline.

The allocated baseline is established upon approval of the top-level drawing and preliminary detailed designs, verified by simulation runs, by the PLAS program manager. This baseline is obsoleted after acceptance of the preproduction baseline.

The developmental baselines are established by the TD at his/her discretion as needed for coordinating the changing allocated baselines during development. The developmental baselines represent incremental software builds needed to develop the prototype system and to verify revisions to the production baseline or different models of the production baseline for various customer applications.

The preproduction baseline is established with the successful demonstration of a prototype system and an absence of any priority 1 (emergency) error reports or changes outstanding. The PLAS program manager authorizes development of the preproduction baseline when given the go-ahead by Marketing management.

The production baseline is established with the concurrence of the PLAS APM and PLAS TD that the design is functionally adequate and that the production facilities of the production division can produce the design in an economical way. The production baseline is a formal agreement between the PLAS program manager and the production division manager.

2.4.2 The Configuration Control Board. The PLAS technical director is the chairperson of the PLAS CCB. This review activity is established at the initiation of preproduction model development.

2.4.3 The Support Environment. The SCAES environment consists of a compatible set of engineering and software development tools that can work with the general engineering data base and special data bases set up for different projects, such as PLAS. The configuration of this support software environment is most carefully controlled by the company data processing and support (DP & S) organization. The support software that interfaces with the data-base management system is most rigidly controlled but there is latitude for engineers to develop special programs restricted to engineering work stations (EWS), that do not generate data for entry into the dynamic engineering data bases. Access keys for controlling entry to the SCAES_PLAS engineering data base are assigned to responsible engineers at the onset of allocated baseline development.

The P-CAMS environment interfaces with a wide variety of CAM and computer-aided test (CAT) systems. These interfaces are critical for the reliable management and administration of company operations. The company DP & S organization manages these interfaces. Any changes must be approved (among other approvals) by the PLAS APM. This review activity is initiated with the development of the preproduction baseline.

Vendor software is used extensively in the supporting software environments of CAES and P-CAMS. Vendor software is also used extensively in the EWS environment supporting SCAES. The management of the vendor software in the EWS environment that is not under the control of the company DP & S organization is initiated by the PLAS TD after the preproduction baseline is established.

2.4.4 SCM Resource Requirements. The resources required for providing configuration management of the PLAS module development and production are embedded in the requirements for training, management oversight, computer resources, administrative support from the engineering support group and DP & S maintenance.

(1) *Training requirements.* Approximately two days training is needed for a new hire engineer to become familiar with use of the data bases and control programs relative to managing configurations. This time is allocated as a part of the overall training program for new hires.

(2) *Management oversight.* Approximately two hours a week are spent on CCB reviews and six hours a week using the PMS control program to schedule analysis and imple-

mentation of system change requests (SCR).

(3) *Computer resources.* Storage requirements for configuration data are a small part of the engineering and production data bases for PLAS modules. The requirements for processor time varies from day to day, but generally does not exceed three minutes of CPU time per day for processing each SCR.

(4) *Support software maintenance.* DP & S is budgeted three man-years effort per year for maintaining the software used for PLAS module configuration management.

2.5 Applicable Policies, Directives, and Procedures

2.5.1 Existing Policies and Procedures. The following company policies are used for configuration management on the PLAS subsystem:

(1) Product Line Identification Numbering—Supplement 2
(2) Corporate Software Protection Policy (Rev 3)
(3) Quality Control Policy for Engineering Data Bases
(4) Production Test Standards
(5) Engineering Standards for Detail Design and Drawings
(6) User's Manual for SCAES
(7) User's Manual for EWS
(8) User's Manual for P-CAMS

2.5.2 New Policies and Procedures To Be Written. The following procedure(s) will be developed for the PLAS project:

(1) Managing of Third-Party Software: Proprietary Marking
(2) PLAS Project Naming Standards

3. SCM Activities

3.1 Configuration Identification. The identification scheme for the PLAS project is developed by the engineering support group and approved by the PLAS TD. The numbering and labeling standards are distributed for project use in the *PLAS Project Naming Standards document.*

3.1.1 Naming Conventions. All data in the SCAES_PLAS engineering data base is arranged and retrievable under the collective identifier PLAS-1800000.

All control level items (programmed logic components and hardware assemblies) are identified within a block of numbers beginning with 532000 and ending with 554000. The engineering support group allocates the numbers for the control level items.

Programmed-logic components have a version description document (VDD) associated with their assigned number. Each assigned number has a preceeding letter identifying the media in which the logic is embedded:

ROM = R
EPROM = E
Diskette[19] = S
Gate arrays = G
PLA = P
Programmable microcontrollers = M

Hardware drawing numbers are assigned to a control level drawing. Parts list for the drawing is made up of part numbers assigned from the 700000 series of numbers.

Reprogrammed logic components keep their basic 1000 number assigned to them in the general SCAES engineering data base. Dash numbers, referencing appropriate VDD, tracks embedded CI, and associated SCR.

3.1.2 Implementation. Identification is assigned to each component and unit defined in the top-level drawing. When an engineer defines a unit, he/she indicates to the program the type of component he/she is defining and the system assigns the appropriate number. Programmed login associated with a defined hardware component or unit is linked to that component's identifier in a *packaging list* associated with the top-level drawing.

Components and units are identified by form, fit, and function (data flow). The engineer defining a component or unit is automatically made *owner* of that component or unit. Changes in the form, fit, or function cannot be made without his/her consent and approval of change. The CAES design tools automatically flag conflicts and force resolution before another of the iterative development baselines can be created.

All system entities associated with the design (specifications, drawings, detail documentation, test data, test procedures, etc) are assigned the appropriate component or unit identifier with which they are associated.

The identifiers assigned in the SCAES_PLAS engineering data base are transferred to the P-CAMS_PLAS production data base at the time

[19]Used for shipping software that executes out of RAM. Software media characteristics may vary but the implementation designator is always *S*.

the preproduction baseline definition effort is initiated.

3.1.3 Ownership Notification Procedures. Filing of software copyright notices for proprietary programmed logic developed for the PLAS project will be performed by Marketing.

Notification to users of the PLAS module copyright will be included in the load module of the software released to the user on the PLAS module diskette. Visual indication of ownership and copyright registration will be displayed at the console when the system is booted, in accordance with Revision 3 (current) of the corporate software protection policy.

All documentation released to customers will be marked with a proprietary notice, vendor license number, or both.

3.2 Configuration Control. Authority for approving changes to baselines varies in accordance with the baseline being changed and the phase of the project.

(1) Authority for approving changes to the *functional baseline* is vested in the PLAS program manager. The PLAS program manager coordinates all changes with the production department manager and with the PLAS TD. This baseline is obsoleted with the initiation of the production baseline.

(2) Authority for approving changes to the *allocated baseline* is vested in the PLAS TD. The PLAS TD coordinates all changes in the allocated baseline with the PLAS program manager and PLAS APM for production. This baseline is shared by SCAES and P-CAMS during the period after the preproduction demonstration is accepted and the production baseline is formally defined.

(3) Authority for approving changes to *developmental baselines* is vested in the PLAS TD. The PLAS TD establishes the developmental baseline criteria, resolves conflicts in allocation and ownership of components or units, and sets schedules for iteration of these baselines.

(4) Authority for approving changes to the *preproduction baseline* is vested in the PLAS TD. The PLAS TD makes changes in allocation and detail design to fit the production facilities on the recommendation of the PLAS APM from production division. Conflicts are resolved by the PLAS program manager.

(5) Authority for approving changes to the *PLAS production baseline* is vested in the PLAS CCB, chaired by the PLAS TD. The PLAS APM and PLAS marketing representative are members of the PLAS CCB. Technical representation from PLAS engineering and PLAS production activities are made when necessary. The PLAS QC representative and production test group representative are permanent members of the PLAS CCB.

Technical review of the system change requests (SCR) is provided by members of the engineering support group who assemble engineering analyses as required, and by members of the PLAS production team who assemble information on the impact of a proposed SCR as required

3.2.1 Change Processing. Changes to the system may originate from the marketing organization (in response to customer desires), from the test group in the production division, or from within the engineering division. Requests for changes are submitted by way of electronic mail using the SCR format provided in the EWS environment. Changes originating from outside the company are entered into the program management system (PMS) by marketing representatives. Internally originated changes are submitted by way of local engineering work stations.

The PMS control system routes SCR to the originator's supervisor for verification when appropriate, and then queues it for review and disposition by the appropriate change authority for the affected baseline. When change requests require further analysis, the change authority routes the SCR (electronically) to the appropriate support group for gathering information. When the support group has assembled a complete analysis package, it is again queued to the appropriate review authority or CCB for disposition. This authority then disposes of the request by indicating approval (providing a schedule and effectivity date of change), deferring it for further analysis or allocation of resources, or disapproving it with reason(s) for disapproval noted.

Approved changes are electronically routed to the PLAS engineering group for implementation.

The tracking of changes is performed in the PMS control system, based on the SCR approval flow status and system change authorizations (SCA), or by extractions from the PLAS data bases in SCAES or P-CAMS to which it has access.

3.2.2 Production Baseline Changes. Changes to the production baseline are made only after changes have been verified in a test environment on a test model of the PLAS module, using simulated test drivers or mock-ups to test the system.

Table 1
Processing Approved Changes

Baseline	Entity	Implemented By	Verified By	Scheduled By
Functional	Document	Engineering	Eng check	Various
Allocated	Document	Engineering	Eng check	Various
Developmental	Document	Engineering	Eng check	TD
	Design data	Engineering	Simulation	TD
	Drawings	Engineering	Eng check	TD
Preproduction	Document	Eng or prod	Eng check	TD
	Design data	Engineering	Simulation	TD
	Drawings	Engineering	Eng check	TD
Production	Document	Production	Test Gp	APM
	Design data	Engineering	Test Gp	APM
	Drawings	Engineering	Eng check	APM

The production test group verifies the changes as operational and authorizes release of the change data from the SCAES_PLAS engineering data base to the P-CAMS_PLAS production data base. The transfer of data is performed by the engineering support group.

3.2.3 PLAS Module Release. Each PLAS module version is released for use in a Quick Stretch System or to individual customers for incorporation into their systems, along with a technical data kit containing the top-level drawings of the system, associated parts lists, and the VDD for control level programmed logic components.

Since PLAS software, released on diskettes, provides the most flexible means of adaptation, provisions exist to release the software VDD independently of the rest of the data packages. This way, revisions to the PLAS functions can be made to PLAS modules in systems released previously. This requires that the configuration of all released modules be maintained in an archive, along with an extraction from the P-CAMS configuration environment containing all support software used in the production and test of that delivery.

3.3 Configuration Status Accounting. The following PLAS configuration status reports are regularly available:

(1) *PLAS Module Development Status.* This is a listing of all configuration items, control level items, and units that are being designed or modified by engineering. The report identifies each unit/control-level-item/CI, status of technical work, outstanding SCR, SCA ready for release, and units or changes released since the last reporting period. This report is generated weekly for the PLAS management team.

(2) *PLAS Module Production Status.* This is a listing of all configuration items, control level items and units that are in production. The report identifies all units in production during the period, SCR/SCA incorporated, scheduled release date (by contract number), and schedule variance. The report is generated weekly for the PLAS management team.

(3) *SCR Status Summary.* This report lists all outstanding SCR that have not been resolved or incorporated into delivered modules. The report lists, for each SCR: CCB action date and disposition; group or department presently responsible for action; status of activity; and schedule for completion. The report is prepared weekly but is available any time the PLAS management team requests it.

(4) *Special Queries.* The report generator of the PMS program provides a query capability that allows anyone to extract the status of:
   (a) Any one SCR
   (b) All open SCR
   (c) All SCR in engineering
   (d) All SCR in production
   (e) PLAS modules in production with associated SCR number

The general query capability for the data-management systems allow formulation of special queries in the PMS control program for interrogating the SCAES_PLAS engineering data base and the P-CAMS_PLAS production data base for information relative to any changes that are in process or that have been released to customers.

3.4 Audits and Reviews

3.4.1 Audits. The PLAS module configuration is audited each time a baseline is established.

(1) *Functional Baseline.* The PLAS program manager is responsible for ascertaining if the reports and design descriptions are complete enough to present to management.

(2) *Allocated Baselines.* The PLAS technical director is responsible for reviewing the designs to ascertain if the designs are complete enough to present to the PLAS program manager. The engineering support group assists the TD in this review.

(3) *Developmental Baselines.* The engineering support group uses the PMS control program to
   (a) Generate set/use type analyses of the detailed designs to uncover outstanding discrepancies and
   (b) Establish design activity cut-offs for a specific iteration.

This group also supports the changes by modifying access codes to the new baseline to restrict entry of changes. The TD reviews the summaries of design activities to estimate technical progress in the design.

(4) *Preproduction Baseline.* Prior to establishing the preproduction baseline, the configuration is again audited by the engineering support group to ascertain that the design of the demonstration meets all functional requirements established by the functional baseline and that all entities generated in the developmental baselines are present or accounted for in the demonstration. The QC representative assists in the review of entities for this baseline.

(5) *Production Baseline.* The QC representative reviews the entities in the P-CAMS_PLAS production data base to ascertain that all functional capabilities demonstrated for the preproduction model and all changes stemming from the review of the demonstration are present in the data base. The production test group reviews the entities to verify that all changes and modifications to the preproduction demonstration have been made to the production data base. The engineering support group performs a comparison of the engineering data base with the production data base to verify that the transfer of data is complete. The PLAS TD is responsible for preparing this audit.

(6) *Shipping Review.* The PLAS module and its associated documentation package is audited prior to shipment to a customer, either as a part of the Quick Stretch System or as an independent line item to a customer.

The functional audit is performed by the QC representative who reviews the PLAS module against the appropriate extracted data from the P-CAM_PLAS library for that item. A representative from the PLAS engineering support group reviews the product to ascertain that the physical configuration of the module and its associated documentation represents

   (a) The specified configuration ordered by the customer
   (b) The corresponding configuration in the P-CAMS_PLAS data base and
   (c) Accurately reflects any changes that have been made to the data base by the PLAS CCB

Discrepancies or problems uncovered in reviews and audits are reported to the PLAS program manager for resolution.

4. Tools, Techniques, and Methodologies

The basic configuration management tool used for PLAS module is the change management program (CMP), which is a part of the PMS. This program supports change management by

(1) Providing the means to enter system/software change requests (SCR)
(2) Forcing reviews by appropriate supervision by way of the electronic mail system
(3) Deriving analytical data from each SCR
(4) Providing for supervision or CCB review and approval, as appropriate
(5) Directing authorized changes to engineering or production supervision
(6) Providing for authorizing of changes to the P-CAMS_PLAS data base
(7) Providing for transfering data from the PLAS engineering data base to the production data base

The SCM tool for establishing the identification scheme for PLAS data bases is resident in the SCAES system. This information is transferred to the production data base during the preproduction phase. It is verified when the production baseline is established.

Order information from marketing is entered into the PLAS production schedule by way of the program management system. The detailed configuration is formatted by the configuration management program in CMS, passed on to the P-CAMS_PLAS data base upon approval of the PLAS program manager, and reviewed for schedule and resource consumption by the PLAS APM. Upon his/her approval, resources are committed to the production configuration. Extractions of this configuration are released for inspection and auditing at time of shipment.

5. Supplier Control

Subcontracted PLAS support software is placed under configuration management after inspection and acceptance by the QC representative.

6. Records Collection and Retention

6.1 Backup Data Base. The engineering data base from SCAES_PLAS is backed-up on a weekly basis and stored in Beskin's storage building during the engineering phase (up to the time the production baseline is established). Following establishment of the production baseline, the engineering data base is backed-up on a monthly basis.

The production data base from P-CAMS_PLAS is backed-up on a weekly basis.

6.2 Archive Data Base. Archive data is maintained for purposes of warranty protection, proprietary data production, and liability insurance. The following data are maintained in on-line optical storage media:

(1) Copies of each baseline data base extracted at the time the baseline is established
(2) Copies of order and configuration data passed from PMS to the production data base for each order and
(3) Copies of each configuration of the data base used for production of customer order
(4) Copies of reviews and audits performed on each production item

Appendix E

References Bibliography

Preface

This Appendix contains selected bibliography pertaining to the subject of software configuration management. The list of publications contains both government and private sector references so users may find material applicable to their situation. Because of the scarcity of literature pertaining to configuration management, and especially to software configuration management, the fullest possible list of references will be useful to the practitioner.

Most of the references contain some information regarding software configuration management. The topic of software configuration management plans is addressed in a subset of these references.

References Bibliography

E1. General Bibliography

[1] BERSOFF, E., HENDERSON, V., and SIEGEL, S. *Software Configuration Management, An Investment in Product Integrity.* Englewood, N.J.: Prentice-Hall, 1980.

[2] BERSOFF, E., HENDERSON, V., and SIEGEL, S. *Software Configuration Management: A Tutorial.* Computer, *IEEE Computer Society Magazine*, vol 12 no 1, Jan 1979.

[3] *Configuration Management Procedures (CMP)*, Global Engineering Documents, 1984.

[4] *IEEE Transactions on Software Engineering*, IEEE Computer Society, vol SE-10, nr 1, Jan 1984.

[5] BUCKLE, J. K. *Software Configuration Management*, New York: The Macmillan Press Ltd, 1982.

[6] DANIELS, M. A. *Principles of Configuration Management*, Advanced Applications Consultants, Inc, 1987.

[7] BABICH, W. A. *Software Configuration Management*: Coordination for Team Productivity, New York: Addison-Wesley, 1986.

E2. Military Standards for SCM[20]

[8] MIL-STD-481A Configuration Control—Engineering Changes, Deviations, and Waivers (Short Form).

[9] MIL-STD-482A Configuration Status Accounting Data Elements and Related Features.

[10] MIL-STD-483A Configuration Management Practices for Systems Equipment, Munitions, and Computer Programs.

[11] MIL-STD-490A Specification Practices.

[12] MIL-STD-499A Engineering Management.

[13] MIL-STD-881A Work Breakdown Structure.

[14] MIL-STD-962A Outline of Forms and Instructions for the Preparation of Military Standards and Military Documents.

[15] MIL-STD-1456 Contractor Configuration Management Plan.

[20] Military Standards may be ordered from the Commanding Officer (Code 301) Naval Publications and Forms Center, 5801 Tabor Avenue, Philadelphia, PA 19120.

[16] MIL-STD-1521B Technical Reviews and Audits for Systems, Equipments, and Computer Programs.

E3. Department of Defense Standards

[17] DoD-STD-480 Configuration Control-Engineering Changes, Deviations and Waivers.

[18] DoD-STD-1467 Software Support Environment.

[19] DoD-STD-2167 Defense System Software Development.

[20] DoD-STD-2168 Software Quality Evaluation.

[21] DoD-STD-7935 Automated Data Systems Documentation.

E4. Military Specification

[22] MIL-D-1000B Drawings, Engineering, and Associated List.

[23] MIL-S-83490 Specifications, Types and Forms.

E5. Department of Defense Directives

[24] DoDD 4120.21 Specifications and Standards Applications.

[25] DoDD 5000.1 Major Systems Applications.

[26] DoDD 5000.19L Acquisition Management Systems and Data Requirements Control List.

[27] DoDD 5000.39 Acquisition and Management of Integrated Logistic Support for Systems and Equipment.

[28] DoDD 5010.19 Configuration Management.

[29] DoDD 7920.1 Life Cycle Management of Automated Information Systems (AIS).

E6. Department of Defense Instructions

[30] DoDI 5000.2 Major Systems Acquisition Process.

[31] DoDI 5000.38 Production Readiness Reviews.

[32] DoDI 7045.7 The Planning, Programming and Budgeting System.

[33] DoDI 7935.1 DoD Automated Data Systems Documentation Standards.

E7. US Government Publications[21]

[34] DoD Configuration Management Standardization Program, (CMAM) Plan.

[35] DoD Trusted Computer System Evaluation Criteria, CSC-STD-001-83, 15 Aug 1983.

[36] *NASA Handbook 8040.2.* System Engineering Management Guide, Defense System Management College, 1983. Configuration Management, Management Instruction, GMI8040.1A. NASA Goddard Space Flight Center.

[37] Federal Information Processing Standards (FIPS) *Publication 106.* Guideline on Software Maintenance, National Bureau of Standards. Institute for Computer Sciences and Technology, 1984.

[38] MARTIN, R. and OSBORNE, W. *Special Publication 500-106,* Guidance on Software Maintenance. National Bureau of Standards, Institute for Computer Sciences and Technology, 1983.

[39] McCALL, JIM, HERNDON, MARY, and OSBORNE, WILMA. *Special Publication 500-129,* Software Maintenance Management, National Bureau of Standards, Institute for Computer Sciences and Technology, 1985.

[21] Copies of these publications can be obtained from the Superintendent of Documents, US Governmental Printing Office, Washington, DC 20402.

E8. Electronic Industries Association Publications[22]

[40] EIA CMB 4-1a (Sept 1984), Configuration Management Definitions for Digital Computer Programs.

[41] EIA CMB 4-2 (June 1981), Configuration Identification for Digital Computer Programs.

[42] EIA CMB 4-3 (Feb 1981), Computer Software Libraries.

[43] EIA CMB 4-4 (May 1982), Configuration Change Control for Digital Computer Programs.

[44] EIA CMB 5 (April 1973), Subcontractor/ Vendor Configuration Management and Technical Data Requirements.

[22] EIA publications can be obtained from the Standards Sales Department, Electronics Industries Association, 2001 Eye Street, NW, Washington DC 20006.

E9. American Defense Preparedness Association Publications[23]

[45] Proceedings of the 24th Annual Meeting, Technical Documentation Division, May 1982, Denver, Colorado.

[46] Proceedings of the 25th Annual Meeting, Technical Documentation Division, May 1983, Ft. Monroe, Virginia.

[47] Proceedings of the 26th Annual Meeting, Technical Documentation Division, May 1984, San Antonio, Texas.

[23] Copies of these publications can be obtained from the American Defense Preparedness Association, 1700 N. Monroe St, Suite 900, Arlington, VA 22209.

Recognized as an
American National Standard (ANSI)

ANSI/IEEE
Std 1058.1-1987
(Reaffirmed 1993)

An American National Standard

IEEE Standard for Software Project Management Plans

Sponsor

The Software Engineering Technical Committee
of the
Computer Society of the IEEE

Approved December 10, 1987
Reaffirmed December 2, 1993

IEEE Standards Board

Approved October 6, 1988

American National Standards Institute

The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017, USA

Recognized as an
American National Standard (ANSI)

ANSI/IEEE
Std 1058.1-1987
(Reaffirmed 1993)

An American National Standard

IEEE Standard for Software Project Management Plans

Sponsor

The Software Engineering Technical Committee
of the
Computer Society of the IEEE

Approved December 10, 1987
Reaffirmed December 2, 1993

IEEE Standards Board

Approved October 6, 1988

American National Standards Institute

The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017, USA

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
345 East 47th Street
New York, NY 10017
USA

Foreword

(This Foreword is not a part of IEEE Std 1058.1-1987, IEEE Standard for Software Project Management Plans.)

Purpose

This standard specifies the format and contents of software project management plans. It does not specify the exact techniques to be used in developing project plans, nor does it provide examples of project management plans. Each organization that uses this standard should develop a set of practices and procedures to provide detailed guidance for preparing and updating software project management plans based on this standard. These detailed practices and procedures should take into account the environmental, organizational, and political factors that influence application of the standard.

Not all software projects are concerned with development of source code for a new software product. Some software projects consist of a feasibility study and definition of product requirements. Other projects terminate upon completion of product design, and some projects are concerned with major modifications to existing software products. This standard is applicable to all types of software projects; applicability is not limited to projects that develop operational versions of new products. Application of this standard is not limited by project size. Small projects may require less formality in planning than large projects, but all components of this standard should be addressed by every software project.

Software projects are sometimes component parts of larger projects. In these cases, the software project management plan may be a separate component of a larger plan or it may be merged into the system level project management plan.

Overview

This standard contains three sections. Section 1 defines the scope of the standard and provides references to other IEEE standards that should be followed when applying this standard. Section 2 provides definitions of terms that are used throughout the standard. Section 3 contains an overview and a detailed specification of the standard, including required components that must be included, and optional components that may be included in project plans based on this standard. The sequence of project plan elements presented in Section 3 does not imply that project plans should be developed in the order of presentation. In most instances, project plans based on this standard will be developed by repeated iteration and refinement of the various elements in the plan.

Audience

This standard is intended for use by software project managers and other personnel who prepare and update project plans and monitor adherence to those plans.

Evolution of Plans

Developing the initial version of the software project management plan should be one of the first activities to be completed on a software project. As the project evolves, the nature of the work to be done and the decomposition of work will be better understood. The project management plan must be updated periodically to reflect the evolving situation. Thus, each version of the plan should be placed under change control, and each version should contain a schedule for subsequent updates to the plan.

Terminology

This standard follows the IEEE Guide to Standards Development. In particular, the words shall, must, and the imperative form identify mandatory material within the standard. The words should, might, and may identify optional material.

History

The project authorization request for development of this standard was approved by the IEEE Standards Board on December 13, 1984. Modification of the authorization request was approved in September, 1986. Ten meetings were held within the United States and internationally between September, 1984, and September, 1986. These meetings produced the draft submitted for balloting in December, 1986.

Contributors

This standard was developed by the Software Project Management Plans Working Group of the Software Engineering Standards Subcommittee of the Computer Society of the IEEE. The following individuals contributed to the development of this standard:

Richard H. Thayer, *Chairman* **Richard E. Fairley,** *Co-Chairman*
Gary L. Whitten, *Secretary*

Steering Committee

Ronald L. Atchley
H. Jack Barnard
François Coallier
Hosein Fallah
Richard Johansson
Kari Kansala
Robert F. Metz
Patrick A. Rich
Hans Schaefer

Working Group

Bakul Banerju
George J. Bozoki
V. Churchill
Peter Coad
P. I. Davis
Raymond Day
T. Debling
J. Deleo
Ake Dovstram
R. Ferreol
Galwin Ferwin
Cheng Hu
John A. King
Thomas M. Kurihara
F. C. Lim
Richard W. MacDonald
Roger Martin
Randy Paddock
Francoise Perrodeau
Robert Poston
David Schultz
Robert Shillato
Daniel Solis
George Spidel
Richard Van Tilburg
Delores Wallace
David Weiss
Lauri Werth
Janusz Zalweski

Supporting Organizations

The following organizations provided support for development of this standard:

AT&T Bell Laboratories
AT&T Information Systems
Bell Canada
California State University, Sacramento
Center for Industrial Research, Norway
Compagnie D'Informatique Militaire, France
Computer Sciences Corporation
Goodyear Atomic Corporation
Hughes Aircraft Company
Institute of Atomic Energy, Poland
Intercon Systems Corporation
Lockheed Missiles & Space Co.
Mirror Systems, Inc.
National Bureau of Standards
NCR Corporation
Programming Environments, Inc.
Standard Telecommunications Labs
System Development Corporation
Technical Research Centre of Finland
Teleindustrier AB, Sweden
U.S. Department of Commerce
U.S. Department of Transportation
U.S. Naval Research Laboratories
University of Nevada, Las Vegas
University of Texas, Austin
Wang Institute of Graduate Studies

The Software Engineering Standards Subcommittee comprised the balloting committee that approved this document for submission to the IEEE Standards Board, and at the time of balloting was composed of the following persons:

A. F. Ackerman
Richard L. Aurbach
Motoei Azuma
H. J. Barnard
J. Behm
H. Ronald Berlack
Michael A. Blackledge
Giles Bracon
J. Joseph Brandt
Kathleen L. Briggs
A. Winsor Brown
F. Buckley
L. J. Call
H. Carl
John W. Center
John Chihorek
T. S. Chow
J. K. Chung
Won L. Chung
Antonio M. Cicu
Judith Clapp
Peter Coad, Jr.
François Coallier
Christopher M. Cooke
Richard Cotter
T. Daughtrey
Peter Denny
F. M. Discenzo
David A. Dobraiz
David Doty
Einar Dragstedt
William Dupres
Albert D. DuRee
Mary Eads
L. E. Egan
W. D. Ehrenberger
Steven R. Eisen
Walter Ellis
Caroline Evans
Richard E. Fairley
David W. Favor
Joan Feld
John Fendrich
Glenn S. Fields
Violet Foldes
T. Foote-Lennox
Joel Forman
Julian Forster
Richard C. Fries
F. K. Gardner
L. B. Gardner
David Gilperin
Shirley Gloss-Soler
J. G. Glynn
Andrej Grebenc
Victor Guarnera
Lawrence M. Gunther
David A. Gustafson
J. A. Haksi
Jeffrey Hickey
John W. Horch
Cheng Hu
Laurel V. Kaleda
Harry Kalmbach
Daniel E. Klingler
Shaye Koenig
Joseph A. Krupinski
Joan Kundig
T. Kurihara
Lak Ming Lam
John B. Lane
Robert A. C. Lane
G. Larsen
F. C. Lim
Bertil Lindberg
B. Livson
Austin Maher
Paulo Cesar Marcondes
Nicholas Marselos
Roger Martin
John McArdle
J. A. McCall
R. McDowell
Manijeh Moghis
Charles S. Mooney
Gary D. Moorhead
D. D. Morton
G. T. Morun
Hiranobu Nagano
Geraldine Neidhart
Dennis Nickle
Wilma M. Osborne
Michael T. Perkins
W. E. Perry
John Petraglia
Donald J. Pfeiffer
I. C. Pyle
Thomas S. Radi
Salim Ramji
Jean-Claude Rault
Meir Razy
Donald Reifer
Patrick A. Rich
R. San Roman
John C. Rowe
Margaret Rumley
Julio G. Sanz
Steven Schach
Wolf A. Schnoege
Norman Schneidewind
David Schultz
Gregory D. Schumacker
Leonard W. Seagren
Gerard P. Shabe
Robert Shillato
David Siefert
William J. Singer
Jacob Slonim
H. M. Sneed
V. Srinivas
G. Wayne Staley
Franklin M. Sterling
Mary Jane Stoughton
W. G. Sutcliffe
Michael H. Taint
Richard H. Thayer
Paul U. Thompson
Terrence L. Tillmans
Valentin Tirman
G. R. Trebble
C. L. Troyanowski
William S. Turner, III
Robert Urling
David Usechak
Thomas E. Vollman
Dolores R. Wallace
John P. Walter
Dale R. Webdale
Charles J. Wertz
Peter J. Weyman
Allan Whitaker
Gary L. Whitten
Andrea S. Williams
Patrick J. Wilson
W. Martin Wong
Dennis L. Wood
Paul R. Work
Nancy Yavne
Natalie C. Yopronka
Leon Young
Donald Zelenu
Hugh Zettel
Peter Zoll

When the IEEE Standards Board approved this standard on December 10, 1987, it had the following membership:

Donald C. Fleckenstein, *Chairman* **Marco W. Migliaro,** *Vice Chairman*
Andrew G. Salem, *Secretary*

James H. Beall
Dennis Bodson
Marshall L. Cain
James M. Daly
Stephen R. Dillon
Eugene P. Fogarty
Jay Forster
Kenneth D. Hendrix
Irvin N. Howell
Leslie R. Kerr
Jack Kinn
Irving Kolodny
Joseph L. Koepfinger*
Edward Lohse
John May
Lawrence V. McCall
L. Bruce McClung
Donald T. Michael*
L. John Rankine
John P. Riganati
Gary S. Robinson
Frank L. Rose
Robert E. Rountree
William R. Tackaberry
William B. Wilkens
Helen M. Wood

*Member emeritus

Contents

IEEE Standard for Software Project Management Plans

1. Scope and References

1.1 Scope. This standard prescribes the format and content of software project management plans. A software project management plan is the controlling document for managing a software project; it defines the technical and managerial processes necessary to satisfy the project requirements.

This standard may be applied to all types of software projects. Use of this standard is not restricted by the size, complexity, or criticality of the software product. This standard is applicable to all forms of product delivery media, including firmware, embedded systems code, programmable logic arrays, and software-in-silicon. This standard can be applied to any and all segments of a software product lifecycle.

This standard identifies the minimal set of elements that shall appear in all software project management plans. In order to conform to this standard, software project management plans must adhere to the format and content for project plans specified in the standard. However, users of this standard may incorporate other elements by appending additional sections or subsections to their project management plans. In any case, the numbering scheme of the required sections and subsections must adhere to the format specified in this standard. Various sections and subsections of a software project management plan may be included in the plan by direct incorporation or by reference to other plans and documents.

This standard for software project management plans incorporates and subsumes the software development plans described in ANSI/IEEE Std 729-1983 [1][1] and ANSI/IEEE Std 730-1984 [2].

[1]The numbers in brackets correspond to those of the references in 1.2.

1.2 References. The standards listed here should be consulted when applying this standard. The latest revisions shall apply.

[1] ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology.[2]

[2] ANSI/IEEE Std 730-1984, IEEE Standard for Software Quality Assurance Plans.

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

[4] ANSI/IEEE Std 829-1983, IEEE Standard for Software Test Documentation.

[5] ANSI/IEEE Std 983-1986, IEEE Guide for Software Quality Assurance Planning.

[6] ANSI/IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans.

2. Definitions

The definitions listed here establish meanings within the context of this standard. Definitions of other terms that may be appropriate within the context of this standard can be found in ANSI/IEEE Std 729-1983 [1].

activity. A major unit of work to be completed in achieving the objectives of a software project. An activity has precise starting and ending dates, incorporates a set of tasks to be completed, consumes resources, and results in work products. An activity may contain other activities in a hierarchical manner.

[2]ANSI/IEEE publications are available from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018; or from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

baseline. A work product that has been formally reviewed and agreed upon, and that can be changed only through formal change control procedures. A baseline work product may form the basis for further work activity(s).

customer. The individual or organization that specifies and accepts the project deliverables. The customer may be internal or external to the parent organization of the project, and may or may not be the end user of the software product. A financial transaction between customer and developer is not necessarily implied.

project agreement. A document or set of documents agreed to by the designated authority for the project and the customer. Documents in a project agreement may include some or all of the following: a contract, a statement of work, system engineering specifications, user requirement specifications, functional specifications, the software project management plan, a business plan, or a project charter.

project deliverables. The work product(s) to be delivered to the customer. The quantities, delivery dates, and delivery locations are specified in the project agreement.

project function. An activity that spans the entire duration of a software project. Examples of project functions include project management, configuration management, quality assurance, and verification and validation.

review. A meeting at which a work product or a set of work products is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.

software project. The set of all project functions, activities, and tasks, both technical and managerial, required to satisfy the terms and conditions of the project agreement. A software project may be self-contained or may be part of a larger project. A software project may span only a portion of the software product lifecycle.

software project management. The process of planning, organizing, staffing, monitoring, controlling, and leading a software project.

software project management plan. The controlling document for managing a software project. A software project management plan defines the technical and managerial project functions, activities, and tasks necessary to satisfy the requirements of a software project, as defined in the project agreement.

SPMP. Software project management plan.

task. The smallest unit of work subject to management accountability. A task is a well-defined work assignment for one or more project members. The specification of work to be accomplished in completing a task is documented in a work package. Related tasks are usually grouped to form activities.

work package. A specification for the work to be accomplished in completing an activity or task. A work package defines the work product(s), the staffing requirements, the expected duration, the resources to be used, the acceptance criteria for the work products, the name of the responsible individual, and any special considerations for the work.

work product. Any tangible item that results from a project function, activity, or task. Examples of work products include customer requirements, project plan, functional specifications, design documents, source and object code, users' manuals, installation instructions, test plans, maintenance procedures, meeting minutes, schedules, budgets, and problem reports. Some subset of the work products will form the set of project deliverables.

3. Software Project Management Plans

The individual or organization responsible for a software project shall also be responsible for the software project management plan (hereafter referred to as the SPMP). This section of the standard describes each of the essential elements of a SPMP. These elements shall be ordered in the sequence of sections and subsections prescribed in Table 1.

Table 1
Software Project Management Plan Format

Title Page
Revision Chart
Preface
Table of Contents
List of Figures
List of Tables

1. Introduction
   - 1.1 Project Overview
   - 1.2 Project Deliverables
   - 1.3 Evolution of the SPMP
   - 1.4 Reference Materials
   - 1.5 Definitions and Acronyms

2. Project Organization
   - 2.1 Process Model
   - 2.2 Organizational Structure
   - 2.3 Organizational Boundaries and Interfaces
   - 2.4 Project Responsibilities

3. Managerial Process
   - 3.1 Management Objectives and Priorities
   - 3.2 Assumptions, Dependencies, and Constraints
   - 3.3 Risk Management
   - 3.4 Monitoring and Controlling Mechanisms
   - 3.5 Staffing Plan

4. Technical Process
   - 4.1 Methods, Tools, and Techniques
   - 4.2 Software Documentation
   - 4.3 Project Support Functions

5. Work Packages, Schedule, and Budget
   - 5.1 Work Packages
   - 5.2 Dependencies
   - 5.3 Resource Requirements
   - 5.4 Budget and Resource Allocation
   - 5.5 Schedule

Additional Components

Index

Appendices

The ordering of SPMP elements presented in Table 1 is not meant to imply that the sections and subsections must be developed in that order. The order of presentation is intended for ease of use, not as a guide to the order of preparing the various elements of a SPMP. The sections and subsections of a SPMP may be included by direct incorporation or by reference to other plans and documents.

Detailed descriptions of each section and subsection in a SPMP are presented in sections 3.1 through 3.5 of this standard. Certain additional components may be included in a SPMP. Additional components are described in section 3.6.

Each version of a SPMP based on this standard shall contain a title and a revision notice sufficient to uniquely identify the document. Revision information may include the project name, version number of the plan, date of release, approval signature(s), a list of pages that have been changed in the current version of the plan, and a list of version numbers and dates of release of all previous versions of the plan.

The preface of a SPMP based on this standard shall describe the purpose, indicate the scope of activities, and identify the intended audience for the SPMP. A Table of Contents, and lists of the Figures and Tables in the SPMP shall be included in every SPMP, as indicated in Table 1.

3.1 Introduction (Section 1 of the SPMP). This section of the SPMP shall provide an overview of the project and the product, a list of project deliverables, the plan for development and evolution of the SPMP, reference materials for the SPMP, and definitions and acronyms used within the SPMP.

3.1.1 Project Overview (1.1 of the SPMP). This subsection of the SPMP shall provide a concise summary of the project objectives, the product to be delivered, major work activities, major work products, major milestones, required resources, and master schedule and budget. The project overview shall also describe the relationship of this project to other projects, as appropriate. This overview shall not be construed as an official statement of product requirements. Reference to the official statement of product requirements shall be provided in this subsection of the SPMP.

3.1.2 Project Deliverables (1.2 of the SPMP). This subsection of the SPMP shall list all of the items to be delivered to the customer, the delivery dates, delivery locations, and quantities required to satisfy the terms of the project agreement. This list of project deliverables shall not be construed as an official statement of project requirements.

3.1.3 Evolution of the SPMP (1.3 of the SPMP). This subsection of the SPMP shall specify the plans for producing both scheduled and unscheduled updates to the SPMP. Methods of disseminating the updates shall be specified. This subsection shall also specify the mechanisms used to place the initial version of the SPMP under change control and to control subsequent changes to the SPMP.

3.1.4 Reference Materials (1.4 of the SPMP). This subsection of the SPMP shall provide a complete list of all documents and other sources of information referenced in the SPMP. Each document should be identified by title, report number, date, author, and publishing organization. Other sources of information, such as electronic files, shall be identified in an unambiguous manner using identifiers such as date and version number. Any deviations from referenced standards or policies shall be identified and justifications shall be provided.

3.1.5 Definitions and Acronyms (1.5 of the SPMP). This subsection of the SPMP shall define, or provide references to the definition of all terms and acronyms required to properly interpret the SPMP.

3.2 Project Organization (Section 2 of the SPMP). This section of the SPMP shall specify the process model for the project, describe the project organizational structure, identify organizational boundaries and interfaces, and define individual responsibilities for the various project elements.

3.2.1 Process Model (2.1 of the SPMP). This subsection of the SPMP shall define the relationships among major project functions and activities by specifying the timing of major milestones, baselines, reviews, work products, project deliverables, and sign-offs that span the project. The process model may be described using a combination of graphical and textual notations. The process model must include project initiation and project termination activities.

3.2.2 Organizational Structure (2.2 of the SPMP). This subsection of the SPMP shall describe the internal management structure of the project. Graphical devices such as hierarchical organization charts or matrix diagrams may be used to depict the lines of authority, responsibility, and communication within the project.

3.2.3 Organizational Boundaries and Interfaces (2.3 of the SPMP). This subsection of the SPMP shall describe the administrative and managerial boundaries between the project and each of the following entities: the parent organization, the customer organization, subcontracted organizations, or any other organizational entities that interact with the project. In addition, the administrative and managerial interfaces of the project support functions, such as configuration management, quality assurance, and verification and validation shall be specified in this subsection.

3.2.4 Project Responsibilities (2.4 of the SPMP). This subsection of the SPMP shall identify and state the nature of each major project function and activity, and identify the individuals who are responsible for those functions and activities. A matrix of functions and activities versus responsible individuals may be used to depict project responsibilities.

3.3 Managerial Process (Section 3 of the SPMP). This section of the SPMP shall specify management objectives and priorities; project assumptions, dependencies, and constraints; risk management techniques; monitoring and controlling mechanisms to be used; and the staffing plan.

3.3.1 Management Objectives and Priorities (3.1 of the SPMP). This subsection of the SPMP shall describe the philosophy, goals, and priorities for management activities during the project. Topics to be specified may include, but are not limited to, the frequency and mechanisms of reporting to be used; the relative priorities among requirements, schedule, and budget for this project; risk management procedures to be followed; and a statement of intent to acquire, modify, or use existing software.

3.3.2 Assumptions, Dependencies, and Constraints (3.2 of the SPMP). This subsection of the SPMP shall state the assumptions on which the project is based, the external events the project is dependent upon, and the constraints under which the project is to be conducted.

3.3.3 Risk Management (3.3 of the SPMP). This subsection of the SPMP shall identify and assess the risk factors associated with the project. This subsection shall also prescribe mechanisms for tracking the various risk factors and implementing contingency plans. Risk factors that should be considered include contractual risks, technological risks, risks due to size and complexity of the product, risks in personnel acquisition and retention, and risks in achieving customer acceptance of the product.

3.3.4 Monitoring and Controlling Mechanisms (3.4 of the SPMP). This subsection of the SPMP shall define the reporting mechanisms, report formats, information flows, review and audit mechanisms, and other tools and techniques to be used in monitoring and controlling adherence to the SPMP. Project monitoring should occur at the level of work packages. The relationship of monitoring and controlling mechanisms to the project support functions shall be delineated in this subsection of the SPMP (see 3.4.3).

3.3.5 Staffing Plan (3.5 of the SPMP). This subsection of the SPMP shall specify the numbers and types of personnel required to conduct the project. Required skill levels, start times, duration of need, and methods for obtaining, training, retaining, and phasing out of personnel shall be specified.

3.4 Technical Process (Section 4 of the SPMP). This section of the SPMP shall specify the technical methods, tools, and techniques to be used on the project. In addition, the plan for software documentation shall be specified, and plans for project support functions such as quality assurance, configuration management, and verification and validation may be specified.

3.4.1 Methods, Tools, and Techniques (4.1 of the SPMP). This subsection of the SPMP shall specify the computing system(s), development methodology(s), team structure(s), programming language(s), and other notations, tools, techniques, and methods to be used to specify, design, build, test, integrate, document, deliver, modify or maintain or both (as appropriate) the project deliverables. In ad-

dition, the technical standards, policies, and procedures governing development or modification or both of the work products and project deliverables shall be included, either directly or by reference to other documents.

3.4.2 Software Documentation (4.2 of the SPMP). This subsection of the SPMP shall contain either directly or by reference, the documentation plan for the software project. The documentation plan shall specify the documentation requirements, and the milestones, baselines, reviews, and sign-offs for software documentation. The documentation plan may also contain a style guide, naming conventions and documentation formats. The documentation plan shall provide a summary of the schedule and resource requirements for the documentation effort. ANSI/IEEE Std 829-1983 [4] provides a standard for software test documentation.

3.4.3 Project Support Functions (4.3 of the SPMP). This subsection of the SPMP shall contain, either directly or by reference, plans for the supporting functions for the software project. These functions may include, but are not limited to, configuration management [3]; software quality assurance [2] and [5]; and verification and validation [6]. Plans for project support functions shall be developed to a level of detail consistent with the other sections of the SPMP. In particular, the responsibilities, resource requirements, schedules, and budgets for each supporting function shall be specified. The nature and type of support functions required will vary from project to project; however, the absence of a software quality assurance, configuration management, or verification and validation plan shall be explicitly justified in project plans that do not include them.

3.5 Work Packages, Schedule, and Budget (Section 5 of the SPMP). This section of the SPMP shall specify the work packages, identify the dependency relationships among them, state the resource requirements, provide the allocation of budget and resources to work packages, and establish a project schedule.

3.5.1 Work Packages (5.1 of the SPMP). This subsection of the SPMP shall specify the work packages for the activities and tasks that must be completed in order to satisfy the project agreement. Each work package shall be uniquely identified; identification may be based on a numbering scheme and descriptive titles. A diagram depicting the breakdown of activities into subactivities and tasks (a work breakdown structure) may be used to depict hierarchical relationships among work packages.

3.5.2 Dependencies (5.2 of the SPMP). This subsection of the SPMP shall specify the ordering relations among work packages to account for interdependencies among them and dependencies on external events. Techniques such as dependency lists, activity networks, and the critical path method may be used to depict dependencies among work packages.

3.5.3 Resource Requirements (5.3 of the SPMP). This subsection of the SPMP shall provide, as a function of time, estimates of the total resources required to complete the project. Numbers and types of personnel, computer time, support software, computer hardware, office and laboratory facilities, travel, and maintenance requirements for the project resources are typical resources that should be specified.

3.5.4 Budget and Resource Allocation (5.4 of the SPMP). This subsection of the SPMP shall specify the allocation of budget and resources to the various project functions, activities, and tasks. An earned value scheme may be used to allocate budget and resources, and to track expenditures and resource utilization.

3.5.5 Schedule (5.5 of the SPMP). This subsection of the SPMP shall provide the schedule for the various project functions, activities, and tasks, taking into account the precedence relations and the required milestone dates. Schedules may be expressed in absolute calendar time or in increments relative to a key project milestone.

3.6 Additional Components. Certain additional components may be required. These may be included by appending additional sections or subsections to the SPMP. However, the numbering scheme for the required sections and subsections must adhere to the format specified in this standard. Additional items of importance on any particular project may include subcontractor management plans, security plans, independent verification and validation plans, training plans, hardware procurement plans, facilities plans, installation plans, data conversion plans,

system transition plans, or the product maintenance plan. If present, additional components must be developed in a format and to a level of detail consistent with the required sections of the SPMP.

3.6.1 Index. An index to the key terms and acronyms used throughout the SPMP is optional, but recommended to improve usability of the SPMP.

3.6.2 Appendices. Appendices may be included, either directly or by reference, to provide supporting details that could detract from the SPMP if included in the body of the SPMP.

Recognized as an
American National Standard (ANSI)

IEEE
Std 1074-1991

IEEE Standard for Developing Software Life Cycle Processes

Sponsor

Software Engineering Standards Subcommittee of the Technical Committee on Software Engineering of the IEEE Computer Society

Approved September 26, 1991

IEEE Standards Board

Approved April 20, 1992

American National Standards Institute

Abstract: The set of activities that constitute the processes that are mandatory for the development and maintenance of software, whether stand-alone or part of a system, is set forth. The management and support processes that continue throughout the entire life cycle, as well as all aspects of the software life cycle from concept exploration through retirement, are covered. Associated input and output information is also provided. Utilization of the processes and their component activities maximizes the benefits to the user when the use of this standard is initiated early in the software life cycle. This standard requires definition of a user's software life cycle and shows its mapping into typical software life cycles; it is not intended to define or imply a software life cycle of its own.

Keywords: project management process, project monitoring and control process, software development process, software implementation process, software installation process, software life cycle, software life cycle model process, software life cycle process, software maintenance process, software operation and support process, software post-development process, software pre-development process, software quality management process, software requirements process, software retirement process, software system allocation process

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Library of Congress Cataloging in Publication information will be found on the following page.

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08555-1331
USA

Library of Congress Cataloging in Publication Data

Institute of Electrical and Electronics Engineers, Inc., the.
IEEE standard for developing software life cycle processes / sponsor, Software Engineering Standards Subcommittee of the Technical Committee on Software Engineering of the IEEE Computer Society.
p. cm.
"Approved September 26, 1991, IEEE Standards Board."
"IEEE Std 1074-1991."
Includes index.
ISBN 1-55937-170-6
1. Computer software—Development—Standards—United States. 2. Software maintenance—Standards—United States. I. IEEE Computer Society. Software Engineering Standards Subcommittee. II. IEEE Standards Board. III. Title
QA76.76.D47I545 1992
005.1'021873—dc20 91-41510
CIP

Foreword

(This Foreword is not a part of IEEE Std 1074-1991, IEEE Standard for Developing Software Life Cycle Processes.)

This Foreword is intended to provide the reader with some background into the rationale used to develop this standard. This information is being provided to aid in the understanding and usage of the standard. The Foreword is nonbinding.

Purpose

This is a standard for the Processes of software development and maintenance. This standard requires definition of a user's software life cycle and shows mapping into typical software life cycles, but it is not intended to define or imply a software life cycle of its own. This standard applies to the management and support Processes that continue throughout the entire life cycle, as well as all aspects of the software life cycle from concept exploration through retirement. Utilization of these Processes, and their component Activities, maximizes the benefits to the user when the use of this standard is initiated early in the software life cycle.

Software that has proceeded past the initialization phase when this standard is invoked should gradually comply with the standard.

This standard was written for any organization responsible for managing and developing software. It will be useful to project managers, software developers, quality assurance organizations, purchasers, users, and maintainers. Since it was written to consider both software and its operating environment, it can be used where software is the total system or where software is embedded in a larger system.

Terminology

The words *shall, must,* and the imperative form identify the mandatory (essential) material within this standard. The words *should* and *may* identify optional (conditional) material. As with other IEEE Software Engineering Standards, the terminology in this document is based on IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology (ANSI). To avoid inconsistency when the Glossary is revised, the definitions are not repeated in this document. New terms and modified definitions are included, however.

History

Work on this standard began in August 1984. A total of 15 meetings produced the draft submitted for ballot in January 1990. Two additional meetings were held to resolve comments and negative ballots, and the document was resubmitted for recirculation in February 1991.

Participants

This standard was developed by a working group consisting of the following members who attended two or more meetings, provided text, or submitted comments on more than two drafts of the standard:

David J. Schultz
Chair

John W. Horch
Vice Chair

Dennis E. Nickle
Secretary

Jean A. Gilmore
Group Leader

Art Godin
Group Leader

Lynn D. Ihlenfeldt
Group Leader

Robert W. Shillato
Group Leader

Joseph J. Guidos,* David W. Burnett
Configuration Managers

Michael Buckley
Editor

Joe Albano
Tom Antczak
Susan Burgess
David Burrows
Dan Chang
Paul Christensen
Raymond Day
Marvin Doran
Mike Ellwood
Arden Forrey
John Graham
Daniel Gray
Rob Harker
Carolyn Harrison
Peter Harvey
Eric Hensel
Denise Holmes
George Jackelen
Linell Jones
Laurel Kaleda
Phil Keys
Tom Kurihara
Bill Mar
Darrell Marsh
Leroy May
Denis Meredith
Manijeh Moghis
Richard Morton
Gerry Neidhart
Brian Nejmeh
Hans Schaefer
Isaac Shadman
Kelley Stalder
David Taylor
Leonard Tripp
George Tucker
Odo Wang
Richard Werling

*Deceased

Contributors

The following individuals also contributed to the development of the standard by attending one meeting or providing comments on one or two drafts:

Scott Allen
Kathleen Alley
Robert Baris
Dwight Bellinger
H. Ronald Berlack
William Blum
Robert Both
Fletcher Buckley
John Chihorek
Geoff Crellin
M. A. Daniels
Geoffrey Darnton
Leonard DeBaets
Ingrid deBuda
Kristin Dittmann
Gary Driver
Susan East
Leo Egan
Violet Foldes
Roger Fujii
Michael Garrard
Yair Gershkovitch
Ole Golubjatnikov
Jim Harkins
James Heil
Cheng Hu
Jim Hughes
Suzana Hutz
Ron Hysom
Phyllis Illyefalvi
Corbin Ingram
Ramon Izbinsky
Tom Jepson
Jia Yaoliang
Allen Jin
David Johnson
Richard Karcich
E. Klamm
Rick Kuhn
Stephan Lacasse
F. C. Lim
Ben Livson
Theresa Mack
Karen Mackey
Stan Magee
John Marciniak
Richard McClellan
Neal Mengel
Rocco Novak
George O'Connell
John Patchen
Jeff Pattee
Virgil Polinski
P. A. Rhodes
Bill Romstadt
Benson Scheff
Richard Schmidt
David Schwartz
Carl Seddio
Paul Sevcik
Randy Shipley
Kimberly Steele
Karen Steelman
Jim Stoner
Wayne Sue
Ann Sullivan
Daniel Teichroew
Russell Theisen
Donna Thomas
George Tice
R. Van Tilburg
Graham Tritt
Dolores Wallace
Valerie Winkler
Grady Wright
Fred Yonda
Lin Zucconi

Balloting Committee

The following persons were on the balloting committee that approved this document for submission to the IEEE Standards Board:

Joseph Albano, Jr.
Eleanor Antreassian
Au Man Kit
Emanuel R. Baker
David Barber
H. Jack Barnard
Dwight Bellinger
M. Ben-Menachem
H. Ronald Berlack
Michael Blackledge
William J. Boll
Ronald Braun
Bruce Brocka
Winsor A. Brown
Fletcher Buckley
Susan Burgess
David Burnett
David Burrows
Kay Bydalek
Clyde Camp
James L. Cartwright
John Chihorek
S. V. Chiyyarath
Won L. Chung
François Coallier
Christopher Cooke
Michael A. Daniels
Geoffrey Darnton
Taz Daughtrey
Bostjan K. Derganc
Audrey Dorofee
Carl E. Dragstedt
Mary L. Eads
Vera D. Edelstein
L. G. Egan
Caroline L. Evans
John W. Fendrich
A. L. Fleetwood
Roger G. Fordham
Joel J. Forman
Julian Forster
Kirby Fortenberry
Thomas J. Fouser
Clairmont Fraser
Roger Fujii
Casey K. Fung
Yair Gershkovitch
Jean Gilmore
Shirley A. Gloss-Soler
Arthur Godin
Richard Goldsmith
Ole Golubjatnikov
Steven Grady
J. Kaye Grau
Thomas A. Grobicki
Victor M. Guarnera
Cynthia Hardenbrook
Lina Hardenburg
R. T. Harley
George B. Hawthorne
Clark Hay
William E. Hefley
Manfred Hein
John W. Horch
Richard Horner
Pei Hsia
James R. Hughes
Lynn Ihlenfeldt
S. C. Jain
Debra Jasinski
Shang-Sheng Jeng
J. O. Jenkins
Jia Yaoliang
Todd Jochim
David Johnson
Donna C. Johnson
Laurel V. Kaleda
Eiichi Kaneko
Myron S. Karasik
Peter Klopfenstein
Shaye Koenig
Richard Kuhn
Joan Kundig
Tom Kurihara
Renee Lamb
John B. Lane
Robert A. C. Lane
J. Dennis Lawrence
Cheryl Leventhal
F. C. Lim
Bertil Lindberg
Carlo Lipizzi
William M. Lively
Ben Livson
Dieter Look
Harold Mains
Kartik C. Majumdar
John H. Manley
Philip C. Marriott
Roger J. Martin
Tomoo Matsubara
Ann Miller
Edward Miller
Manijeh Moghis
Richard Morton
Gene T. Morun
Hironobu Nagano
Saied Najafi
Gerry Neidhart
Dennis E. Nickle
James O'Day
Tuncer I. Oren
Martin F. Owens
Thomas Parrish
Betty Paul
William Perry
Donald J. Pfeiffer
John G. Phippen
John Pope
Salim Ramji
Jean-Claude Rault
Meir Razy
Larry K. Reed
Donald J. Reifer
James R. Roberts
R. Waldo Roth
Rafael San Roman
Julio Sanz
Stephen R. Schach
Hans Schaefer
Benson H. Scheff
Max J. Schindler
Norman Schneidewind
Wolf A. Schnoege
David J. Schultz
Gregory D. Schumacher
Leonard Seagren
Carl S. Seddio
Tony Sgarlatti
Isaac Shadman
Lawrence H. Shafer
Robert Shillato
David M. Siefert
Harry M. Sneed
A. R. Sorkowitz
Kelley Stalder
David Szombatfaluy
William Theiford
Russell E. Theisen
Glendon R. Trebble
Leonard Tripp
Mark-Rene Uchida
Margaret Updike
Spyrpos Villios
Thomas E. Vollman
Dolores R. Wallace
John W. Walz
Andrew H. Weigel
Richard Werling
Paul Wolfgang
Paul R. Work
Natalie C. Yopconka
Peter F. Zoll

When the IEEE Standards Board approved this standard on September 26, 1991, it had the following membership:

Marco W. Migliaro, *Chairman* **Donald C. Loughry,** *Vice Chairman*
Andrew G. Salem, *Secretary*

Dennis Bodson
Paul L. Borrill
Clyde Camp
James M. Daly
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ingrid Fromm
Thomas L. Hannan
Donald N. Heirman
Kenneth D. Hendrix
John W. Horch
Ben C. Johnson
Ivor N. Knight
Joseph L. Koepfinger*
Irving Kolodny
Michael A. Lawler
John E. May, Jr.
Lawrence V. McCall
Donald T. Michael*
Stig L. Nilsson
John L. Rankine
Ronald H. Reimer
Gary S. Robinson
Terrance R. Whittemore

*Member Emeritus

Kristin M. Dittmann
IEEE Standards Project Editor

Acknowledgments

Participants in the working group were individually supported by their employers with travel expenses and working days. This support does not constitute or imply approval or endorsement of this standard. These organizations were:

Abbott Critical Care
ARINC, Inc.
Apollo Computer Inc.
AT&T Bell Laboratories
AT&T Technologies
Bell Canada
Bellcore
Bell Northern Research, Inc.
Boeing Computer Services
Burnett Associates
Computer Sciences Corp.
Digital Switch Corporation
E. I. Dupont de Nemours & Co.
E-Systems, Inc.
Eastman Kodak Co.
Hewlett Packard
Honeywell, Inc.
IBM
Institute for Defense Analyses
Jet Propulsion Laboratory
Litton Aero Products
Lockheed Aircraft Service Co.
Martin Marietta
MUMPS Development Committee
NCR Corp.
Naval Air Development Center
Northern Telecom Canada Ltd.
Northrop Electronics Div.
Northrop Aircraft Div.
Perkin-Elmer
Quality Assurance Institute
Singer Link
Tandem Telecommunications Systems, Inc.
Tektronix
Teledyne Brown Engineering
Teledyne Controls
Texas Instruments
The Horch Company
U. S. Air Force
U. S. Dept. of Transportation
Unisys
3M

The following organizations hosted working group meetings in their respective cities:

AT&T Bell Laboratories, Columbus, OH
Bellcore, Piscataway, NJ
Boeing Computer Services, Seattle, WA
Computer Sciences Corp., Silver Spring, MD
E-Systems, Inc., Salt Lake City, UT
Eastman Kodak Company, Rochester, NY
Hewlett Packard, San Jose, CA
Honeywell, Inc., Phoenix, AZ
IEEE Computer Society, Dallas, TX
Martin Marietta, Orlando, FL
NCR Corp, San Diego, CA
Northrop Electronics, Hawthorne, CA
Northern Telecom, Ottawa, Ont.
Tektronix, Beaverton, OR
Teledyne Brown Engineering, Huntsville, AL
Travelers Companies, Hartford, CT
Unisys, Eagan, MN

AT&T Bell Laboratories provided word processing support. Bellcore and Computer Sciences Corporation assisted with mailings. The X3K1 Logical Flow Project provided technical coordination.

Suggestions for the improvement of this standard will be welcome. They should be sent to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, P. O. Box 1331, Piscataway, NJ 08855-1331.

Contents

SECTION PAGE

SECTION PAGE

FIGURES PAGE

TABLES

APPENDIXES

APPENDIX TABLES

APPENDIX FIGURE

IEEE Standard for Developing Software Life Cycle Processes

1. Introduction

1.1 Scope. This standard provides the set of Activities that constitute the Processes that are mandatory for the development and maintenance of software, whether stand-alone or part of a system. (Non-software Activities, such as hardware development and purchasing, are outside of the scope of this standard.) This standard also provides associated Input and Output Information.

For convenience, Activities are listed and described under specific Processes. In practice, the Activities may be performed by persons whose organizational titles or job descriptions do not clearly convey that a Process is part of their job. The Process under which an Activity is listed in this standard may be transparent in practice.

This standard does not prescribe a specific software life cycle model (SLCM). Each using organization must map the activities specified in the standard into its own software life cycle (SLC). If an organization has not yet defined an SLC, it will be necessary for them to select or define one before attempting to follow this standard. Further, this standard does not presume the use of any specific software development methodology nor the creation of specific documents.

For software already developed, it is recommended that these requirements, or a subset thereof, be applied. The existence of this standard should not be construed to prohibit the imposition of additional or more stringent requirements where the need exists, e.g., critical software.

Compliance with this standard is defined in 1.5.1.

1.2 References. No other publications are required for use of this standard. However, a list of other IEEE standards, which may be consulted for additional guidance, is given in the Bibliography in Section 8. Although this standard does not require adherence to any other IEEE standard, knowledge of principles and concepts described in the standards listed in the Bibliography would be helpful.

1.3 Definitions and Acronyms

1.3.1 Definitions. The definitions listed here establish meanings within the context of this standard. Definitions of other terms used in this document can be found in IEEE Std 610.12-1990 [1].[1]

Activity. A constituent task of a Process. *See:* **task.**

analysis. Examination for the purpose of understanding.

anomaly. Any deviation from requirements, expected or desired behavior, or performance of the software.

contractual requirements. Customer-imposed performance, logistics, and other requirements and commitments governing the scope of software development, delivery, or support.

customer. The person, or persons, who pay for the product and usually (but not necessarily) decide the requirements. In the context of this document the customer and the supplier may be members of the same organization. [5]

data base. A collection of data fundamental to a system. [22]

[1] The numbers in brackets correspond to those of the bibliographic references listed in Section 8.

evaluation. Determination of fitness for use.

external. An Input Information source or Output Information destination that is outside the control of this standard, and therefore may or may not exist.

function. A specific purpose of an entity or its characteristic action. [22]

installation. The period of time in the software life cycle during which a software product is integrated into its operational environment and tested in this environment to ensure that it performs as required. [1]

Mapping. Establishing a chronological relationship of the Activities in this standard according to a selected SLCM.

methodology. A body of methods, rules, and postulates employed by a discipline.

owner. A single point of contact, identified by organization position.

problem. The inability of a system or component to perform its required functions within specified performance requirements. [1]

Process. A function that must be performed in the software life cycle. A Process is composed of Activities.

product. Any output of the software development Activities; e.g., document, code, model.

quality management. That aspect of the overall management function that determines and implements the quality policy. (ISO 9000)

quality policy. The overall quality intentions and direction of an organization as regards quality, as formally expressed by top management. (ISO 9000)

revision. A controlled item with the same functional capabilities as the original plus changes, error resolution, or enhancements.

software life cycle (SLC). A project-specific, sequenced mapping of Activities.

software life cycle model (SLCM). The skeleton framework selected by each using organization on which to map the Activities of this standard to produce the software life cycle.

software quality management. That aspect of the overall software management function that determines and implements the software quality policy.

software quality policy. The overall quality intentions and direction of an organization as regards software quality, as expressed by top management.

software system. Software that is the subject of a single software project.

supplier. The person, or persons, who produce a product for a customer. In the context of this document, the customer and the supplier may be members of the same organization. [5]

task. The smallest unit of work subject to management accountability. A task is a well-defined work assignment for one or more project members. Related tasks are usually grouped to form Activities. [17]

unit. A logically separable part of a program. [1]

user. The person, or persons, who operate or interact directly with the system. [5]

1.3.2 Acronyms. The following acronyms appear within the text of this standard:

CASE Computer-Aided Software Engineering
I/O Input/Output
PR&RP Problem Report and Resolution Planned Information
SCMP Software Configuration Management Planned Information
SDD Software Design Description
SLC Software Life Cycle
SLCM Software Life Cycle Model
SPMP Software Project Management Planned Information
SQA Software Quality Assurance
SRS Software Requirements
SVVP Software Verification and Validation Planned Information

1.4 Organization of This Document. The organization of this standard provides a logical approach to the development, operation, and maintenance of software. The detailed requirements of this document are organized into 17 Processes, which are comprised of a total of 65 Activities. The Processes and their Activities are described in six major sections. Table 1 depicts this organization.

Table 1
Organization of the Standard

Section	Title	Processes
2	Software Life Cycle Model Process	Software Life Cycle Model
3	Project Management Processes	Project Initiation Project Monitoring and Control Software Quality Management
4	Pre-Development Processes	Concept Exploration System Allocation
5	Development Processes	Requirements Design Implementation
6	Post-Development Processes	Installation Operation and Support Maintenance Retirement
7	Integral Processes	Verification and Validation Software Configuration Management Documentation Development Training

All of a Process' required actions are specified in its constituent Activities. Each Activity discussion has three parts:

(1) Input Information, which lists information that is to be used by the Activity, and its source.
(2) Description, which details the actions to be performed.
(3) Output Information, which lists the information that is generated by the Activity, and its destination.

Where information flows between Activities, it can be traced from its original Activity to the receiving Activity through the Input and Output Information tables.

As mentioned above, all Processes are mandatory. Activities, however, are categorized as mandatory[2] or "If Applicable."[3] "If Applicable" Activities are marked "If Applicable" in the Activity title. All other Activities are mandatory. Each "If Applicable" Activity contains an explanation of the cases to which it will apply (e.g., 5.2.4, Design Data Base, applies when the software product contains a data base).

1.5 Use of This Standard. To facilitate the understanding and use of a standard of this magnitude, this section provides additional information.

1.5.1 Applicability. This standard applies to software development and maintenance projects.

This standard can be applied to commercial, scientific, and military software. Applicability is not restricted by size, complexity, or criticality of the software. This standard considers both the software and its context.

It is recognized that a project may be too small, in terms of schedule, budget, risk, nature, or use of software to be developed, used, and maintained, to warrant total application of the standard. In such cases, selected Activities may be applied even though compliance with this standard may not be claimed. This may also apply when only purchased software is involved.

A large project may be subdivided into smaller manageable projects, and this standard applied to each of the smaller projects and then to the whole. Similarly, some projects may be of long duration, and may be delivered in multiple versions or releases; it may be helpful in some cases to treat the development of each successive version as a separate project with its own life cycle.

1.5.2 Compliance. Compliance with this standard is defined as the performance of all mandatory Activities. Some mandatory Activities may occur in different instances (e.g., performing tests at various levels); compliance with this standard means the complete performance of each instance of the mandatory Activity. The standard does not specify instances of any Activity.

The performance of an Activity or an instance thereof is complete when all Input Information has been processed, and all Output Information has been generated. This may require several iterations of an Activity or instance.

All Input and Output Information are not required for a given occurrence of an iterative Activity. The presence of sufficient Input Information to permit processing by the Activity to begin constitutes the entry criterion, and the creation of any Output Information is a sufficient exit criterion.

This standard does not impose the order in which Activities must be performed. However, an order must be established by executing the Activities defined in Section 2 and the Activity in 3.1.3.

In some cases, certain Input and Output Information may not be required for completion of an Activity. These are indicated as "External" to the SLC and may not exist. To the extent that they exist, they must be processed by affected Activities.

This standard prescribes the *processes* of the software life cycle, not the *products* of that life cycle. Therefore, the standard does not require the completion of specific documents. The information generated by Activities, listed in the Output Information tables, may be collected into documents in any manner consistent with the selected SLCM.

In the event that this standard is contractually imposed, and one or more subcontractors are involved in the project, it is recommended that the requirements of this standard be imposed on the subcontractors.

[2]The term "mandatory" as used in this standard is synonymous with the term "essential."
[3]The term "if applicable" as used in this standard is synonymous with the terms "conditional" and "optional."

1.5.3 Intended Audience. This standard cannot be implemented by a single functional group within a development organization. Mapping this standard's Activities into an organization's SLCM, and coordinating these Activities with existing development and support methodologies and standards, require specific expertise and authority within the organization.

After mapping is complete, the Activities in Sections 3–7 are ready for execution. These Activity descriptions are directed to the functional specialist most likely to be performing them (e.g., the Requirements and Analysis Activities assume a basic familiarity with analysis techniques).

1.5.4 Process and Activity Relationships

1.5.4.1 Project Management Processes. There are three Processes in this section: the Project Initiation Process, the Project Monitoring and Control Process, and the Software Quality Management Process. The Project Initiation Process consists of those Activities that create and maintain the project framework. The Activities within the Project Monitoring and Control Process and the Software Quality Management Process are performed throughout the life of the project to ensure the appropriate level of project management and compliance with the mandated Activities.

1.5.4.2 Development-Oriented Processes. These are the Processes that must be performed before, during, and after the development of the software. The Pre-Development Processes are Concept Exploration and System Allocation. Development Processes include Requirements, Design, and Implementation. Finally, the Post-Development Processes include Installation, Operation and Support, Maintenance, and Retirement.

1.5.4.3 Integral Processes. This section includes those Processes that are necessary to ensure the successful completion of a project, but are not Development Processes. The Integral Processes are Verification and Validation, Software Configuration Management, Documentation Development, and Training. All of these Processes contain two types of Activities:

(1) Those that are performed discretely and are therefore mapped into an SLCM.
(2) Those that are performed in the course of completing another Activity. These are invoked Activities and will not be mapped into the SLCM in every instance.

Many Activities invoke, or call like a subroutine, appropriate Integral Process(es). This is an intuitive method of getting a task, such as the evaluation or production of a document, done without specifying an exact control flow within this standard.

To track the flow of a product into, through, and back from an Integral Process, there are generic sources and destinations listed in the Input and Output Information tables called "Creating Process." Within the Integral Process that is invoked, the Activity that first receives the product has a generic Input Information whose source is Creating Process. This product passes through one or more Activities of that Integral Process, then is returned to the invoking Activity through an Activity's Output Information, whose destination is Creating Process. An example of this flow is shown in Fig 1, which illustrates how one Activity within the Design Process invokes the Integral Processes. This figure will be easier to follow if it is compared directly with 5.2.7, "Perform Detailed Design." The order of Invoked Processes in the figure differs from that in 5.2.7.

The invoked Process is specified in the text by the name of the Process and the number of the first Activity to be performed within that Process [e.g., Verification and Validation (7.1.4)].

1.5.4.4 Use of I/O Tables. The Input Information to and Output Information from each Activity are listed in tables that accompany the Activity descriptions. As a convention of this document, Input and Output Information names are always capitalized in the text.

The I/O tables show the flow of Information through the Activities. The pertinent Information is listed in the left-hand column. The source or destination of the Information (both Process and Activity) is shown in the right-hand columns. The Information names were chosen, where possible, to suggest the titles of documents commonly used in the software industry. The Information names also frequently relate to the documents described in the standards listed in Section 8. Some of these Information items are represented in the text by acronyms that were deliberately chosen to suggest the commonly used names of the corresponding documents.

External Input Information sources and External Output Information destinations are outside the scope of this standard. External Input Information may or may not exist, and if it does not exist, it is not required. When an External Input does exist and is therefore used, it is presumed to include any associated documentation. External output destinations also may or may not exist. External sources and destinations are not necessarily Processes, and no corresponding Activities are shown in the I/O tables.

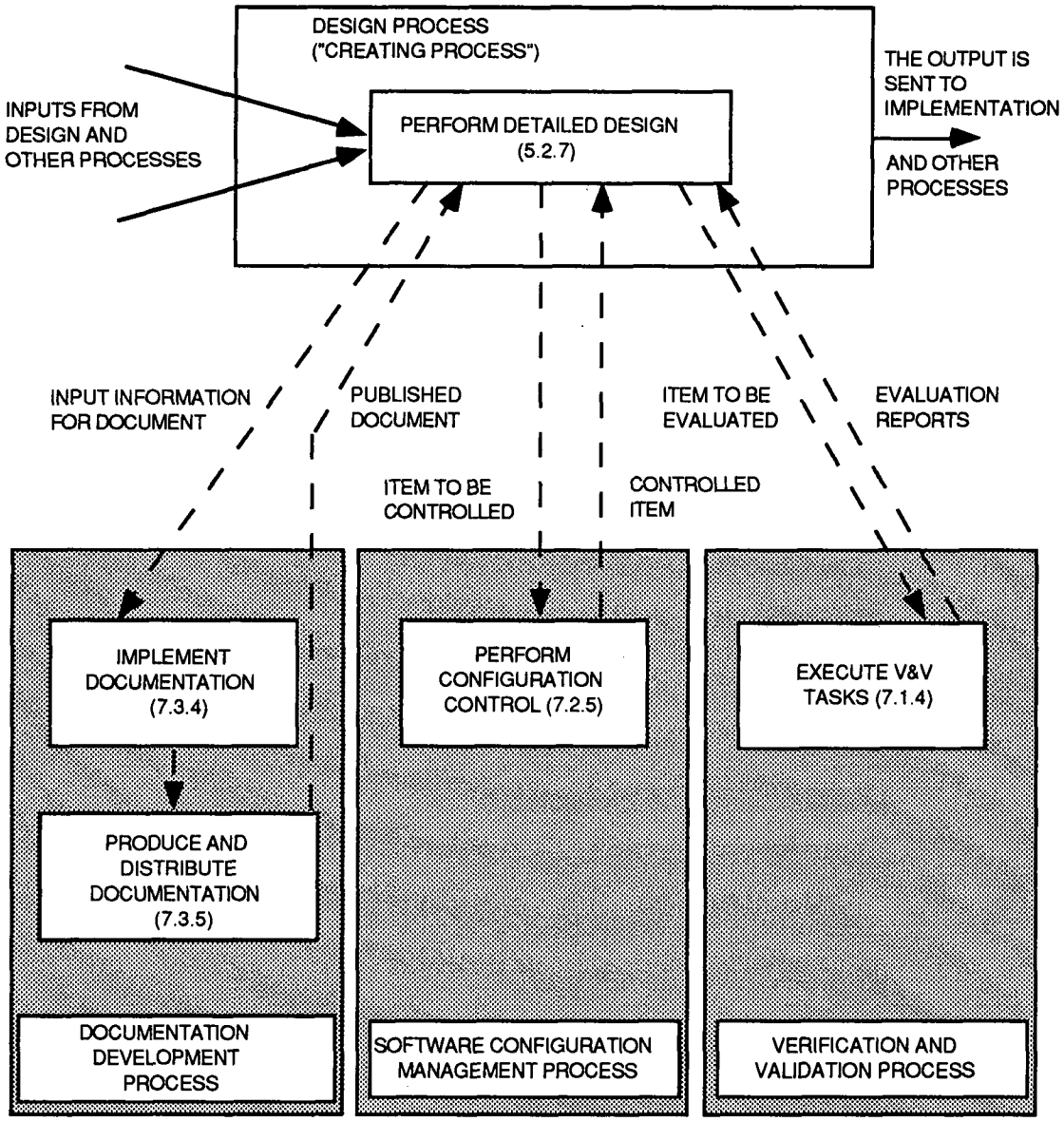


Fig 1
Example of Invoked Processes

In most cases, the Input Information and Output Information columns of the tables designate the specific information that enters or exits the Activity. However, since many Activities have Output Information whose destination is Retain Records (3.2.6), the various Input Information to Retain Records is collected under the term "Original Records." The corresponding Process and Activity columns refer simply to Originating Process and Originating Activity. Figure 2 depicts the conceptual flow of Input Information and Output Information into and out from an Activity, respectively.

1.5.5 Getting Started. Before beginning a project that will use this standard, the Activities need to be reviewed for applicability to a specific project and organized into a time sequence appropriate to that project. To perform that time sequencing, an SLCM must be chosen or developed and Activities mapped into the SLCM. This mapping is discussed in Section 2 and the mapping Activity in 3.1.3. Examples are given in Appendix A.

This mapping produces a temporal "road map" called the Software Life Cycle (SLC) used to follow this standard throughout the project. The mapped Activities must be initiated in their designated sequence.

1.5.6 Additional Considerations

1.5.6.1 Organizational Independence. This standard does not presume or dictate an organizational structure for a software project. Therefore, it is neither implied nor required that Activities within a Process be performed by the same organizational entity, nor that an organizational entity's involvement be concentrated in only one Process. To ensure that all Activities are assigned to an appropriate organizational entity, the concept of Activity Ownership is described in the Activity in 3.1.3.

1.5.6.2 Combining Documents. The Information developed in this standard, as shown in the Output Information tables, may carry generic names similar to those used in other IEEE standards. This does not imply that the format and content specified in other IEEE standards must be followed, nor that this information must be packaged into documents in any particular manner.

Combination of documents into a single document is acceptable as long as understanding is not compromised.

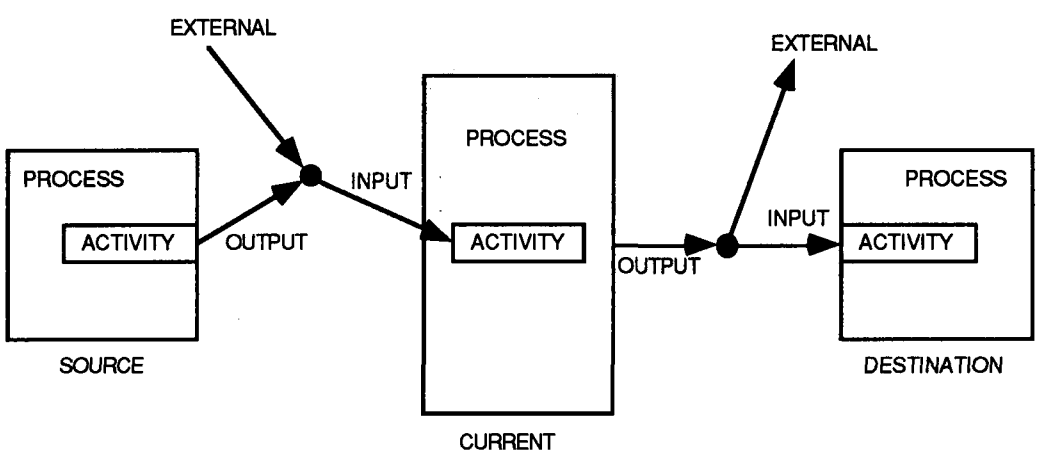


Fig 2
Information Flow

2. Software Life Cycle Model Process

2.1 Overview. Many variables affect an organization's selection of a software life cycle model (SLCM). While this standard neither dictates nor defines a specific software life cycle (SLC) or its underlying methodologies, it does require that an SLCM be chosen and used.

This Process provides the Activities required to identify candidate SLCMs and select the SLCM to be used by other Activities in the standard.

This standard includes the specification of the non-time-ordered set of "Mandatory" Activities that must be incorporated into an SLCM. An SLCM (e.g., Rapid Prototyping) defines a specific approach to producing software. It specifies a time-ordered set of Activities (including all of the "Mandatory" Activities identified in this standard), which is to be used as the basis for mapping the Activities of this standard. An SLCM may also propose standards for the performance of the Activities or the deliverables produced during the project.

The SLC (defined for a project by the Activity in 3.1.3) is the time-ordered set of Activities or instances of Activities to be performed. This set is to be mapped into a selected SLCM. The SLC also identifies specific responsibilities for each Activity. While the same SLCM may be valid for several projects, each project must define its own SLC.

Once an SLCM is selected, there are two additional required actions:

(1) Mapping the Activities described in this standard into the chosen life cycle (3.1.3).
(2) Identifying and documenting the standards and controls that govern the SLC (3.1.5).

Figure 3 illustrates this progression from this standard to a project-specific SLC.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

2.2 Activities List

(1) Identify Candidate Software Life Cycle Models
(2) Select Project Model

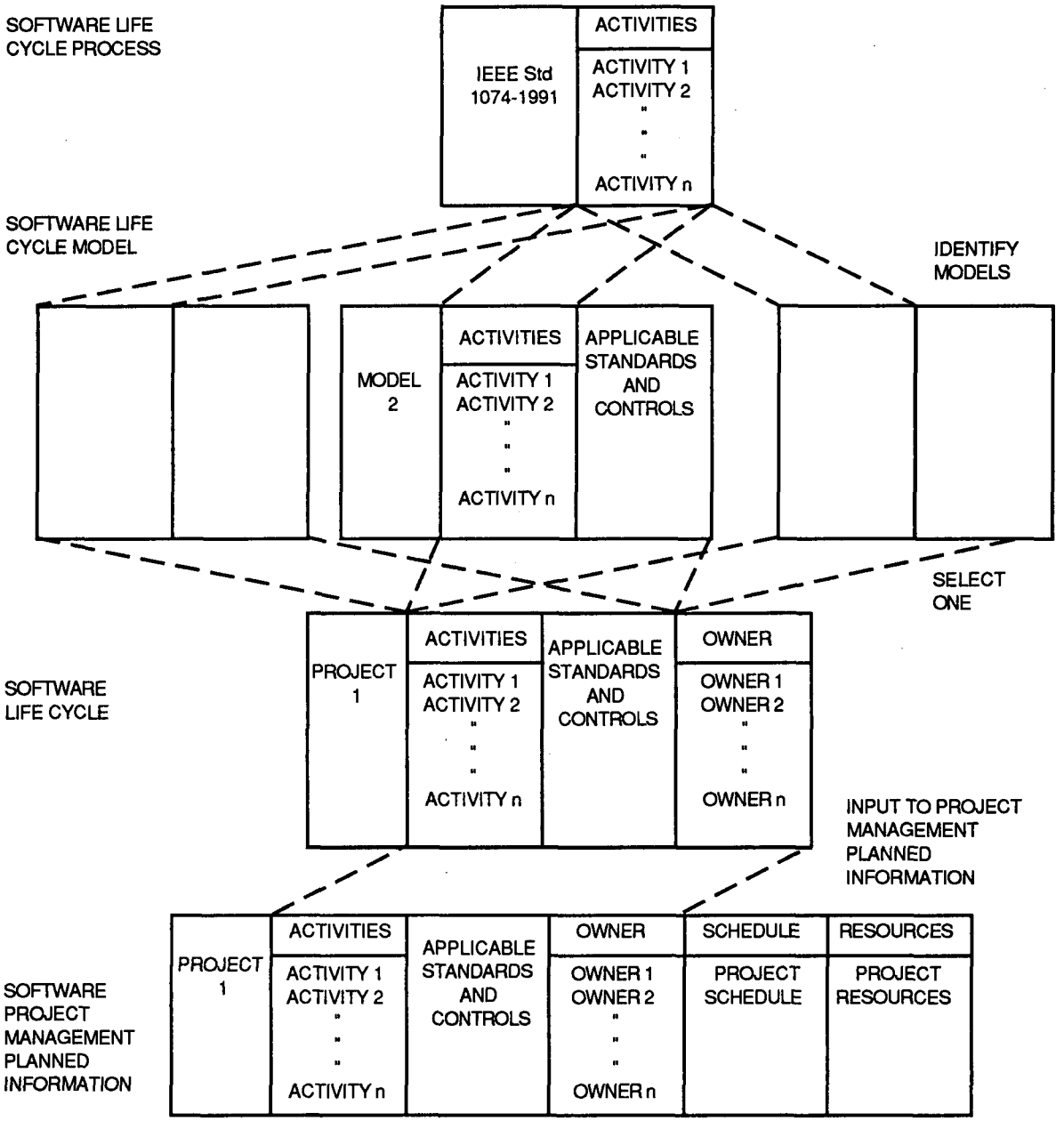


Fig 3
Software Life Cycle Relationships

2.3 Identify Candidate Software Life Cycle Models

2.3.1 Input Information

Input Information	Source	
	Process	Activity
Available Software Life Cycle Models)	External	
Constraints	External	

2.3.2 Description. In this Activity, the set of Available SLCMs and applicable Constraints shall be considered and Candidate SLCMs identified. A new model may be constructed by combining elements of other SLCMs.

Maintenance is an iteration of the Software Life Cycle, and the SLCM must support this iteration.

2.3.3 Output Information

Output Information	Destination	
	Process	Activity
Candidate Software Life Cycle Model(s)	Software Life Cycle Model	Select Project Model (2.4)

2.4 Select Project Model

2.4.1 Input Information

Input Information	Source	
	Process	Activity
Historical Project Records	External	
Constraints	External	
Candidate Software Life Cycle Model(s)	Software Life Cycle Model	Identify Candidate Software Life Cycle Models (2.3)

2.4.2 Description. In this Activity, one of the candidate SLCMs from 2.3.2 is selected for use.

Based on the type of product (interactive, batch, transaction processing, etc.), Constraints, and Historical Project Records, an SLCM analysis shall be conducted, and a decision made as to which model will best support the management of the project.

It is possible for an organization to have more than one SLCM, but only one model may be selected for a project. It is not necessary to have a single, organization-wide SLCM.

The SLCM shall provide the necessary framework for software projects to map the Activities to produce the SLC (as shown in Fig 3). The mapping effort is specified in the Project Initiation Process, Map Activities to Software Life Cycle Model (3.1.3).

2.4.3 Output Information

Output Information	Destination	
	Process	Activity
Selected Software Life Cycle Model	Project Initiation	Map Activities to Software Life Cycle Model (3.1.3)

3. Project Management Processes

These are the Processes that initiate, monitor, and control software projects throughout the software life cycle (SLC).

3.1 Project Initiation Process

3.1.1 Overview. This Process contains those Activities that create the framework for the project. During this Process, the SLC is created for this project, and plans for managing the project are established. Standards, methodologies, and tools needed to manage and execute the project are identified and a plan prepared for their timely implementation.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

3.1.2 Activities List

(1) Map Activities to Software Life Cycle Model
(2) Allocate Project Resources
(3) Establish Project Environment
(4) Plan Project Management

3.1.3 Map Activities to Software Life Cycle Model

3.1.3.1 Input Information

Input Information	Source	
	Process	Activity
Contractual Requirements	External	
Selected Software Life Cycle Model	Software Life Cycle Model	Select Project Model (2.4)
Statement of Need	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)

3.1.3.2 Description. The Activities identified in this standard shall be mapped into the selected SLC Model (SLCM). Mapping involves establishing the chronological relationship of the Activities in this standard according to the selected SLCM. It may be necessary to use the Contractual Requirements and the Statement of Need to accomplish this mapping. Appendix A provides several examples of such mappings. Appendix B is a template for adding additional project-specific information, such as document titles and applicable standards, to the mapped Activities.

The use of certain software development methods defines the execution of some Activities to be automated. Compliance with this standard must be demonstrated by mapping those automated Activities into the appropriate points within the SLCM.

Each Activity shall be assigned a single "owner." An owner is a single point of contact, and is identified by organizational position. Ownership is assumed by the person currently filling that position. Each owner has the responsibility and authority to control and complete the Activity within the planned schedule and budget. In addition, each owner is accountable for the quality of the Activity outputs. If Activities are to be performed by multiple organizations, the owning organization and position of the owner shall be identified. In the case of multiple instances of an Activity, an owner for each instance shall be identified.

The resulting map of the Activities to be performed, with their corresponding owners, is the SLC for this project. All "If Applicable" Activities that do not apply to this project shall be identified and explained in the List of Activities Not Used.

3.1.3.3 Output Information

Output Information	Destination	
	Process	Activity
Software Life Cycle	Project Initiation	Allocate Project Resources (3.1.4)
		Establish Project Environment (3.1.5)
		Plan Project Management (3.1.6)
List of Activities Not Used	Project Monitoring and Control	Retain Records (3.2.6)

3.1.4 Allocate Project Resources

3.1.4.1 Input Information

Input Information	Source	
	Process	Activity
Historical Project Records	External	
Resources	External	
Statement of Need	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)
Software Life Cycle	Project Initiation	Map Activities to Software Life Cycle Model (3.1.3)
System Functional Software Requirements	System Allocation	Decompose System Requirements (4.2.5)

3.1.4.2 Description. Resource Allocations shall be identified at the Software Life Cycle's Activity level. Resources to be allocated include personnel, equipment, space, etc. Available Historical Project Records and the Statement of Need may provide valuable insight into Resource Allocation.

3.1.4.3 Output Information

Output Information	Destination	
	Process	Activity
Resource Allocations	Project Initiation	Establish Project Environment (3.1.5)
		Plan Project Management (3.1.6)
	Project Management and Control	Analyze Risks (3.2.3)

3.1.5 Establish Project Environment

3.1.5.1 Input Information

Input Information	Source	
	Process	Activity
Methodologies	External	
Standards	External	
Tools	External	
Software Library	External	
Purchased Software	External	
Contractual Requirements	External	
Analysis of Risks	Project Monitoring and Control	Analyze Risks (3.2.3)
Software Life Cycle	Project Initiation	Map Activities to Software Life Cycle Model (3.1.3)
Defined Metrics	Software Quality Management	Define Metrics (3.3.4)
Collection and Analysis Methods	Software Quality Management	Define Metrics (3.3.4)
Resource Allocations	Project Initiation	Allocate Project Resources (3.1.4)
Statement of Need	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)

3.1.5.2 Description. The needs of the project for procedural and technological Tools, Methodologies, and Standards shall be defined. Approaches to these needs shall be identified and evaluated. These approaches could include automated and nonautomated tools, modeling and prototyping methodologies, environment simulators, test beds, and software libraries. Selection criteria for Tools and Methodologies should include resource, schedule, safety, and security considerations, and the project requirements defined in the Statement of Need and Analysis of Risks. The project standards shall include requirements, design, coding, test, and documentation standards.

After evaluating the approaches, a set of Tools, Methodologies, Standards, and reusable or Purchased Software shall be selected to provide the Project Environment, considering the Input Information.

The selected tools shall be acquired or developed and installed for use in project Activities. The owner of this Activity shall ensure that applicable personnel are familiar with the Tools, Methodologies, and Standards selected for the project.

For assistance in identifying applicable standards, [9] should be consulted.

Prior to distribution of the Project Environment, the Training Process (7.4.4) shall be invoked.

3.1.5.3 Output Information

Output Information	Destination	
	Process	Activity
Project Environment	Project Initiation	Plan Project Management (3.1.6)

3.1.6 Plan Project Management

3.1.6.1 Input Information

Input Information	Source	
	Process	Activity
Contractual Requirements	External	
Software Life Cycle	Project Initiation	Map Activities to Software Life Cycle Model (3.1.3)
Resource Allocations	Project Initiation	Allocate Project Resources (3.1.4)
Project Environment	Project Initiation	Establish Project Environment (3.1.5)
Contingency Planned Information	Project Monitoring and Control	Perform Contingency Planning (3.2.4)
Project Management Reported Information	Project Monitoring and Control	Manage the Project (3.2.5)
Preliminary Statement of Need	Concept Exploration	Identify Ideas or Needs (4.1.3)
Recommendations	Concept Exploration	Conduct Feasibility Studies (4.1.5)
Statement of Need	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)

3.1.6.2 Description. Project management planning requires collection and synthesis of a great deal of information into a coherent and organized Software Project Management Planned Information (SPMP) based on the SLC. This Activity shall initially define and subsequently update the SPMP using the Input Information. This Activity shall detail the project organization and assign responsibilities. Standards, methodologies, and tools for configuration management, quality assurance, verification and validation, training, documentation, and development shall be specified. This Activity shall apportion the project budget and staffing, and define schedules, using the applicable Input Information. It also shall define procedures for scheduling, tracking, and reporting, and shall address considerations such as regulatory approvals, required certifications, user involvement, subcontracting, and security.

This Activity shall include planning for support, problem reporting, and retirement. Support planning shall include methods for supporting the software in the operational environment. Problem Reporting and Resolution Planning Information shall include, at a minimum, defining a method for logging, routing, and handling problem reports; categories of severity; and the method for verifying problem resolution. Retirement Planned Information shall address issues such as probable retirement date, archiving, replacement, and residual support issues.

As new or revised Input Information is received in this Activity, project plans shall be updated and further project planning shall be based upon these updated plans.

Additional guidance for SPMPs can be found in [17].

Prior to distribution of the SPMP, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

3.1.6.3 Output Information

Output Information	Destination	
	Process	Activity
Problem Reporting and Resolution Planned Information	Project Monitoring and Control	Manage the Project (3.2.5)
		Analyze Risks (3.2.3)
		Implement Problem Reporting Method (3.2.7)
Retirement Planned Information	Project Monitoring and Control	Manage the Project (3.2.5)
	Retirement	Notify User (6.4.3)
		Conduct Parallel Operations (If Applicable) (6.4.4)
		Retire System (6.4.5)
Software Project Management Planned Information	Most Processes	Most Activities
Support Planned Information	Project Monitoring and Control	Analyze Risks (3.2.3)
		Manage the Project (3.2.5)
	Operation and Support	Maintain Support Request Log (6.2.5)
		Operate the System (6.2.3)
		Provide Technical Assistance and Consulting (6.2,4)

3.2 Project Monitoring and Control Process

3.2.1 Overview. Monitoring and control is an iterative Process of tracking, reporting, and managing costs, schedules, problems, and performance of a project throughout its life cycle. The progress of a project is reviewed and measured against project milestones established in the Software Project Management Planned Information (SPMP).

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

3.2.2 Activities List

(1) Analyze Risks
(2) Perform Contingency Planning
(3) Manage the Project
(4) Retain Records
(5) Implement Problem Reporting Method

3.2.3 Analyze Risks

3.2.3.1 Input Information

Input Information	Source	
	Process	Activity
Procurement/Lease Data	External	
System Constraints	External	
Historical Project Records	External	
Support Planned Information	Project Initiation	Plan Project Management (3.1.6)
Resource Allocations	Project Initiation	Allocate Project Resources (3.1.4)
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Problem Reporting and Resolution Planned Information	Project Initiation	Plan Project Management (3.1.6)
Transition Impact Statement (If Applicable)	Concept Exploration	Plan System Transition (If Applicable) (4.1.6)
Statement of Need	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)
Software Interface Requirements	Requirements	Define Interface Requirements (5.1.4)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
Software Design Description	Design	Perform Detailed Design (5.2.7)
Integration Planned Information	Implementation	Plan Integration (5.3.7)
Analysis Reported Information	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
Test Planned Information(s)	Verification and Validation	Plan Testing (7.1.6)
Test Summary Reported Information	Verification and Validation	Execute the Tests (7.1.8)

3.2.3.2 Description. Because risk management often involves trade-offs between many factors, risk analysis is an iterative Activity performed throughout a project's life. This analysis

shall consider project risks, including technical, economic, operational support, and schedule risks.

Factors that may impair, prevent, or require technical trade-offs for accomplishing the technical objectives of the project or product shall be identified and analyzed. Technical factors may include such items as real-time performance, safety considerations, security considerations, implementation considerations, testability, and maintainability. Analytical approaches for technical risk assessment may include static and dynamic modeling and simulation, prototyping, independent reviews, and audits.

Cost, resource factors, earnings, liabilities, or other economic measures involved in the project shall be identified and analyzed. The objective of this analysis is to identify potential economic opportunities, losses, and trade-offs. Analytical approaches for economic risk assessment may include financial analysis, such as return on investment and possible incentive and penalty contract clauses.

Operational and support risk analysis shall determine the probability that the delivered software will meet the user's requirements. Operational and support requirements such as interoperability, security, performance, installability, and maintainability shall be considered. Both completeness of, and conformance to, these requirements shall be analyzed.

Cost, resource, technical, and other requirements shall be evaluated for their impact on project schedule. This analysis should consider project interdependence and the effect of schedule adjustments. Analytical approaches for schedule risk assessment may include critical path analysis and resource leveling techniques.

3.2.3.3 Output Information

Output Information	Destination	
	Process	Activity
Analysis of Risks	Project Initiation	Establish Project Environment (3.1.5)
	Project Monitoring and Control	Perform Contingency Planning (3.2.4)
	Requirements	Define and Develop Software Requirements (5.1.3)
	Verification and Validation	Plan Verification and Validation (7.1.3)

3.2.4 Perform Contingency Planning

3.2.4.1 Input Information

Input Information	Source	
	Process	Activity
Analysis of Risks	Project Monitoring and Control	Analyze Risks (3.2.3)
Analysis Reported Information	Verification and Validation	Collect and Analyze Metric Data (7.1.5)

3.2.4.2 Description. This Activity shall define alternative actions in the event that a given risk materializes, using the Input Information. Contingency Planned Information shall include resource planning and the establishment of trigger conditions that would invoke a contingency action. Contingency actions may include consideration of revised requirements, delay, or cancellation of the project.

3.2.4.3 Output Information

Output Information	Destination	
	Process	Activity
Contingency Planned Information	Project Initiation	Plan Project Management (3.1.6)
	Project Monitoring and Control	Manage the Project (3.2.5)

3.2.5 Manage the Project

3.2.5.1 Input Information

Input Information	Source	
	Process	Activity
Problem Reporting and Resolution Planned Information	Project Initiation	Plan Project Management (3.1.6)
Retirement Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Support Planned Information	Project Initiation	Plan Project Management (3.1.6)
Contingency Planned Information	Project Monitoring and Control	Perform Contingency Planning (3.2.4)
Software Quality Management Planned Information	Software Quality Management	Plan Software Quality Management (3.3.3)
Quality Improvement Recommendations	Software Quality Management	Identify Quality Improvement Needs (3.3.6)
Integration Planned Information	Implementation	Plan Integration (5.3.7)
Installation Reported Information	Installation	Install Software (6.1.5)
Evaluation Reported Information	Verification and Validation	Execute Verification and Validation Tasks (7.1.4)
Analysis Reported Information	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
Test Planned Information(s)	Verification and Validation	Plan Testing (7.1.6)
Test Summary Reported Information	Verification and Validation	Execute the Tests (7.1.8)
Status Reported Information	Software Configuration Management	Perform Status Accounting (7.2.6)
Feedback Data	Operation and Support	Operate the System (6.2.3)

3.2.5.2 Description. Throughout the life cycle, the progress of the project shall be reviewed and measured against the established milestones and budget in the plan(s) (i.e., predicted and planned progress versus actual progress, and budgeted versus actual expenditures). Project tracking and reporting includes analyzing the Input Information, collecting other pertinent data, and monitoring project Activities. Anomalies may result. Risk management procedures must be implemented to control risk.

This Activity also encompasses the day-to-day management of the project needed to ensure successful project completion. Information collected within this Activity is used to improve the performance of the project.

Prior to distributing the Project Management Reported Information, the Verification and Validation Process (7.1.4) should be invoked.

3.2.5.3 Output Information

Output Information	Destination	
	Process	Activity
Project Management Reported Information	Project Initiation	Plan Project Management (3.1.6)
	Project Monitoring and Control	Retain Records (3.2.6)
	External	
Anomalies	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)

3.2.6 Retain Records

3.2.6.1 Input Information

Input Information	Source	
	Process	Activity
Documentation Retention Standards	External	
Original Records	Originating Process	Originating Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Configuration Management Planned Information	Software Configuration Management	Plan Configuration Management (7.2.3)
Documentation Planned Information	Documentation Development	Plan Documentation (7.3.3)
Published Document	Documentation Development	Produce and Distribute Documentation (7.3.5)

3.2.6.2 Description. This Activity accepts the original project documentation and records from each originating Process. The records shall be retained in accordance with the SPMP, Software Configuration Management Planned Information, and any external document retention standards. Input Information documentation becomes part of the Historical Project Records of the organization. Uses for these records may include project audits, future project planning, and corporate accounting.

3.2.6.3 Output Information

Output Information	Destination	
	Process	Activity
Historical Project Records	External	

3.2.7 Implement Problem Reporting Method

3.2.7.1 Input Information

Input Information	Source	
	Process	Activity
Anomalies	External	
	Creating Process	
Controlled Item	Software Configuration Management	Perform Configuration Control (7.2.5)
Problem Reporting and Resolution Planned Information	Project Initiation	Plan Project Management (3.1.6)

3.2.7.2 Description. This Activity accepts Anomalies from any source and prepares a problem report. The problem report shall contain information as specified in the Problem Reporting and Resolution Planned Information (PR&RP). Possible problem solutions may be suggested by the problem reporter. Problems may be resolved through corrections or enhancements (as defined in the PR&RP). Corrections are documented in the Correction Problem Reported Information for further consideration. Enhancements may be documented in the Enhancement Problem Reported Information and are possible candidates for new projects. A Report Log shall be maintained to assure that all problems are tracked until they are resolved and the resolution has been approved.

This Activity shall also analyze the problem including the Controlled Item, the problem report, and the Report Log to make the following determinations:

(1) What the anomalies are.
(2) Source and cause of product or process problem.
(3) Product(s) or process(es) presumed to contain the error, including documentation.
(4) Problem severity.
(5) Course of corrective action.

Problem reports that originate from an Activity not included in this standard are noted as resolved within this Activity and forwarded for appropriate action to the responsible authority.

This Activity shall monitor the problem correction efforts performed by the responsible Process, shall determine (according to the PR&RD) that the implementation of the solution by the responsible Process has been completed, and shall then record the resolution of the problem in the Resolved Problem Reported Information. The Resolved Problem Reported Information shall be distributed as specified in the Problem Reporting and Resolution Planned Information.

Further information related to this Activity may be found in [15].

The Resolved Problem Reported Information should be made available to the Process or external source that reported the problem.

Prior to distribution of a Problem Reported Information or the Report Log, the Software Configuration Management Process (7.2.5) should be invoked.

3.2.7.3 Output Information

Output Information	Destination	
	Process	Activity
Resolved Problem Reported Information	External	
	Creating Process	
	Software Quality Management	Manage Software Quality (3.3.5)
	Verification and Validation	Execute Verification and Validation Tasks (7.1.4)
		Collect and Analyze Metric Data (7.1.5)
Report Log	Software Quality Management	Manage Software Quality (3.3.5)
	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
Enhancement Problem Reported Information	Concept Exploration	Identify Ideas or Needs (4.1.3)
	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
Correction Problem Reported Information	Maintenance	Reapply Software Life Cycle (6.3.3)
	Verification and Validation	Collect and Analyze Metric Data (7.1.5)

3.3 Software Quality Management Process

3.3.1 Overview. An important role of Software Quality Management is to address the planning and administration of the Software Quality Assurance (SQA) program. It further addresses such concerns as client satisfaction (which transcends adherence only to established technical requirements), and internal quality improvement programs. The responsibilities, functions, obligations, and duties of an SQA program are properly a constituent part of all Activities in the Software Life Cycle, and thus are interspersed into each Activity as appropriate. Software Quality Management is the methodology used in this standard for tying the SQA responsibilities together with other Quality concerns. The Activities of this Process span the entire Software Life Cycle (SLC).

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

3.3.2 Activities List

(1) Plan Software Quality Management
(2) Define Metrics
(3) Manage Software Quality
(4) Identify Quality Improvement Needs

3.3.3 Plan Software Quality Management

3.3.3.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Defined Metrics	Software Quality Management	Define Metrics (3.3.4)
Collection and Analysis Methods	Software Quality Management	Define Metrics (3.3.4)

3.3.3.2 Description. A Software Quality Management program shall be initiated and documented.

It shall include a Software Quality Assurance program, which may be documented separately.

The goals of the Software Quality Management program are to identify SQA actions, describe supplier quality requirements, address client satisfaction, and provide for the identification of quality improvement needs.

Overall quality objectives are derived using the organizational guidelines and contractual requirements from the Software Project Management Planned Information.

The program information shall include the Software Quality Management organization and responsibilities, and the tools, techniques, and methodologies to implement the program.

The goals and standards to be applied to the project shall also be identified.

The goals are further expanded into quality objectives and milestones in the Software Quality Management Planned Information.

Further information related to this Activity may be found in [2] and [8].

Prior to distribution of the Software Quality Management Planned Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

3.3.3.3 Output Information

Output Information	Destination	
	Process	Activity
Software Quality Management Planned Information	Project Monitoring and Control	Manage the Project (3.2.5)
	Software Quality Management	Define Metrics (3.3.4)
		Manage Software Quality (3.3.5)
		Identify Quality Improvement Needs (3.3.6)
	Verification and Validation	Plan Verification and Validation (7.1.3)
		Collect and Analyze Metric Data (7.1.5)

3.3.4 Define Metrics

3.3.4.1 Input Information

Input Information	Source	
	Process	Activity
Software Quality Management Planned Information	Software Quality Management	Plan Software Quality Management (3.3.3)
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)

3.3.4.2 Description. The metrics required for the project, based on the Software Project Management Planned Information, shall be defined. Metrics should be applied to the products of the project and to the processes that affect the project. The metrics shall be used throughout the SLC. For each Defined Metric, Collection and Analysis Methods shall be specified.

Further information related to this Activity may be found in [6], [7], [16], and [19].

Prior to the distribution of Defined Metrics, the Verification and Validation Process (7.1.4) shall be invoked.

3.3.4.3 Output Information

Output Information	Destination	
	Process	Activity
Defined Metrics	Software Quality Management	Manage Software Quality (3.3.5)
		Plan Software Quality Management (3.3.3)
	Project Initiation	Establish Project Environm't (3.1.5)
	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
Collection and Analysis Methods	Software Quality Management	Plan Software Quality Management (3.3.3)
		Manage Software Quality (3.3.5)
	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
	Project Initiation	Establish Project Environm't (3.1.5)

3.3.5 Manage Software Quality

3.3.5.1 Input Information

Input Information	Source	
	Process	Activity
Report Log	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Resolved Problem Reported Information	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Software Quality Management Planned Information	Software Quality Management	Plan Software Quality Management (3.3.3)
Defined Metrics	Software Quality Management	Define Metrics (3.3.4)
Collection and Analysis Methods	Software Quality Management	Define Metrics (3.3.4)
Quality Improvement Recommendations	Software Quality Management	Identify Quality Improvement Needs (3.3.6)
Analysis Reported Information	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
Post-Operation Review Reported Information	Retirement	Retire System (6.4.5)

3.3.5.2 Description. Using the listed Input Information, this Activity implements the provisions of the Software Quality Management Planned Information. Based on the Software Quality Management Planned Information quality objectives and milestones, progress shall be measured and reported in Project Quality Assessments.

3.3.5.3 Output Information

Output Information	Destination	
	Process	Activity
Project Quality Assessments	Verification and Validation	Execute Verification and Validation Tasks (7.1.4)

3.3.6 Identify Quality Improvement Needs

3.3.6.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Quality Management Planned Information	Software Quality Management	Plan Software Quality Management (3.3.3)
Software Verification and Validation Planned Information	Verification and Validation	Plan Verification and Validation (7.1.3)
Evaluation Reported Information	Verification and Validation	Execute Verification and Validation Tasks (7.1.4)
Analysis Reported Information	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
Test Planned Information(s)	Verification and Validation	Plan Testing (7.1.6)
Training Planned Information(s)	Training	Plan Training Program (7.4.3)

3.3.6.2 Description. This Activity identifies needs for quality improvements and outputs the Quality Improvement Recommendations in accordance with the Software Quality Management Planned Information. This is accomplished by using the Input Information. These recommendations shall include their impact on the quality of the software delivered. In addition, applicable tools, techniques, and methods for implementation of these recommendations should be identified.

3.3.6.3 Output Information

Output Information	Destination	
	Process	Activity
Quality Improvement Recommendations	Project Monitoring and Control	Manage the Project (3.2.5)
	Software Quality Management	Manage Software Quality (3.3.5)
	External	

4. Pre-Development Processes

These are the Processes that must be performed before software development can begin.

4.1 Concept Exploration Process

4.1.1 Overview. A development effort is initiated with the identification of an idea or need for a system to be developed, whether it is a new effort or a change to all or part of an existing application. The Concept Exploration Process examines the requirements at the system level, producing a Statement of Need that initiates the System Allocation or Requirements Process. The Concept Exploration Process includes the identification of an idea or need, its evaluation and refinement, and, once boundaries are placed around it, generation of a Statement of Need for developing a system.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

4.1.2 Activities List

(1) Identify Ideas or Needs
(2) Formulate Potential Approaches
(3) Conduct Feasibility Studies
(4) Plan System Transition (If Applicable)
(5) Refine and Finalize the Idea or Need

4.1.3 Identify Ideas or Needs

4.1.3.1 Input Information

Input Information	Source	
	Process	Activity
Changing Software Requirements	External	
Customer Requests	External	
Ideas from Within the Development Organization	External	
Marketing Information Sources	External	
User Requests	External	
Enhancement Problem Reported Information	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Maintenance Recommendations	Maintenance	Reapply Software Life Cycle (6.3.3)
Feedback Data	Operation and Support	Operate the System (6.2.3)

4.1.3.2 Description. An idea or a need for a new or modified system is generated from one or more of the sources identified in the table above. Input Information to the Preliminary Statement of Need shall be documented, outlining function and performance needs. Changing Software Requirements may come from legislation, regulations, national and international standards, maintenance, etc.

Prior to distribution of the Preliminary Statement of Need to other Activities, the Verification and Validation Process (7.1.4) may be invoked.

4.1.3.3 Output Information

Output Information	Destination	
	Process	Activity
Preliminary Statement of Need	Project Initiation	Plan Project Management (3.1.6)
	Concept Exploration	Formulate Potential Approaches (4.1.4)
		Conduct Feasibility Studies (4.1.5)
		Plan System Transition (If Applicable) (4.1.6)
		Refine and Finalize the Idea or Need (4.1.7)

4.1.4 Formulate Potential Approaches

4.1.4.1 Input Information

Input Information	Source	
	Process	Activity
Development Resources and Budget	External	
Market Availability Data	External	
Resource Information	External	
Preliminary Statement of Need	Concept Exploration	Identify Ideas or Needs (4.1.3)

4.1.4.2 Description. Using Resource Information, budget data, and availability of third party software products, Potential Approaches shall be developed based upon the Preliminary Statement of Need and any data pertinent to the decision to develop or acquire the system. The Formulate Potential Approaches Activity shall also produce the constraints and benefits with regard to development of the software. The Constraints and Benefits should include all aspects of the life cycle.

Prior to release of Constraints and Benefits and Potential Approaches, the following Processes may be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

4.1.4.3 Output Information

Output Information	Destination	
	Process	Activity
Constraints and Benefits	Concept Exploration	Conduct Feasibility Studies (4.1.5)
		Refine and Finalize the Idea or Need (4.1.7)
Potential Approaches	Concept Exploration	Conduct Feasibility Studies (4.1.5)
		Refine and Finalize the Idea or Need (4.1.7)

4.1.5 Conduct Feasibility Studies

4.1.5.1 Input Information

Input Information	Source	
	Process	Activity
Preliminary Statement of Need	Concept Exploration	Identify Ideas or Needs (4.1.3)
Constraints and Benefits	Concept Exploration	Formulate Potential Approaches (4.1.4)
Potential Approaches	Concept Exploration	Formulate Potential Approaches (4.1.4)

4.1.5.2 Description. The feasibility study shall include the analysis of the idea or need, Potential Approaches, and all life cycle Constraints and Benefits. Modeling and prototyping techniques may be considered. In conducting the feasibility study, there may be a need to decide whether to make or buy the system, in part or in total. Justification for each Recommendation shall be fully documented and formally approved by all concerned organizations (including the user and the developer).

Prior to the distribution of the Recommendations, the Verification and Validation Process (7.1.4) may be invoked.

4.1.5.3 Output Information

Output Information	Destination	
	Process	Activity
Recommendations	Project Initiation	Plan Project Management (3.1.6)
	Concept Exploration	Plan System Transition (If Applicable) (4.1.6)
		Refine and Finalize the Idea or Need (4.1.7)
	System Allocation	Analyze Functions (4.2.3)

4.1.6 Plan System Transition (If Applicable)

4.1.6.1 Input Information

Input Information	Source	
	Process	Activity
Retirement Planned Information	External	
Preliminary Statement of Need	Concept Exploration	Identify Ideas or Needs (4.1.3)
Recommendations	Concept Exploration	Conduct Feasibility Studies (4.1.5)

4.1.6.2 Description. This Activity is applicable only when an existing system (automated or manual) is being replaced with a new system. The transition shall be planned and documented in accordance with the Retirement Planned Information of the system being replaced, Preliminary Statement of Need, and recommended solutions. Transition strategies and tools shall be part of the Transition Planned Information. A Transition Impact Statement shall also be produced.

Prior to distribution of the Transition Planned Information, the following Processes may be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

4.1.6.3 Output Information

Output Information	Destination	
	Process	Activity
Transition Impact Statement	Project Monitoring and Control	Analyze Risks (3.2.3)
Transition Planned Information	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)
	Installation	Plan Installation (6.1.3)

4.1.7 Refine and Finalize the Idea or Need

4.1.7.1 Input Information

Input Information	Source	
	Process	Activity
Preliminary Statement of Need	Concept Exploration	Identify Ideas or Needs (4.1.3)
Constraints and Benefits	Concept Exploration	Formulate Potential Approaches (4.1.4)
Potential Approaches	Concept Exploration	Formulate Potential Approaches (4.1.4)
Recommendations	Concept Exploration	Conduct Feasibility Studies (4.1.5)
Transition Planned Information (If Applicable)	Concept Exploration	Plan System Transition (If Applicable) (4.1.6)

4.1.7.2 Description. The idea or need shall be refined by analyzing the Preliminary Statement of Need, the Potential Approaches, Recommendations, and Transition Planned Information (If Applicable). An approach shall be selected and documented that refines the initial idea or need.

Based upon the refined ideas or needs, a Statement of Need shall be generated that identifies the software idea, need, or desire, the recommended approach for its implementation, and any data pertinent to a management decision concerning the initiation of the described development effort.

Prior to distribution of the Statement of Need, the following Processes may be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)

4.1.7.3 Output Information

Output Information	Destination	
	Process	Activity
Statement of Need	Project Initiation	Map Activitieso Software Life Cycle Model (3.1.3)
		Allocate Project Resources (3.1.4)
		Establish Project Environment (3.1.5)
		Plan Project Management (3.1.6)
	Project Monitoring and Control	Analyze Risks (3.2.3)
	System Allocation	Analyze Functions (4.2.3)
		Develop System Architecture (4.2.4)

4.2 System Allocation Process

4.2.1 Overview. The System Allocation Process is the bridge between Concept Exploration and the definition of software requirements. This Process maps the required functions to software and hardware.

The Statement of Need forms the basis for the analysis of the system, resulting in system requirements. This definition determines the inputs to the system, the processing to be applied to the inputs, and the required outputs. The software and hardware operational functions are also identified in these definitions.

The architecture of the system must be developed during the System Allocation Process. The system functions are derived from system requirements, and the hardware, software, and operational requirements are identified. These requirements are analyzed to produce System Functional Software Requirements and System Functional Hardware Requirements. The hardware, software, and operational interfaces must be defined and closely monitored. The hardware requirements analysis is not discussed in this document since it is beyond the scope of this standard.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

4.2.2 Activities List

(1) Analyze Functions
(2) Develop System Architecture
(3) Decompose System Requirements

4.2.3 Analyze Functions

4.2.3.1 Input Information

Input Information	Source	
	Process	Activity
Recommendations	Concept Exploration	Conduct Feasibility Studies (4.1.5)
Statement of Need	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)

4.2.3.2 Description. The Statement of Need and Recommendations for solution shall be analyzed to identify the functions of the total system. Once the functions have been defined, they are delineated in the Functional Description of the System and used to develop the system architecture and identify the hardware and software functions.

Prior to the distribution of the Functional Description of the System, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

4.2.3.3 Output Information

Output Information	Destination	
	Process	Activity
Functional Description of the System	System Allocation	Develop System Architecture (4.2.4)
		Decompose System Requirements (4.2.5)
	Requirements	Define Interface Requirements (5.1.4)

4.2.4 Develop System Architecture

4.2.4.1 Input Information

Input Information	Source	
	Process	Activity
Project Environment	Project Initiation	Establish Project Environment (3.1.5)
Statement of Need	Concept Exploration	Refine and Finalize the Idea or Need (4.1.7)
Functional Description of the System	System Allocation	Analyze Functions (4.2.3)

4.2.4.2 Description. The Statement of Need and the Functional Description of the System shall be transformed into the System Architecture, using the methodology, standards, and tools established by the organization. The System Architecture becomes the basis for the Design Process and the determination of the hardware and software functions.

4.2.4.3 Output Information

Output Information	Destination	
	Process	Activity
System Architecture	System Allocation	Decompose System Requirements (4.2.5)
	Design	Perform Architectural Design (5.2.3)

4.2.5 Decompose System Requirements

4.2.5.1 Input Information

Input Information	Source	
	Process	Activity
Functional Description of the System	System Allocation	Analyze Functions (4.2.3)
System Architecture	System Allocation	Develop System Architecture (4.2.4)

4.2.5.2 Description. The system functions documented in the Functional Description of the System shall be divided according to the System Architecture to form software requirements, hardware requirements, and the system interfaces. The System Interface Requirements define the interfaces that are external to the system and the interfaces between configuration items that comprise the system. Note that the hardware requirements go to an external destination since they are beyond the scope of this standard. The decomposition of the system may result in requirements for more than one project. Each software project shall be managed individually.

Prior to distribution of the requirements produced by this Activity, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)
(4) Training (7.4.4)

4.2.5.3 Output Information

Output Information	Destination	
	Process	Activity
System Functional Hardware Requirements	External	
System Functional Software Requirements	Project Initiation	Allocate Resources (3.1.4)
	Requirements	Define and Develop Software Requirements (5.1.3)
		Define Interface Requirements (5.1.4)
System Interface Requirements (If Applicable)	Requirements	Define and Develop Software Requirements (5.1.3)
		Define Interface Requirements (5.1.4)
	External	

5. Development Processes

These are the Processes that must be performed during the development of a software product.

5.1 Requirements Process

5.1.1 Overview. This Process includes those Activities directed toward the development of software requirements. In the development of a system containing both hardware and software components, the Requirements Process follows the development of total system requirements, and the functional allocation of those system requirements to hardware and software. For a system involving only software development, this effort begins once the Statement of Need is completed.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

5.1.2 Activities List

(1) Define and Develop Software Requirements
(2) Define Interface Requirements
(3) Prioritize and Integrate Software Requirements

5.1.3 Define and Develop Software Requirements

5.1.3.1 Input Information

Input Information	Source	
	Process	Activity
Installation Support Requirements	External	
System Constraints	External	
Project Environment	Project Initiation	Establish Project Environment (3.1.5)
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Analysis of Risks	Project Monitoring and Control	Analyze Risks (3.2.3)
System Functional Software Requirements (If Applicable)	System Allocation	Decompose System Requirements (4.2.5)
System Interface Requirements (If Applicable)	System Allocation	Decompose System Requirements (4.2.5)

5.1.3.2 Description. The first Activity in this Process, defining the software requirements, is iterative in nature. Whether the software development constitutes the entire project or is part of a system (hardware and software), software requirements, including constraints, shall be generated from Input Information documents and the results of modeling, prototyping, or other techniques.

Using the above Input Information, the developer shall analyze the software requirements to determine traceability, clarity, validity, testability, safety, and any other project-specific characteristics. The use of a comprehensive methodology is recommended to ensure that requirements are complete and consistent. Techniques such as structured analysis, modeling, prototyping, or transaction analysis are helpful in this Activity. When needed, the requirements for a data base shall be included in the requirements.

The Preliminary Software Requirements shall include consideration of System Constraints such as timing, sizing, language, marketing restrictions, and technology.

Further information related to this Activity may be found in [5].

Prior to the distribution of the Preliminary Software Requirements and Installation Requirements, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

5.1.3.3 Output Information

Output Information	Destination	
	Process	Activity
Preliminary Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
		Define Interface Requirements (5.1.4)
	Verification and Validation	Plan Testing (7.1.6)
Installation Requirements	Installation	Plan Installation (6.1.3)

5.1.4 Define Interface Requirements

5.1.4.1 Input Information

Input Information	Source	
	Process	Activity
System Constraints	External	
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Preliminary Software Requirements	Requirements	Define and Develop Software Requirements (5.1.3)
Functional Description of the System (If Applicable)	System Allocation	Analyze Functions (4.2.3)
System Functional Software Requirements (If Applicable)	System Allocation	Decompose System Requirements (4.2.5)
System Interface Requirements (If Applicable)	System Allocation	Decompose System Requirements (4.2.5)

5.1.4.2 Description. All user, software, and hardware interfaces shall be defined using the applicable Input Information. These interfaces shall be defined either as requirements or as constraints and shall be reviewed by all involved parties.

The user interface is critical in determining the usability of the system. The user interface definition shall specify not only the information flow between the user and the system but also how a user goes about using the system. For a complex interactive system, user interface definition may be a separate document.

The Software Interface Requirements shall specify all software interfaces required to support the development and execution of the software system. Software interfaces may be affected by System Constraints including operating system, data base management system, language compiler, tools, utilities, network protocol drivers, and hardware interfaces.

Prior to the distribution of the Output Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)
(3) Software Configuration Management (7.2.5)

5.1.4.3 Output Information

Output Information	Destination	
	Process	Activity
Software Interface Requirements	Project Monitoring and Control	Analyze Risks (3.2.3)
	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
	Design	Design Interfaces (5.2.5)
	Implementation	Create Operating Documentation (5.3.6)

5.1.5 Prioritize and Integrate Software Requirements

5.1.5.1 Input Information

Input Information	Source	
	Process	Activity
Preliminary Software Requirements	Requirements	Define and Develop Software Requirements (5.1.3)
Software Interface Requirements	Requirements	Define Interface Requirements (5.1.4)

5.1.5.2 Description. The functional and performance requirements shall be reviewed and a prioritized list of requirements shall be produced, addressing any tradeoffs that may be needed. The organization of the emerging Software Requirements shall be reviewed and revised as necessary. While completing the requirements, a particular design shall not be imposed (i.e., design decisions are made in the Design Process). The Software Requirements shall describe the functional, interface, and performance requirements. It shall also define the required operational and support environments.

Further information related to this Activity may be found in [5].

Prior to distribution of the Software Requirements, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

5.1.5.3 Output Information

Output Information	Destination	
	Process	Activity
Software Requirements	Project Monitoring and Control	Analyze Risks (3.2.3)
	Design	All Activities (5.2)
	Implementation	Create Test Data (5.3.3)
		Plan Integration (5.3.7)
	Verification and Validation	Plan Testing (7.1.6)
		Develop Test Specification(s) (7.1.7)
	Training	Plan Training Program (7.4.3)

5.2 Design Process

5.2.1 Overview. During the Design Process, major decisions are made that determine the structure of the system. The objective of the Design Process is to develop a coherent, well-organized representation of the software system that meets the Software Requirements.

The Design Process maps the "what to do" of requirements specifications into the "how to do it" of design specifications. At the architectural design level, the focus is on the functions and structure of the software components that comprise the software system. At the detailed design level, the emphasis is on the data structures and algorithms that are used within each software component.

The Perform Architectural Design and Perform Detailed Design Activities are usually carried out in sequence because detailed design is derived from the architectural design. They differ from each other in the level of design detail. Other Design Process Activities may be carried out in parallel with these Activities.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

5.2.2 Activities List

(1) Perform Architectural Design
(2) Design Data Base (If Applicable)
(3) Design Interfaces
(4) Select or Develop Algorithms (If Applicable)
(5) Perform Detailed Design

5.2.3 Perform Architectural Design

5.2.3.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
System Architecture	System Allocation	Develop System Architecture (4.2.4)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)

5.2.3.2 Description. The Perform Architectural Design Activity transforms the Software Requirements and the System Architecture into high-level design concepts. During this Activity the software components constituting the software system and their structures are identified. Purchased software and the contents of the software libraries (as referenced in the SPMP) may influence the architectural design. Techniques such as modeling and prototyping may be used to evaluate alternative designs if called for in the SPMP.

By the end of the Perform Architectural Design Activity, the design description of each of the software components shall have been completed. The data, relationships, and constraints shall be specified. In addition, all internal interfaces (among components) shall be defined. This Activity shall create the Software Architectural Design Description.

Prior to distribution of the Software Architectural Design Description, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

5.2.3.3 Output Information

Output Information	Destination	
	Process	Activity
Software Architectural Design Description	Design	Perform Detailed Design (5.2.7)

5.2.4 Design Data Base (If Applicable)

5.2.4.1 Input Information

Input Information	Source	
	Process	Activity
Project Environment	Project Initiation	Establish Project Environment (3.1.5)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)

5.2.4.2 Description. The Design Data Base Activity applies when a data base is to be created as a part of the project. This Activity shall specify the information structure outlined in the Software Requirements and its characteristics within the software system. The Design Data Base Activity involves three separate but dependent steps: conceptual data base design, logical data base design, and physical data base design. Techniques such as data dictionary, data base optimization, and data modeling may be considered. Requirements are molded into an external schema that describes data entities, attributes, relationships, and constraints. The various external schemas are integrated into a single conceptual schema. The conceptual schema is then mapped into an implementation-dependent logical schema. Finally, the physical data structures and access paths are defined. The result of this Activity is to generate the Data Base Description.

Prior to distribution of the Data Base Description, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

5.2.4.3 Output Information

Output Information	Destination	
	Process	Activity
Data Base Description	Design	Perform Detailed Design (5.2.7)

5.2.5 Design Interfaces

5.2.5.1 Input Information

Input Information	Source	
	Process	Activity
Software Interface Requirements	Requirements	Define Interface Requirements (5.1.4)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)

5.2.5.2 Description. The Design Interfaces Activity shall be concerned with the interfaces of the software system contained in the Software Requirements and Software Interface Requirements. This Activity shall consolidate these interface descriptions into a single Interface Description of the software system.

5.2.5.3 Output Information

Output Information	Destination	
	Process	Activity
Interface Description	Design	Perform Detailed Design (5.2.7)

5.2.6 Select or Develop Algorithms

5.2.6.1 Input Information

Input Information	Source	
	Process	Activity
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)

5.2.6.2 Description. This Activity is concerned with selecting or developing a procedural representation of the functions specified in the Software Requirements for each software component and data structure. The algorithms shall completely satisfy the applicable functional and/or mathematical specifications. To the extent possible, the use of existing algorithms should be considered.

Prior to distribution of the Algorithm Descriptions, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

5.2.6.3 Output Information

Output Information	Destination	
	Process	Activity
Algorithm Descriptions	Design	Perform Detailed Design (5.2.7)

5.2.7 Perform Detailed Design

5.2.7.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
Software Architectural Design Description	Design	Perform Architectural Design (5.2.3)
Data Base Description (If Applicable)	Design	Design Data Base (If Applicable) (5.2.4)
Interface Description	Design	Design Interfaces (5.2.5)
Algorithm Descriptions	Design	Select or Develop Algorithms (5.2.6)

5.2.7.2 Description. In the Perform Detailed Design Activity, design alternatives shall be chosen for implementing the functions specified for each software component. By the end of this Activity, the data structure, algorithm, and control information of each software component shall be specified. The Software Design Description (SDD) contains the consolidated data for all of the above Input Information. The details of the interfaces shall be identified within the SDD.

For further information on this topic, see [12].

Prior to distribution of the SDD, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

5.2.7.3 Output Information

Output Information	Destination	
	Process	Activity
Software Design Description	Project Monitoring and Control	Analyze Risks (3.2.3)
	Implementation	Create Test Data (5.3.3)
		Create Source (5.3.4)
		Create Operating Documentation (5.3.6)
		Plan Integration (5.3.7)
	Verification and Validation	Plan Testing (7.1.6)
		Develop Test Specification(s) (7.1.7)
	Training	Develop Training Materials (7.4.4)

5.3 Implementation Process

5.3.1 Overview. The Activities completed during the Implementation Process result in the transformation of the Detailed Design representation of a software product into a programming language realization. This Process produces the source code, data base (if applicable), and the documentation constituting the physical manifestation of the design. In addition, the code and data base are integrated. Care must also be taken during the Implementation Process to apply the appropriate coding standards.

The output of this Process must be the subject of all subsequent testing and validation. The code and data base, along with documentation produced during previous Processes, are the first complete representation of the software product.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

5.3.2 Activities List

(1) Create Test Data
(2) Create Source
(3) Generate Object Code
(4) Create Operating Documentation
(5) Plan Integration
(6) Perform Integration

5.3.3 Create Test Data

5.3.3.1 Input Information

Input Information	Source	
	Process	Activity
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
Software Design Description	Design	Perform Detailed Design (5.2.7)
Source Code (If Applicable)	Implementation	Create Source (5.3.4)
Data Base (If Applicable)	Implementation	Create Source (5.3.4)
Test Planned Information(s)	Verification and Validation	Plan Testing (7.1.6)
Test Requirements	Verification and Validation	Develop Test Requirements (7.1.7)

5.3.3.2 Description. Using the Software Requirements, the Software Design Description (SDD), and the Source Code (when required), Test Data shall be generated. The Test Planned Information(s) describe the test environment. Test Requirements define the type of test data to be used. To support the testing effort, test Stubs and Drivers may be generated at this time for each item to be tested. The test drivers allow the execution of software tests on an individual or integrated basis. Test Data may be loaded for use in testing the data base.

Further information may be found in [10].

5.3.3.3 Output Information

Output Information	Destination	
	Process	Activity
Stubs and Drivers (If Applicable)	Implementation	Perform Integration (5.3.8)
Test Data	Verification and Validation	Execute the Tests (7.1.8)

5.3.4 Create Source

5.3.4.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Design Description	Design	Perform Detailed Design (5.2.7)

5.3.4.2 Description. The Source Code, including suitable comments, shall be generated using the project environment, as found in the Software Project Management Planned Information (SPMP) and the Software Design Description. If the software requires a Data Base, then the Data Base utilities may need to be coded. If Source Code is going to be used to create test data, the Source Code shall be made available to the Create Test Data Activity (5.3.3).

5.3.4.3 Output Information

Output Information	Destination	
	Process	Activity
Data Base (If Applicable)	Implementation	Create Test Data (5.3.3)
		Generate Object Code (5.3.5)
Source Code (If Applicable)	Implementation	Create Test Data (5.3.3)
Source Code	Implementation	Generate Object Code (5.3.5)

5.3.5 Generate Object Code

5.3.5.1 Input Information

Input Information	Source	
	Process	Activity
Data Base (If Applicable)	Implementation	Create Source (5.3.4)
Source Code	Implementation	Create Source (5.3.4)

5.3.5.2 Description. The code shall be grouped into processable units. (This will be dictated by selected language and design information.) All assembly language units shall be assembled and all high-level language units compiled into Object Code. Syntactically incorrect code, identified by the assembler or compiler output, shall be reworked until the source code can be processed free of syntactical errors. These units shall be debugged. If a Data Base is coded, it too shall be debugged.

Prior to distribution of the software, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

5.3.5.3 Output Information

Output Information	Destination	
	Process	Activity
Corrected Data Base (If Applicable)	Implementation	Perform Integration (5.3.8)
Corrected Source Code	Implementation	Perform Integration (5.3.8)
Object Code	Implementation	Perform Integration (5.3.8)

5.3.6 Create Operating Documentation

5.3.6.1 Input Information

Input Information	Source	
	Process	Activity
Software Design Description	Design	Perform Detailed Design (5.2.7)
Software Interface Requirements	Requirements	Define Interface Requirements (5.1.4)
Documentation Planned Information(s)	Documentation Development	Plan Documentation (7.3.3)

5.3.6.2 Description. This Activity shall produce the software project's operating documentation from the SDD and the Software Interface Requirements in accordance with the Documentation Planned Information. The Operating Documentation is required for installing, operating, and supporting the system throughout the life cycle.

For further information, [21] may be used.

Prior to distribution of the documents listed below, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

5.3.6.3 Output Information

Output Information	Destination	
	Process	Activity
Operating Documentation	Installation	Distribute Software (6.1.4)

5.3.7 Plan Integration

5.3.7.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
Software Design Description	Design	Perform Detailed Design (5.2.7)
Test Planned Information(s)	Verification and Validation	Plan Testing (7.1.6)

5.3.7.2 Description. During the Plan Integration Activity, the Software Requirements and the SDD are analyzed to determine the order of combining software components into an overall system. The project environment, as defined in the SPMP, shall be considered when planning integration. The integration methods shall be documented in the Integration Planned Information. The Integration Planned Information shall be coordinated with the Test Planned Information(s) and they may be combined.

Prior to distribution of the Integration Planned Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

5.3.7.3 Output Information

Output Information	Destination	
	Process	Activity
Integration Planned Information	Project Monitoring and Control	Analyze Risks (3.2.3)
		Manage the Project (3.2.5)
	Implementation	Perform Integration (5.3.8)
	Verification and Validation	Plan Testing (7.1.6)

5.3.8 Perform Integration

5.3.8.1 Input Information

Input Information	Source	
	Process	Activity
Stubs and Drivers (If Applicable)	Implementation	Create Test Data (5.3.3)
Corrected Data Base (If Applicable)	Implementation	Generate Object Code (5.3.5)
Corrected Source Code	Implementation	Generate Object Code (5.3.5)
Integration Planned Information	Implementation	Plan Integration (5.3.7)
System Components	External	
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Object Code	Implementation	Generate Object Code (5.3.5)
Tested Software	Verification and Validation	Execute the Tests (7.1.8)

5.3.8.2 Description. This Activity shall execute the Integration Planned Information. This is accomplished by appropriately combining the Corrected Data Base, Corrected Source Code, Object Code, and Stubs and Drivers, as specified, into Integrated Software. Other necessary Object Code, from the project environment as defined in the SPMP, shall also be integrated. If a system includes both hardware and software components, the system integration may be included as part of this Activity.

Prior to software integration and the distribution of the Integrated Software, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

5.3.8.3 Output Information

Output Information	Destination	
	Process	Activity
Integrated Software	Verification and Validation	Execute the Tests (7.1.8)

6. Post-Development Processes

These are the Processes that must be performed to install, operate, support, maintain, and retire a software product.

6.1 Installation Process

6.1.1 Overview. Installation consists of the transportation and installation of a software system from the development environment to the target environment. It includes the necessary software modifications, checkout in the target environment, and customer acceptance. If a problem arises, it must be identified and reported; if necessary and possible, a temporary "work-around" may be applied.

During the Installation Process, the software to be delivered is installed, operationally checked out, and monitored. This effort culminates in formal customer acceptance. The scheduling of turnover and customer acceptance is defined in the Software Project Management Planned Information (SPMP).

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

6.1.2 Activities List

(1) Plan Installation
(2) Distribute Software
(3) Install Software
(4) Accept Software in Operational Environment

6.1.3 Plan Installation

6.1.3.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Installation Requirements	Requirements	Define and Develop Software Requirements (5.1.3)
Transition Planned Information (if applicable)	Concept Exploration	Plan System Transition (4.1.6)
Operating Documentation	Implementation	Create Operating Documentation (5.3.6)

6.1.3.2 Description. The tasks to be performed during installation shall be described in Software Installation Planned Information. The Installation Requirements and the other Input Information shall be analyzed to guide the development of the Software Installation Planned Information. This Planned Information, the associated documentation, and the developed software shall be used to install the software product.

Prior to distribution of the Software Installation Planned Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)
(4) Training (7.4.4)

6.1.3.3 Output Information

Output Information	Destination	
	Process	Activity
Software Installation Planned Information	Installation	Distribute Software (6.1.4)

6.1.4 Distribute Software

6.1.4.1 Input Information

Input Information	Source	
	Process	Activity
Operating Documentation	Implementation	Create Operating Documentation (5.3.6)
Software Installation Planned Information	Installation	Plan Installation (6.1.3)
Data Base Data	External	
Tested Software	Verification and Validation	Execute the Tests (7.1.8)
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)

6.1.4.2 Description. During this Activity, the Tested Software with necessary Data Base Data, Operating Documentation, and Installation Planned Information shall be packaged onto their respective media as designated in the SPMP. The Packaged Software is distributed to the appropriate site(s) for installation. The Installation Planned Information is distributed as appropriate to the site(s) to facilitate the installation efforts. The Packaged Operating Documentation shall be available for operation of the system.

Prior to distribution of the Output Information, the following Processes shall be invoked:

(1) Software Configuration Management (7.2.5)
(2) Verification and Validation (7.1.4)
(3) Documentation Development (7.3.4)

6.1.4.3 Output Information

Output Information	Destination	
	Process	Activity
Packaged Operating Documentation	Operation and Support	Operate the System (6.2.3)
Packaged Installation Planned Information	Installation	Install Software (6.1.5)
Packaged Software	Installation	Install Software (6.1.5)

6.1.5 Install Software

6.1.5.1 Input Information

Input Information	Source	
	Process	Activity
Packaged Installation Planned Information	Installation	Distribute Software (6.1.4)
Packaged Operating Documentation	Installation	Distribute Software (6.1.4)
Packaged Software Installation	Installation	Distribute Software (6.1.4)
Data Base Data	External	

6.1.5.2 Description. The packaged software and any required Data Base Data shall be installed in the target environment according to the procedures in the Software Installation Planned Information. This may include tailoring by the customer. The Installation Reported Information shall document the installation and any problems encountered.

6.1.5.3 Output Information

Output Information	Destination	
	Process	Activity
Installation Reported Information	Project Monitoring and Control	Manage the Project (3.2.5)
Installed Software	Installation	Accept Software in Operational Environment (6.1.6)

6.1.6 Accept Software in Operational Environment

6.1.6.1 Input Information

Input Information	Source	
	Process	Activity
User Acceptance Planned Information	External	
Test Summary Reported Information	Verification and Validation	Execute the Tests (7.1.8)
Installed Software	Installation	Install Software (6.1.5)

6.1.6.2 Description. The software acceptance shall consist of analysis of the Test Summary Reported Information(s) according to the User Acceptance Planned Information to assure that the Installed Software performs as expected. When the results of the analysis satisfy the requirements of the User Acceptance Planned Information, the Installed Software System is accepted by the User.

Prior to completion of accepting software in the operational environment, the following Processes should be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

6.1.6.3 Output Information

Output Information	Destination	
	Process	Activity
Customer Acceptance	External	
Installed Software System	Operation and Support	Operate the System (6.2.3)
	Retirement	Conduct Parallel Operations (If Applicable) (6.4.4)

6.2 Operation and Support Process

6.2.1 Overview. The Operation and Support Process involves user operation of the system and ongoing support. Support includes providing technical assistance, consulting with the user, and recording user support requests by maintaining a Support Request Log. Thus the Operation and Support Process may trigger Maintenance Activities via the ongoing Project Monitoring and Control Process, which will provide information re-entering the software life cycle (SLC).

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

6.2.2 Activities List

(1) Operate the System
(2) Provide Technical Assistance and Consulting
(3) Maintain Support Request Log

6.2.3 Operate the System

6.2.3.1 Input Information

Input Information	Source	
	Process	Activity
Packaged Operating Documentation	Installation	Distribute Software (6.1.4)
Support Planned Information	Project Initiation	Plan Project Management (3.1.6)
Installed Software System	Installation	Accept Software in Operational Environment (6.1.6)

6.2.3.2 Description. During this Activity, the Installed Software System shall be utilized in the intended environment and in accordance with the operating instructions. Feedback Data are collected for product and documentation improvement and system tuning. The user shall analyze the Feedback Data and identify System Anomalies (which include desired enhancements). System Anomalies are reported.

Prior to the distribution of the Output Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

6.2.3.3 Output Information

Output Information	Destination	
	Process	Activity
System Anomalies	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Operation Logs	External	
Feedback Data	Project Monitoring and Control	Manage the Project (3.2.5)
	Concept Exploration	Identify Ideas or Needs (4.1.3)

6.2.4 Provide Technical Assistance and Consulting

6.2.4.1 Input Information

Input Information	Source	
	Process	Activity
Request for Support	External	
Support Planned Information	Project Initiation	Plan Project Management (3.1.6)

6.2.4.2 Description. This Activity applies after the user has accepted the software. The support function shall include providing responses to the user's technical questions or problems. A Support Response is sent to the Maintain Support Request Log Activity so that feedback can be provided to other Processes.

6.2.4.3 Output Information

Output Information	Destination	
	Process	Activity
Support Response	Operation and Support	Maintain Support Request Log (6.2.5)
	External	

6.2.5 Maintain Support Request Log

6.2.5.1 Input Information

Input Information	Source	
	Process	Activity
Support Planned Information	Project Initiation	Plan Project Management (3.1.6)
Support Response	Operation and Support	Provide Technical Assistance and Consulting (6.2.4)

6.2.5.2 Description. This Activity shall record support requests in the Support Request Log. Methodology regarding management of this Activity shall be identified in the Support Planned Information. Anomalies that are reported shall be reported also to the Project Monitoring and Control Process. Prior to release of the Support Request Log, the Verification and Validation Process (7.1.4) shall be invoked.

6.2.5.3 Output Information

Output Information	Destination	
	Process	Activity
Anomalies	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Support Request Log	Verification and Validation	Execute Verification and Validation Tasks (7.1.4)

6.3 Maintenance Process

6.3.1 Overview. The Maintenance Process is concerned with the resolution of software errors, faults, and failures. The requirement for software maintenance initiates software life cycle (SLC) changes. The SLC is remapped and executed, thereby treating the Maintenance Process as iterations of development.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

6.3.2 Activities List

(1) Reapply Software Life Cycle

6.3.3 Reapply Software Life Cycle

6.3.3.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Correction Problem Reported Information	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)

6.3.3.2 Description. The information provided by the Correction Problem Reported Information and the current Software Project Management Planned Information (SPMP) shall result in the generation of Maintenance Recommendations. These Maintenance Recommendations will then enter the SLC at the Concept Exploration Process to improve the quality of the software system.

6.3.3.3 Output Information

Output Information	Destination	
	Process	Activity
Maintenance Recommendations	Concept Exploration	Identify Ideas or Needs (4.1.3)

6.4 Retirement Process

6.4.1 Overview. The Retirement Process involves the removal of an existing system from its active support or use either by ceasing its operation or support, or by replacing it with a new system or an upgraded version of the existing system.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

6.4.2 Activities List

(1) Notify User
(2) Conduct Parallel Operations (If Applicable)
(3) Retire System

6.4.3 Notify User

6.4.3.1 Input Information

Input Information	Source	
	Process	Activity
Retirement Planned Information	Project Initiation	Plan Project Management (3.1.6)

6.4.3.2 Description. This Activity shall be the formal notification to any user (both internal and external customers) of an operating software system that is to be removed from active support or use. This notification can take any of several forms as appropriate for the individual environment. It is important that all users of the outgoing system be made aware that it will become unsupported. The actual dates of the removal of support are to be clearly specified and must allow time for current users to make whatever arrangements are necessary to respond to this notification. Included in the user notification should be one or more of the following:

(1) Description of the replacement system including its date of availability.
(2) Statement as to why the system is not being supported.
(3) Description of possible other support.

Prior to the distribution of the Official Notification, the Documentation Development Process (7.3.4) shall be invoked.

6.4.3.3 Output Information

Output Information	Destination	
	Process	Activity
Official Notification	Project Monitoring and Control	Retain Records (3.2.6)
	External	

6.4.4 Conduct Parallel Operations (If Applicable)

6.4.4.1 Input Information

Input Information	Source	
	Process	Activity
Transition Planned Information (for the replacing system)	External	
Retirement Planned Information	Project Initiation	Plan Project Management (3.1.6)
Installed Software System	Installation	Accept Software in Operational Environment (6.1.6)

6.4.4.2 Description. If the outgoing system is being replaced by a new system, this Activity may apply. This Activity shall involve a period of dual operation utilizing the retiring system for official results, while completing the preparation of the new system for formal operation. It is a period of user training on the new system and validation of the new system. The Retirement Planned Information, as well as the Transition Planned Information, may be used to provide information to conduct parallel operations for the replacing system.

While conducting this Activity, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Training (7.4.4)

6.4.4.3 Output Information

Output Information	Destination	
	Process	Activity
Parallel Operations Log	Project Monitoring and Control	Retain Records (3.2.6)

6.4.5 Retire System

6.4.5.1 Input Information

Input Information	Source	
	Process	Activity
Retirement Planned Information	Project Initiation	Plan Project Management (3.1.6)

6.4.5.2 Description. This Activity shall consist of the actual removal and archiving of the retiring system from regular usage according to the Retirement Planned Information. It may be spread over a period of time and take the form of a phased removal, or it may be the simple removal of the entire system from the active software library. Prior to the retirement, users must have been notified of the event. Any preparations for the use of a replacement system should have been completed. The Post-Operation Review Reported Information is generated at this time. The Retire System Activity must be documented in Archive Reported Information.

Prior to the final distribution of the Post-Operation Review Reported Information and Archive Reported Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

6.4.5.3 Output Information

Output Information	Destination	
	Process	Activity
Post-Operation Review Reported Information	Project Monitoring and Control	Retain Records (3.2.6)
	Software Quality Management	Manage Software Quality (3.3.5)
Archive Reported Information	External	

7. Integral Processes

These are the Processes needed to successfully complete project Activities. These Processes are utilized to ensure the completion and quality of project functions.

7.1 Verification and Validation Process

7.1.1 Overview. The Verification and Validation Process includes planning and performing both Verification and Validation tasks. Verification tasks include reviews, configuration audits, and quality audits. Validation tasks include all phases of testing. These Verification and Validation tasks are conducted throughout the software life cycle (SLC) to ensure that all requirements are satisfied. This Process addresses each life cycle Process and product.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

7.1.2 Activities List

(1) Plan Verification and Validation
(2) Execute Verification and Validation Tasks
(3) Collect and Analyze Metric Data
(4) Plan Testing
(5) Develop Test Requirements
(6) Execute the Tests

7.1.3 Plan Verification and Validation

7.1.3.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Analysis of Risks	Project Monitoring and Control	Analyze Risks (3.2.3)
Software Quality Management Planned Information	Software Quality Management	Plan Software Quality Management (3.3.3)

7.1.3.2 Description. This Activity shall be responsive to the Software Project Management Planned Information and the Software Quality Assurance (SQA) Planned Information by identifying Processes and Process Output Information to be verified and validated. The purpose and scope of the verification and validation task shall be defined for each Process and all Process Output Information. The planning shall include developing schedules, estimating resources, identifying special resources, staffing, and establishing exit or acceptance criteria. Verification and validation methods to be considered in this planning Activity include audits (e.g., functional and physical configuration, compliance), reviews (e.g., design, code, document), prototyping, inspection, formal proof, analysis, and demonstration. Special attention should be given to minimizing technical risks and verifying requirements traceability. This planning shall be documented in the Software Verification and Validation Planned Information (SVVP).

Because of the importance of testing in the Verification and Validation Process, this standard addresses testing Activities separately. Test planning and execution may be included in the Verification and Validation Planning and Execution Activities.

Further information on verification and validation planning may be found in [2], [3], [4], [6], [7], [8], [12], [13], [14], [15], and [16].

Prior to distribution of the SVVP, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)
(3) Software Configuration Management (7.2.5)

7.1.3.3 Output Information

Output Information	Destination	
	Process	Activity
Software Verification and Validation Planned Information	Software Quality Management	Identify Quality Improvement Needs (3.3.6)
	Verification and Validation	Execute Verification and Validation Tasks (7.1.4)
		Plan Testing (7.1.6)

7.1.4 Execute Verification and Validation Tasks

7.1.4.1 Input Information

Input Information	Source	
	Process	Activity
Item(s) to Be Evaluated	Creating Process	
Resolved Problem Reported Information	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Project Quality Assessments	Software Quality Management	Manage Software Quality (3.3.5)
Support Request Log	Operation and Support	Maintain Support Request Log (6.2.5)
Software Verification and Validation Planned Information	Verification and Validation	Plan Verification and Validation (7.1.3)
Basis or Bases for Evaluation	External	
	Creating Process	

7.1.4.2 Description. This Activity shall include performing the tasks specified in the SVVP using the Input Information. Results shall be provided in Evaluation Reported Information. Anomalies identified during the performance of these tasks shall be reported.

Further information related to this Activity may be found in [2], [3], [8], [12], [13], and [14].

Prior to distribution of Evaluation Reported Information, the following Processes shall be invoked:

(1) Documentation Development (7.3.4)
(2) Software Configuration Management (7.2.5)

7.1.4.3 Output Information

Output Information	Destination	
	Process	Activity
Evaluation Reported Information	Project Monitoring and Control	Manage the Project (3.2.5)
	Software Quality Management	Identify Quality Improvement Needs (3.3.6)
	Verification and Validation	Collect and Analyze Metric Data (7.1.5)
	Creating Process	
Anomalies	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)

7.1.5 Collect and Analyze Metric Data

7.1.5.1 Input Information

Input Information	Source	
	Process	Activity
Support Personnel Reported Information	External	
User Input Information	External	
Metric Data	Originating Process	Originating Activity
Correction Problem Reported Information	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Enhancement Problem Reported Information	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Report Log	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Resolved Problem Reported Information	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)
Software Quality Management Planned Information	Software Quality Management	Plan Software Quality Management (3.3.3)
Defined Metrics	Software Quality Management	Define Metrics (3.3.4)
Collection and Analysis Methods	Software Quality Management	Define Metrics (3.3.4)
Evaluation Reported Information	Verification and Validation	Execute Verification and Validation Tasks (7.1.4)

7.1.5.2 Description. This Activity collects Evaluation Reported Information, Problem Reported Information, and project-generated Metric Data, as stated in the Software Quality Management Planned Information. The data shall be analyzed using defined methodologies. This Activity shall identify improvements in both quality and requirements as a result of Support Personnel and User Input Information. Analysis Reported Information shall be generated describing the results of metrics validation and analysis, defect trend analysis, and the user's view analysis.

Further information related to this Activity may be found in [6], [7], [15], and [16].

Prior to distribution of the Analysis Reported Information, the following Processes should be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

7.1.5.3 Output Information

Output Information	Destination	
	Process	Activity
Analysis Reported Information	Project Monitoring and Control	Analyze Risks (3.2.3)
		Perform Contingency Planning (3.2.4)
		Manage the Project (3.2.5)
	Software Quality Management	Manage Software Quality (3.3.5)
		Identify Quality Improvement Needs (3.3.6)

7.1.6 Plan Testing

7.1.6.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Preliminary Software Requirements	Requirements	Define and Develop Software Requirements (5.1.3)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
Software Design Description	Design	Perform Detailed Design (5.2.7)
Integration Planned Information	Implementation	Plan Integration (5.3.7)
Software Verification and Validation Planned Information	Verification and Validation	Plan Verification and Validation (7.1.3)

7.1.6.2 Description. This Activity shall identify the overall scope, approach, resources, and schedule of the testing tasks over the entire SLC and document them in Test Planned Information(s). The Test Planned Information(s) shall define the generic levels of testing and the basic test environment and structure needed to support required levels of testing. Each Test Planned Information shall identify the items to be tested, the requirements to be tested, and the test pass-or-fail criteria based on the Software Requirements and the Software Design Description (SDD) (as soon as available). The Test Planned Information(s) shall identify test coverage criteria, the tools and approaches being applied, the environmental needs, the testing tasks to be performed, the organizational structure, the management controls and reporting procedures, and the risks and contingencies.

The Test Planned Information(s) shall be coordinated with, and may be combined with, the Integration Planned Information and SVVP.

Further information related to this Activity may be found in [4], [10], [12], and [18].

Prior to distribution of the Test Planned Information(s), the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

7.1.6.3 Output Information

Output Information	Destination	
	Process	Activity
Test Planned Information(s)	Project Monitoring and Control	Analyze Risks (3.2.3)
		Manage the Project (3.2.5)
	Software Quality Management	Identify Quality Improvement Needs (3.3.6)
	Implementation	Create Test Data (5.3.3)
		Plan Integration (5.3.7)
	Verification and Validation	Develop Test Requirements (7.1.7)
		Execute the Tests (7.1.8)

7.1.7 Develop Test Requirements

7.1.7.1 Input Information

Input Information	Source	
	Process	Activity
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
Software Design Description	Design	Perform Detailed Design (5.2.7)
Test Planned Information(s)	Verification and Validation	Plan Testing (7.1.6)

7.1.7.2 Description. Test Requirements for each generic level of testing shall be developed to refine the test approach from the Test Planned Information(s) to item-specific test procedures used for test execution. The Test Requirements shall define what is to be tested, the data to be used in testing, expected results, the test environment components, and the procedures to be followed in testing. Information from the SRS, the SDD, and the Test Planned Information(s) is used to generate the Test Requirements.

Further information related to this Activity may be found in [4] and [10].

Prior to distribution of the Test Requirements, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

7.1.7.3 Output Information

Output Information	Destination	
	Process	Activity
Test Requirements	Implementation	Create Test Data (5.3.3)
	Verification and Validation	Execute the Tests (7.1.8)

7.1.8 Execute the Tests

7.1.8.1 Input Information

Input Information	Source	
	Process	Activity
Test Environment Components	External	
Test Data	Implementation	Create Test Data (5.3.3)
Integrated Software	Implementation	Perform Integration (5.3.8)
Test Planned Information(s)	Verification and Validation	Plan Testing (7.1.6)
Test Requirements	Verification and Validation	Develop Test Requirements (7.1.7)

7.1.8.2 Description. This Activity shall configure the Test Environment Components as required by the Test Requirements. Each test shall be conducted on the Integrated Software using Test Data as defined in its associated Test Requirements and in accordance with the Test Planned Information(s).

This Activity could be iterative, with several instances performed during the software's life. Not all Input Information and Output Information are required for a given iteration; the presence of any Input Information is sufficient as an entry criterion, and the creation of any Output Information is a sufficient exit criterion.

Based on comparison of actual results with expected results, according to the pass-fail criteria, a pass-fail determination shall be made and recorded in a test log. Each anomalous event that occurs during execution which requires further investigation shall be reported. The impact on the validity of the test should also be noted.

Test Summary Reported Information shall summarize the results of a test based on its Test Requirements and test log. Tested Software is that software which has successfully passed all tests at the appropriate level and met the specified criteria and requirements. Tested Software may then be further integrated with other software or sent for installation.

Further information related to this Activity may be found in [4] and [10].

Prior to distribution of the Output Information from this Activity, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

7.1.8.3 Output Information

Output Information	Destination	
	Process	Activity
Test Summary Reported Information	Project Monitoring and Control	Analyze Risks (3.2.3)
		Manage the Project (3.2.5)
	External	
Tested Software	Implementation	Perform Integration (5.3.8)
	Installation	Distribute Software (6.1.4)
Anomalies	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)

7.2 Software Configuration Management Process

7.2.1 Overview. Software Configuration Management the items in a software development project and provides both for control of the identified items and for the generation of Status Reported Information for management visibility and accountability throughout the software life cycle (SLC). Items to be managed are those defined in Software Configuration Management Planned Information (SCMP). Examples to be considered for inclusion in the SCMP are code, documentation, plans, and specifications. Configuration audits, if required by the Project, should be addressed in the Verification and Validation Process. The Software Configuration Management approach for a given project should be compatible with the Configuration Management approach being used on associated systems.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

7.2.2 Activities List

(1) Plan Configuration Management
(2) Develop Configuration Identification
(3) Perform Configuration Control
(4) Perform Status Accounting

7.2.3 Plan Configuration Management

7.2.3.1 Input Information

Input Information	Source	
	Process	Activity
Contract Deliverable List	External	
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Configuration Identification	Software Configuration Management	Develop Configuration Identification (7.2.4)

7.2.3.2 Description. This Activity shall plan and document specific software configuration management organizations and responsibilities, procedures, tools, techniques, and methodologies in an SCMP. The SCMP shall also describe how and when such procedures are to be performed.

Overall software configuration management objectives are derived using internal guidelines as well as contractual requirements from the Software Project Management Planned Information (SPMP).

Further information related to this Activity may be found in [3] and [14].

Prior to distribution of the Planned Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)

7.2.3.3 Output Information

Output Information	Destination	
	Process	Activity
Software Configuration Management Planned Information	Project Monitoring and Control	Retain Records (3.2.6)
	Software Configuration Management	Develop Configuration Identification (7.2.4)
		Perform Configuration Control (7.2.5)
		Perform Status Accounting (7.2.6)

7.2.4 Develop Configuration Identification

7.2.4.1 Input Information

Input Information	Source	
	Process	Activity
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Configuration Management Planned Information	Software Configuration Management	Plan Configuration Management (7.2.3)

7.2.4.2 Description. This Activity shall define a Configuration Identification that includes project baseline definition, titling, labeling, and numbering to reflect the structure of the product for tracking. The SCMP identifies those configuration items to be addressed by the Configuration Identification. The identification shall support the software throughout the SLC, and shall be documented in the SCMP. The Configuration Identification shall also define the documentation required to record the functional and physical characteristics of each Configuration Item.

A series of baselines shall be established as the product moves from initial idea to the maintenance phase as required by the SPMP.

Further information related to this Activity may be found in [3] and [14].

7.2.4.3 Output Information

Output Information	Destination	
	Process	Activity
Configuration Identification	Software Configuration Management	Plan Configuration Management (7.2.3)

7.2.5 Perform Configuration Control

7.2.5.1 Input Information

Input Information	Source	
	Process	Activity
Items to Be Controlled	Creating Process	
Software Configuration Management Planned Information	Software Configuration Management	Plan Configuration Management (7.2.3)

7.2.5.2 Description. This Activity controls the configuration of products. Changes to controlled products shall be tracked to assure that the configuration of the product is known at all times. Each baseline shall be established and all subsequent changes tracked relative to it. All items specified in the SCMP are subject to this change management discipline. The history of changes to each configuration item shall be maintained throughout the SLC for status accounting.

Changes to Controlled Items shall be allowed only with approval of the responsible authority. This may result in establishment of a formal software configuration control board. Controlled Items shall be maintained in a software library.

Further information related to this Activity may be found in [3] and [14].

7.2.5.3 Output Information

Output Information	Destination	
	Process	Activity
Change Status	Software Configuration Management	Perform Status Accounting (7.2.6)
Controlled Item	Creating Process	
	Project Monitoring and Control	Implement Problem Reporting Method (3.2.7)

7.2.6 Perform Status Accounting

7.2.6.1 Input Information

Input Information	Source	
	Process	Activity
Software Configuration Management Planned Information	Software Configuration Management	Plan Configuration Management (7.2.3)
Change Status	Software Configuration Management	Perform Configuration Control (7.2.5)

7.2.6.2 Description. This Activity shall include the receipt of Change Status from the Perform Configuration Control Activity and the preparation of Status Reported Information that reflects the status and history of controlled items. Status Reported Information may include such data as number of changes to date for the project, number of releases, and the latest version and revision identifiers.

Further information related to this Activity may be found in [3] and [14].

Prior to the distribution of the Status Reported Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)

7.2.6.3 Output Information

Output Information	Destination	
	Process	Activity
Status Reported Information	Project Monitoring and Control	Manage the Project (3.2.5)
	External	

7.3 Documentation Development Process

7.3.1 Overview. The Documentation Development Process for software development and usage is the set of Activities that plan, design, implement, edit, produce, distribute, and maintain those documents needed by developers and users. The purpose of the Documentation Development Process is to provide timely software documentation to those who need it, based on Input Information from the invoking Processes.

This Process covers both product- and procedure-oriented documentation for internal and external users. Examples of internal users include those who plan, design, implement, or test software. External users may include those who install, operate, apply, or maintain the software.

The Documentation Development Process occurs over various phases of the software life cycle (SLC) depending on the individual document and the timing of its development. Typically there will be multiple documents, each at different stages of development.

The Documentation Development Process has Activities that must be performed concurrently with the software development or usage. Since the software is seldom stable during development or testing, this requires effective communication and timely response between the software personnel and documentation personnel.

7.3.2 Activities List

(1) Plan Documentation
(2) Implement Documentation
(3) Produce and Distribute Documentation

7.3.3 Plan Documentation

7.3.3.1 Input Information

Input Information	Source	
	Process	Activity
Contractual Requirements	External	
Project Standards	Project Initiation	Establish Project Environment (3.1.5)
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)

7.3.3.2 Description. In this Activity, information such as the Software Project Management Planned Information (SPMP) product descriptions, schedules, and resource constraints shall be assimilated to create a consistent and disciplined approach to achieving the required documentation. The approach shall identify required documents, document production schedules and delivery schedules, and documentation standards. Responsible organizations, information sources, and intended audiences shall be defined for each document. The approach shall be documented in the Documentation Planning Information. The Documentation Planning Information shall include resource allocations for this Activity.

Additional guidance for the development of user documentation can be found in [21].

Prior to distribution of the Documentation Planning Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)

The Software Configuration Management Process (7.2.5) should also be invoked.

7.3.3.3 Output Information

Output Information	Destination	
	Process	Activity
Documentation Planned Information	Project Monitoring and Control	Retain Records (3.2.6)
	Implementation	Create Operating Documentation (5.3.6)
	Documentation Development	All Activities (7.3)

7.3.4 Implement Documentation

7.3.4.1 Input Information

Input Information	Source	
	Process	Activity
Input Information for Document	Creating Process	
Project Environment	Project Initiation	Establish Project Environment (3.1.5)
Documentation Planned Information	Documentation Development	Plan Documentation (7.3.3)

7.3.4.2 Description. This Activity includes the design, preparation, and maintenance of documentation. Those documents identified in the Documentation Planned Information shall be defined in terms of audience, approach, content, structure, and graphics. Arrangements may be made with word or text processing and graphics facilities for their support of implementation.

Input Information shall be used to produce the document, including related graphics. This involves extensive use of information sources, close communication with the responsible subject matter experts, and utilization of word or text processing and graphics tools.

Following a documentation review, any changes shall be incorporated to produce a technically correct document. Format, style, and production rules shall be applied to produce a final document.

Prior to distribution of the Document, the following Processes should be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)

7.3.4.3 Output Information

Output Information	Destination	
	Process	Activity
Document	Documentation Development	Produce and Distribute Documentation (7.3.5)

7.3.5 Produce and Distribute Documentation

7.3.5.1 Input Information

Input Information	Source	
	Process	Activity
Documentation Planned Information	Documentation Development	Plan Documentation (7.3.3)
Document	Documentation Development	Implement Documentation (7.3.4)

7.3.5.2 Description. This Activity shall provide the intended audience with the needed information collected in the document, as specified in the Documentation Planned Information. Document production and distribution may involve electronic file management, paper document reproduction and distribution, or other media handling techniques.

7.3.5.3 Output Information

Output Information	Destination	
	Process	Activity
Published Document	Project Monitoring and Control	Retain Records (3.2.6)
	Creating Process	
	External	

7.4 Training Process

7.4.1 Overview. The development of quality software products is largely dependent upon knowledgeable and skilled people. These include the developer's technical staff and management. Customer personnel may also have to be qualified to install, operate, and maintain the software. Training is therefore essential for developers, technical support staff, and customers. It is essential that Training Planned Information be completed early in the software life cycle, prior to the time when personnel would be expected to apply required expertise to the project. Plans for customer training should be prepared and reviewed with the customer.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in this section.

7.4.2 Activities List

(1) Plan the Training Program
(2) Develop Training Materials
(3) Validate the Training Program
(4) Implement the Training Program

7.4.3 Plan the Training Program

7.4.3.1 Input Information

Input Information	Source	
	Process	Activity
Applicable Information	External	
Skills Inventory	External	
Project Environment	Project Initiation	Establish Project Environment (3.1.5)
Software Project Management Planned Information	Project Initiation	Plan Project Management (3.1.6)
Software Requirements	Requirements	Prioritize and Integrate Software Requirements (5.1.5)
Training Feedback	Training	Implement the Training Program (7.4.6)
		Validate the Training Program (7.4.5)

7.4.3.2 Description. This Activity shall identify the needs for different types of training and the categories of people requiring training for each need. Customer and project documents shall be reviewed along with existing personnel inventories. This information is used to produce documented Training Planned Information. Implementation schedules shall also be generated and resources allocated to the training program. Implementation schedules, resource allocations, and training needs shall be specified in the Training Planned Information.

Prior to distribution of the Training Planned Information, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Software Configuration Management (7.2.5)
(3) Documentation Development (7.3.4)

7.4.3.3 Output Information

Output Information	Destination	
	Process	Activity
Training Planned Information	Software Quality Management	Identify Quality Improvement Needs (3.3.6)
	Training	All Activities (7.4)

7.4.4 Develop Training Materials

7.4.4.1 Input Information

Input Information	Source	
	Process	Activity
Applicable Documentation	External	
	Creating Process	
Project Environment	Project Initiation	Establish Project Environment (3.1.5)
Software Design Description	Design	Perform Detailed Design (5.2.7)
Training Planned Information	Training	Plan Training Program(7.4.3)

7.4.4.2 Description. This Activity shall consist of identification and review of all available materials that appear pertinent to the training objectives. Included in the Develop Training Materials Activity shall be the development of the substance of the training, training manual, and materials to be used in presenting the training, such as outlines, text, exercises, case studies, visuals, and models.

Prior to distribution of the Training Manual and Training Materials, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)

The Software Configuration Management Process (7.2.5) should also be invoked.

7.4.4.3 Output Information

Output Information	Destination	
	Process	Activity
Training Manual	Training	Validate Training Program (7.4.5)
Training Materials	Training	Validate Training Program (7.4.5)

7.4.5 Validate the Training Program

7.4.5.1 Input Information

Input Information	Source	
	Process	Activity
Training Planned Information	Training	Plan Training Program7.4.3)
Training Manual	Training	Develop Training Materials (7.4.4)
Training Materials	Training	Develop Training Materials (7.4.4)

7.4.5.2 Description. This Activity shall consist of presenting the training to a class of evaluators using the preliminary training manual and materials. The evaluators shall assess the training presentation and materials in detail. The purpose is to evaluate the effectiveness of the delivery and the validity of the material presented. Lessons learned in the test of the training program shall be incorporated into the material prior to a general offering. All training manuals and materials shall be evaluated and, if necessary, updated at this time.

Prior to distribution of Updated Training Manuals and Materials, the following Processes shall be invoked:

(1) Verification and Validation (7.1.4)
(2) Documentation Development (7.3.4)

The Software Configuration Management Process (7.2.5) should be invoked.

7.4.5.3 Output Information

Output Information	Destination	
	Process	Activity
Updated Training Manual	Training	Implement Training Program (7.4.6)
Updated Training Materials	Training	Implement Training Program (7.4.6)
Training Feedback	Training	Plan Training Program (7.4.3)

7.4.6 Implement the Training Program

7.4.6.1 Input Information

Input Information	Source	
	Process	Activity
Staff Participants	External	
Students	External	
Training Planned Information	Training	Plan Training Program(7.4.3)
Updated Training Manual	Training	Validate Training Program (7.4.5)
Updated Training Materials	Training	Validate Training Program (7.4.5)

7.4.6.2 Description. This Activity shall ensure the provision of all necessary materials, arrange the locations and facilities for training, assign instructors and, if necessary, train them. Included in this Activity shall be the enrolling of students and monitoring of the course effectiveness.

Lessons learned and information needed for updating the materials for the next training cycle shall be fed back into the beginning of the Training Process.

7.4.6.3 Output Information

Output Information	Destination	
	Process	Activity
Trained Personnel	Creating Process	
Training Feedback	Training	Plan Training Program (7.4.3)
Updated Skills Inventory	External	

8. Bibliography

The IEEE standards listed below and other subsequent standards should be consulted when using this document. However, compliance with this standard neither requires nor implies compliance with the listed standards. Table 2 provides a cross reference of specific Activities to other IEEE standards.

[1] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology (ANSI).[4]

[2] IEEE Std 730-1989, IEEE Standard for Software Quality Assurance Plans (ANSI).

[3] IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans (ANSI).

[4] IEEE Std 829-1983, IEEE Standard for Software Test Documentation (ANSI).

[5] IEEE Std 830-1984, IEEE Guide to Software Requirements Specifications (ANSI).

[6] IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software (ANSI).

[7] IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software (ANSI).

[8] IEEE Std 983-1986, IEEE Guide for Software Quality Assurance Planning (ANSI).

[9] IEEE Std 1002-1987, IEEE Standard Taxonomy for Software Engineering Standards (ANSI).

[10] IEEE Std 1008-1987, IEEE Standard for Software Unit Testing (ANSI).

[11] IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans (ANSI).

[12] IEEE Std 1016-1987, IEEE Recommended Practice for Software Design Descriptions (ANSI).

[13] IEEE Std 1028-1988, IEEE Standard for Software Reviews and Audits (ANSI).

[14] IEEE Std 1042-1987, IEEE Guide to Software Configuration Management Planning (ANSI).

[15] P1044, Standard for Classification of Software Errors, Faults, and Failures.[5]

[16] P1045, Standard for Software Productivity Metrics.

[17] IEEE Std 1058.1-1987, IEEE Standard for Software Project Management Plans (ANSI).

[18] P1059, Guide for Software Verification and Validation.

[4]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

[5]References [15], [16], [18], [19], and [20] are authorized standards projects that were not approved by the IEEE Standards Board at the time this document went to press. The latest drafts of the documents are available from the IEEE Service Center.

[19] P1061, Standard for a Software Quality Metrics Methodology.

[20] P1062, Recommended Practice for Software Acquisition.

[21] IEEE Std 1063-1987, IEEE Standard for Software User Documentation (ANSI).

[22] ANSI Technical Report, American National Dictionary for Information Processing, X3/TR-1-77, September 1977.

Table 2
Cross Reference of IEEE Standards and Authorized Standards Projects

	Applicability of Standards								
Activities (Paragraph Numbers) / Referenced IEEE Standards and Authorized Standards Projects	1.3.1	3.1.5	3.1.6	3.2.7	3.3.3	3.3.4	5.1.3 5.1.5	5.2.7	5.3.3 5.3.5
[1] 610.12-1990	X								
[2] 730-1989					X				
[3] 828-1990									
[4] 829-1983									
[5] 830-1984	X						X		
[6] 982.1-1988						X			
[7] 982.2-1988						X			
[8] 983-1986					X				
[9] 1002-1987		X							
[10] 1008-1987									
[11] 1012-1986									
[12] 1016-1987									
[13] 1028-1988									
[14] 1042-1987									
[15] P1044				X					
[16] 1045						X			
[17] 1058.1-1987	X		X						
[18] P1059									
[19] P1061						X			
[20] P1062									
[21] 1063-1987									

Table 2
Cross Reference of IEEE Standards and Authorized Standards Projects *(Continued)*

	Applicability of Standards								
Activities (Paragraph Numbers) / Referenced IEEE Standards	5.3.3 5.3.5	5.3.6	7.1.3	7.1.4	7.1.5	7.1.6	7.1.7 7.1.8	7.2.3 7.2.4 7.2.5 7.2.6	7.3.3
[1] 610.12-1990									
[2] 730-1989			X	X					
[3] 828-1990			X	X				X	
[4] 829-1983			X			X	X		
[5] 830-1984									
[6] 982.1-1988			X		X				
[7] 982.2-1988			X		X				
[8] 983-1986			X	X					
[9] 1002-1987									
[10] 1008-1987	X					X			
[11] 1012-1986			X	X		X	X		
[12] 1016-1987									
[13] 1028-1988			X	X					
[14] 1042-1987			X	X				X	
[15] P1044			X		X				
[16] P1045			X		X				
[17] 1058.1-1987									
[18] P1059						X			
[19] P1061									
[20] P1062									
[21] 1063-1987		X							X

Appendixes

(These Appendixes are not a part of IEEE Std 1074-1991, IEEE Standard for Developing Life Cycle Processes, but are included for information only.)

Appendix A
Mapping Software Life Cycle Processes to Various Examples of Software Life Cycles

This Appendix demonstrates the mappings of the Activities in this standard to four different software life cycles (SLCs). This Appendix is not intended to be comprehensive. Many other SLCs are possible, for example, small development, quick reaction, and the spiral model. The SLCs presented here are examples only, and the user of this document is not required to select any of these SLCs.

Table A1 demonstrates a mapping of Activities to an eight-phase SLC.

Table A2 demonstrates a mapping of Activities to a five-phase SLC.

Table A3 demonstrates a mapping of Activities to an SLC that uses prototyping to establish requirements and design.

Table A4 demonstrates a mapping of Activities to an SLC that includes a theoretical, highly automated software development mode.

Section 1.5, "Use of This Standard," provides background information necessary for the successful understanding and application of this material. It should be read prior to proceeding further in these Appendixes.

Table A1
Software Life Cycle Example Based on Eight Phases

Concept Exploration (CE)
Design (DE)
Test (TE)
Operation and Maintenance (OM)
Requirements (RQ)
Implementation (IM)
Installation and Checkout (IN)
Retirement (RT)

Activities	CE	RQ	DE	IM	TE	IN	OM	RT
SOFTWARE LIFE CYCLE PROCESS								
Identify Candidate SLC Models	X							
Select Project Model	X							
PROJECT MANAGEMENT PROCESSES								
Project Initiation Process								
Map Activities to SLC Model	X							
Allocate Project Resources	X	X	X	X	X	X	X	X
Establish Project Environment	X	X	X					
Plan Project Management	X	X						
Project Monitoring and Control Process								
Analyze Risks	X	X	X	X	X	X		
Perform Contingency Planning		X	X	X	X	X		
Manage the Project	X	X	X	X	X	X	X	X
Retain Records	X	X	X	X	X	X	X	X
Implement Problem Reporting System		X	X	X	X	X	X	X
Software Quality Management Process								
Plan Software Quality Management	X	X	X					
Define Metrics		X	X					
Manage Software Quality	X	X	X	X	X	X	X	X
Identify Quality Improvement Needs	X	X	X	X	X	X	X	X
PRE-DEVELOPMENT PROCESSES								
Concept Exploration Process								
Identify Ideas or Needs	X							
Formulate Potential Approaches	X	X						
Conduct Feasibility Studies	X	X	X	X	X	X	X	X
Plan System Transition (If Applicable)	X	X						X
Refine and Finalize the Idea or Need		X						
System Allocation Process								
Analyze Functions		X	X					
Develop System Architecture		X	X					
Decompose System Requirements		X						

Table A1
Software Life Cycle Example Based on Eight Phases *(Continued)*

Activities	CE	RQ	DE	IM	TE	IN	OM	RT
DEVELOPMENT PROCESSES								
Requirements Process								
Define and Develop Software Requirements		X	X					
Define Interface Requirements		X	X					
Prioritize and Integrate Software Requirements		X	X					
Design Process								
Perform Architectural Design			X					
Design Data Base (If Applicable)			X					
Design Interfaces			X					
Select or Develop Algorithms		X	X					
Perform Detailed Design			X					
Implementation Process								
Create Test Data			X	X				
Create Source			X	X				
Generate Object Code			X	X				
Create Operating Documentation			X	X				
Plan Integration			X					
Perform Integration				X	X			
POST-DEVELOPMENT PROCESSES								
Installation Process								
Plan Installation					X			
Distribute Software						X		
Install Software						X		
Accept Software in Operational Environment						X		
Operation and Support Process								
Operate the System							X	
Provide Tech. Asst. & Consult.							X	
Maintain Support Request Log							X	
Maintenance Process								
Reapply Software Life Cycle							X	
Retirement Process								
Notify User							X	X
Conduct Parallel Operations (If Applicable)								X
Retire System								X

Table A1
Software Life Cycle Example Based on Eight Phases *(Continued)*

Activities	CE	RQ	DE	IM	TE	IN	OM	RT
INTEGRAL PROCESSES								
Verification and Validation Process								
Plan Verification and Validation	X	X	X					
Execute V&V Tasks	X	X	X	X	X	X	X	X
Collect and Analyze Metric Data		X	X	X	X	X	X	X
Plan Testing		X	X	X				
Develop Test Requirements			X	X				
Execute the Tests				X	X	X		
Software Configuration Management Process								
Plan Configuration Management		X	X					
Perform Configuration Identification		X	X	X	X			
Perform Configuration Control			X	X	X	X	X	X
Perform Status Accounting			X	X	X	X	X	X
Documentation Development Process								
Plan Documentation		X	X					
Implement Documentation		X	X					
Produce and Distribute Documentation				X	X			
Training Process								
Plan the Training Program		X	X					
Develop Training Materials			X	X	X			
Validate the Training Program					X	X		
Implement the Training Program						X		

Table A2
Software Life Cycle Example Based on Five Phases

Project Initiation (PI) Concept Development (CD)
Definition and Design (DD) System Development (SD)
Installation and Operation (IO)

Activities	PI	CD	DD	SD	IO
SOFTWARE LIFE CYCLE PROCESS					
Identify Candidate SLC Models	X				
Select Project Model	X				
PROJECT MANAGEMENT PROCESSES					
Project Initiation Process					
Map Activities to SLC Model	X				
Allocate Project Resources	X	X	X	X	X
Establish Project Environment	X	X	X		
Plan Project Management	X	X			
Project Monitoring and Control Process					
Analyze Risks	X	X	X	X	X
Perform Contingency Planning		X	X	X	X
Manage the Project	X	X	X	X	X
Retain Records	X	X	X	X	X
Implement Problem Reporting System		X	X	X	X
Software Quality Management Process					
Plan Software Quality Management	X	X	X		
Define Metrics		X	X		
Manage Software Quality	X	X	X	X	X
Identify Quality Improvement Needs	X	X	X	X	X
PRE-DEVELOPMENT PROCESSES					
Concept Exploration Process					
Identify Ideas or Needs	X				
Formulate Potential Approaches	X	X			
Conduct Feasibility Studies	X	X			
Plan System Transition (If Applicable)	X	X			
Refine and Finalize the Idea or Need		X			
System Allocation Process					
Analyze Functions		X	X		
Develop System Architecture		X	X		
Decompose System Requirements		X			

Table A2
Software Life Cycle Example Based on Five Phases *(Continued)*

Activities	PI	CD	DD	SD	IO
DEVELOPMENT PROCESSES					
Requirements Process					
Define and Develop Software Requirements		X	X		
Define Interface Requirements		X	X		
Prioritize and Integrate Software Requirements		X	X		
Design Process					
Perform Architectural Design			X		
Design Data Base (If Applicable)			X		
Design Interfaces			X		
Select or Develop Algorithms		X	X		
Perform Detailed Design			X		
Implementation Process					
Create Test Data			X	X	
Create Source			X	X	
Generate Object Code			X	X	
Create Operating Documentation			X	X	
Plan Integration			X		
Perform Integration				X	
POST-DEVELOPMENT PROCESSES					
Installation Process					
Plan Installation				X	
Distribute Software					X
Install Software					X
Accept Software in Operational Environment					X
Operation and Support Process					
Operate the System					X
Provide Tech. Asst. & Consult.					X
Maintain Support Request Log					X
Maintenance Process					
Reapply Software Life Cycle					X
Retirement Process					
Notify User					X
Conduct Parallel Operations (If Applicable)					X
Retire System					X

Table A2
Software Life Cycle Example Based on Five Phases *(Continued)*

Activities	PI	CD	DD	SD	IO
INTEGRAL PROCESSES					
Verification and Validation Process					
Plan Verification and Validation	X	X	X		
Execute V&V Tasks		X	X	X	X
Collect and Analyze Metric Data		X	X	X	X
Plan Testing			X	X	
Develop Test Requirements			X	X	
Execute the Tests			X	X	X
Software Configuration Management Process					
Plan Configuration Management		X	X		
Perform Configuration Identification		X	X	X	
Perform Configuration Control			X	X	X
Perform Status Accounting			X	X	X
Documentation Development Process					
Plan Documentation		X	X		
Implement Documentation			X	X	
Produce and Distribute Documentation				X	X
Training Process					
Plan the Training Program		X	X		
Develop Training Materials			X	X	
Validate the Training Program				X	X
Implement the Training Program					X

Table A3
Software Life Cycle Example Based on Prototyping

Concept Exploration (CE)
Implementation (IM)
Installation and Checkout (IN)
Retirement (RT)
Prototyping (PT)
Test (TE)
Operation and Maintenance (OM)

Activities	CE	PT	IM	TE	IN	OM	RT
SOFTWARE LIFE CYCLE PROCESS							
Identify Candidate SLC Models	X						
Select Project Model	X						
PROJECT MANAGEMENT PROCESSES							
Project Initiation Process							
Map Activities to SLC Model	X						
Allocate Project Resources	X	X	X	X	X	X	X
Establish Project Environment	X	X					
Plan Project Management	X	X					
Project Monitoring and Control Process							
Analyze Risks	X	X	X	X	X		
Perform Contingency Planning		X	X	X	X		
Manage the Project	X	X	X	X	X	X	X
Retain Records	X	X	X	X	X	X	X
Implement Problem Reporting System		X	X	X	X	X	X
Software Quality Management Process							
Plan Software Quality Management		X					
Define Metrics		X					
Manage Software Quality	X	X	X	X	X	X	X
Identify Quality Improvement Needs	X	X	X	X	X	X	X
PRE-DEVELOPMENT PROCESSES							
Concept Exploration Process							
Identify Ideas or Needs	X						
Formulate Potential Approaches	X	X					
Conduct Feasibility Studies	X	X					
Plan System Transition (If Applicable)	X	X					X
Refine and Finalize the Idea or Need		X					
System Allocation Process							
Analyze Functions		X					
Develop System Architecture		X					
Decompose System Requirements		X					

Table A3
Software Life Cycle Example Based on Prototyping *(Continued)*

Activities	CE	PT	IM	TE	IN	OM	RT
DEVELOPMENT PROCESSES							
Requirements Process							
Define and Develop Software Requirements		X					
Define Interface Requirements		X					
Prioritize and Integrate Software Rqmts.		X					
Design Process							
Perform Architectural Design		X					
Design Data Base (If Applicable)		X					
Design Interfaces		X					
Select or Develop Algorithms		X					
Perform Detailed Design		X					
Implementation Process							
Create Test Data		X	X				
Create Source		X	X				
Generate Object Code		X	X	X			
Create Operating Documentation		X	X				
Plan Integration		X					
Perform Integration			X	X			
POST-DEVELOPMENT PROCESSES							
Installation Process							
Plan Installation				X			
Distribute Software					X		
Install Software					X		
Check out Software in Operational Envnmt.					X		
Operation and Support Process							
Operate the System						X	
Provide Tech. Asst. & Consult.						X	
Maintain Support Request Log						X	
Maintenance Process							
Reapply Software Life Cycle						X	

Table A3
Software Life Cycle Example Based on Prototyping *(Continued)*

Activities	CE	PT	IM	TE	IN	OM	RT
Retirement Process							
Notify User						X	X
Conduct Parallel Operations (If Applicable)							X
Retire System							X
INTEGRAL PROCESSES							
Verification and Validation Process							
Plan Verification and Validation		X					
Execute V&V Tasks		X	X	X	X	X	X
Collect and Analyze Metric Data		X	X	X	X	X	X
Plan Testing		X	X				
Develop Test Specifications		X	X				
Execute the Tests			X	X	X		
Software Configuration Management Process							
Plan Configuration Management		X					
Perform Configuration Identification		X	X	X			
Perform Configuration Control		X	X	X	X	X	X
Perform Status Accounting		X	X	X	X	X	X
Documentation Development Process							
Plan Documentation		X					
Implement Documentation		X	X				
Produce and Distribute Documentation				X	X		
Training Process							
Plan the Training Program		X					
Develop Training Materials		X	X	X			
Validate the Training Program				X	X		
Implement the Training Program					X		

Table A4
Software Life Cycle Example Based on an Operational Specification*

System Requirements (SR) Operational Specification (OS)
Transformed Specification (TS) Delivered System (DS)

Activities	SR	OS	TS	DS
SOFTWARE LIFE CYCLE PROCESS				
Identify Candidate SLC Models	X			
Select Project Model	X			
PROJECT MANAGEMENT PROCESSES				
Project Initiation Process				
Map Activities to SLC Model	X			
Allocate Project Resources	X	X	X	X
Establish Project Environment	X	X		
Plan Project Management	X			
Project Monitoring and Control Process				
Analyze Risks	X	X	X	X
Perform Contingency Planning	X	X	X	X
Manage the Project	X	X	X	X
Retain Records	X	X	X	X
Implement Problem Reporting System		X	X	X
Software Quality Management Process				
Plan Software Quality Management	X	X		
Define Metrics	X	X		
Manage Software Quality	X	X	X	X
Identify Quality Improvement Needs	X	X	X	X
PRE-DEVELOPMENT PROCESSES				
Concept Exploration Process				
Identify Ideas or Needs	X			
Formulate Potential Approaches	X			
Conduct Feasibility Studies	X	X		
Plan System Transition (If Applicable)	X			X
Refine and Finalize the Idea or Need	X			
System Allocation Process				
Analyze Functions	X	X		
Develop System Architecture	X	X		
Decompose System Requirements	X			

* As defined in the IEEE Tutorial, "New Paradigms for Software Development," by William W. Agresti.

Table A4
Software Life Cycle Example Based on an Operational Specification *(Continued)*

Activities	SR	OS	TS	DS
DEVELOPMENT PROCESSES				
Requirements Process				
Define and Develop Software Requirements	X	X		
Define Interface Requirements	X	X		
Prioritize and Integrate Software Requirements	X	X		
Design Process				
Perform Architectural Design		X		
Design Data Base (If Applicable)		X		
Design Interfaces		X		
Select or Develop Algorithms	X	X		
Perform Detailed Design		X		
Implementation Process				
Create Test Data		X	X	
Create Source		X	X	
Generate Object Code		X	X	
Create Operating Documentation		X	X	
Plan Integration		X		
Perform Integration			X	
POST-DEVELOPMENT PROCESSES				
Installation Process				
Plan Installation			X	
Distribute Software				X
Install Software				X
Check out Software in Operational Environment				X
Operation and Support Process				
Operate the System				X
Provide Tech. Asst. & Consult.				X
Maintain Support Request Log				X
Maintenance Process				
Reapply Software Life Cycle				X
Retirement Process				
Notify User				X
Conduct Parallel Operations (If Applicable)				X
Retire System				X

Table A4
Software Life Cycle Example Based on an Operational Specification *(Continued)*

Activities	SR	OS	TS	DS
INTEGRAL PROCESSES				
Verification and Validation Process				
Plan Verification and Validation	X	X		
Execute V&V Tasks	X	X	X	X
Collect and Analyze Metric Data	X	X	X	X
Plan Testing		X	X	
Develop Test Specifications		X	X	
Execute the Tests			X	X
Software Configuration Management Process				
Plan Configuration Management	X	X		
Perform Configuration Identification	X	X	X	
Perform Configuration Control		X	X	X
Perform Status Accounting		X	X	X
Documentation Development Process				
Plan Documentation	X	X		
Implement Documentation		X	X	
Produce and Distribute Documentation			X	X
Training Process				
Plan the Training Program	X	X		
Develop Training Materials		X	X	
Validate the Training Program			X	X
Implement the Training Program				X

Appendix B
Software Project Management Tailoring Template

The Software Project Management Tailoring Template is designed to assist project managers in identifying project-critical deliverables and assuring their completion as needed.

This template may be used to assist in the project-specific mapping of information into the required project documentation.

Table B1
Software Project Management Tailoring Template

SOFTWARE PROJECT MANAGEMENT TAILORING TEMPLATE			
Process or Activity Name	Section	Output Information	Mapped Deliverables or Activities
PROCESS GROUP **Process** Activity		Required Output Information	
SOFTWARE LIFE CYCLE PROCESS	2		
Identify Candidate SLC Models	2.3	Candidate SLC Model(s)	
Select Project Model	2.4	Selected SLC Model	
PROJECT MANAGEMENT PROCESSES	3		
Project Initiation	3.1		
Map Activities to SLC Model	3.1.3	Software Life Cycle	
		List of Activities Not Used	
Allocate Project Information	3.1.4	Resource Allocations	
Establish Project Environment	3.1.5	Project Environment	
Plan Project Management	3.1.6	Problem Reporting & Resolution Planned Info.	
		Retirement Planned Info.	
		Software Project Management Planned Info.	
		Support Planned Info.	
Project Monitoring and Control	3.2		
Analyze Risks	3.2.3	Analysis of Risks	
Perform Contingency Planning	3.2.4	Contingency Planned Info.	
Manage the Project	3.2.5	Project Management Reported Info.	
		Anomalies	
Retain Records	3.2.6	Historical Project Records	
Implement Problem Reporting Method	3.2.7	Resolved Problem Reported Info.	
		Report Log	
		Enhancement Problem Reported Info.	
		Corrections Problem Reported Info.	

Table B1
Software Project Management Tailoring Template
(Continued)

SOFTWARE PROJECT MANAGEMENT TAILORING TEMPLATE			
Process or Activity Name	Section	Output Information	Mapped Deliverables or Activities
Software Quality Management	3.3		
Plan Software Quality Management	3.3.3	Software Quality Management Planned Info.	
Define Metrics	3.3.4	Defined Metrics	
		Collection and Analysis Methods	
Manage Software Quality	3.3.5	Project Quality Assessments	
Identify Quality Improvement Needs	3.3.6	Quality Improvement Recommendations	
PRE-DEVELOPMENT PROCESSES	**4**		
Concept Exploration	**4.1**		
Identify Ideas or Needs	4.1.3	Prelim. Statement of Need	
Formulate Potential Approaches	4.1.4	Constraints and Benefits	
		Potential Approaches	
Conduct Feasibility Studies	4.1.5	Recommendations	
Plan System Transition (If Applicable)	4.1.6	Transition Impact Statement	
		Transition Planned Info.	
Refine and Finalize the Idea or Need	4.1.7	Statement of Need	
System Allocation	**4.2**		
Analyze Functions	4.2.3	Funct'l Description of System	
Develop System Architecture	4.2.4	System Architecture	
Decompose System Requirements	4.2.5	Funct'l Hardware Rqmts.	
		Funct'l Software Rqmts.	
		System Interface Requirements (If Applicable)	
DEVELOPMENT PROCESSES	**5**		
Requirements	**5.1**		
Define and Develop Software Rqmts.	5.1.3	Prelim. Software Rqmts.	
		Installation Rqmts.	
Define Interface Requirements	5.1.4	Software Interface Rqmts.	
Prioritize and Integrate Software Rqmts.	5.1.5	Software Rqmts.	

Table B1
Software Project Management Tailoring Template
(Continued)

SOFTWARE PROJECT MANAGEMENT TAILORING TEMPLATE			
Process or Activity Name	Section	Output Information	Mapped Deliverables or Activities
Design	**5.2**		
Perform Architectural Design	5.2.3	Software Architectural Design Description	
Design Data Base (If Applicable)	5.2.4	Data Base Description	
Design Interfaces	5.2.5	Interface Description	
Select or Develop Algorithms	5.2.6	Algorithm Descriptions	
Perform Detailed Design	5.2.7	Software Design Description	
Implementation	**5.3**		
Create Test Data	5.3.3	Stubs and Drivers (If Applicable)	
		Test Data	
Create Source	5.3.4	Data Base (If Applicable)	
		Source Code	
Generate Object Code	5.3.5	Corrected Data Base (If Applicable)	
		Corrected Source Code	
		Object Code	
Create Operating Documentation	5.3.6	Operating Documentation	
Plan Integration	5.3.7	Integration Planned Info.	
Perform Integration	5.3.8	Integrated Software	
POST-DEVELOPMENT PROCESSES	**6**		
Installation	**6.1**		
Plan Installation	6.1.3	Software Installation Planned Info.	
Distribute Software	6.1.4	Packaged Operating Docs.	
		Packaged Software	
		Packaged Installation Planned Info.	
Install Software	6.1.5	Installation Reported Info.	
		Installed Software	
Accept Software in Operational Envrmt.	6.1.6	Installed Software System	
		Customer Acceptance	

Table B1
Software Project Management Tailoring Template
(Continued)

SOFTWARE PROJECT MANAGEMENT TAILORING TEMPLATE			
Process or Activity Name	Section	Output Information	Mapped Deliverables or Activities
Operation and Support	**6.2**		
Operate the System	6.2.3	System Anomalies	
		Operation Logs	
		Feedback Data	
Provide Tech. Asst. & Consult.	6.2.4	Support Response	
Maintain Support Request Log	6.2.5	Anomalies	
		Support Request Log	
Maintenance	**6.3**		
Reapply Software Life Cycle	6.3.3	Maintenance Recommendations	
Retirement	**6.4**		
Notify User	6.4.3	Official Notification	
Conduct Parallel Operations	6.4.4	Parallel Operations Log	
Retire System	6.4.5	Post-Operation Review Reported Information	
		Archive Reported Info.	
INTEGRAL PROCESSES	**7**		
Verification and Validation	**7.1**		
Plan V&V	7.1.3	Software V&V Planned Info.	
Execute V&V Tasks	7.1.4	Evaluation Reported Info.	
		Anomalies	
Collect and Analyze Metric Data	7.1.5	Analysis Reported Info.	
Plan Testing	7.1.6	Test Planned Info.	
Develop Test Requirements	7.1.7	Test Requirements	
Execute the Tests	7.1.8	Test Summary Reported Info.	
		Tested Software	
		Anomalies	

Table B1
Software Project Management Tailoring Template
(Continued)

SOFTWARE PROJECT MANAGEMENT TAILORING TEMPLATE			
Process or Activity Name	Section	Output Information	Mapped Deliverables or Activities
Software Configuration Management	**7.2**		
Plan Configuration Management	7.2.3	Software Config. Management Planned Info.	
Develop Configuration Identification	7.2.4	Config. Identification	
Perform Configuration Control	7.2.5	Change Status Controlled Item	
Perform Status Accounting	7.2.6	Status Reported Info.	
Documentation Development	**7.3**		
Plan Documentation	7.3.3	Documentation Planned Info.	
Implement Documentation	7.3.4	Document	
Produce and Distribute Documentation	7.3.5	Published Document	
Training	**7.4**		
Plan the Training Program	7.4.3	Training Planned Info.	
Develop Training Materials	7.4.4	Training Manual	
		Training Materials	
Validate the Training Program	7.4.5	Updated Training Manual	
		Updated Training Materials	
		Training Feedback	
Implement the Training Program	7.4.6	Trained Personnel	
		Training Feedback	
		Updated Skills Inventory	

Appendix C
Process Interrelationships

Figure C1 in this Appendix shows the interrelationships between the Processes. The boxes in Fig C1 are Processes; the directed lines show the flow of input and output information between Processes. The Management Processes are grouped in the center of the figure, with the Development-Oriented Processes arranged around them.

This figure does not attempt to show Process invocations. All the directed lines to and from the Invoked Processes represent explicit information passed between those Processes.

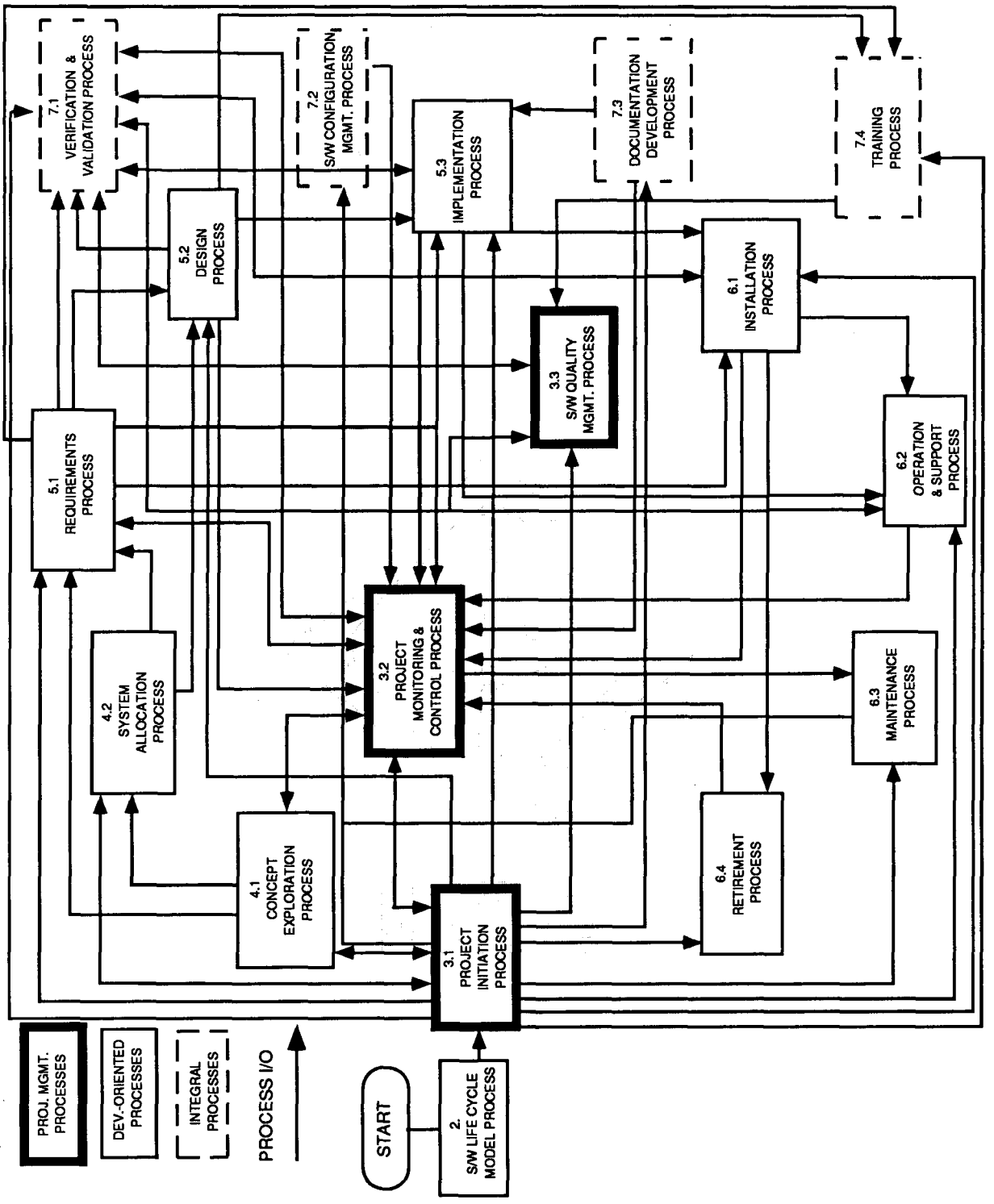
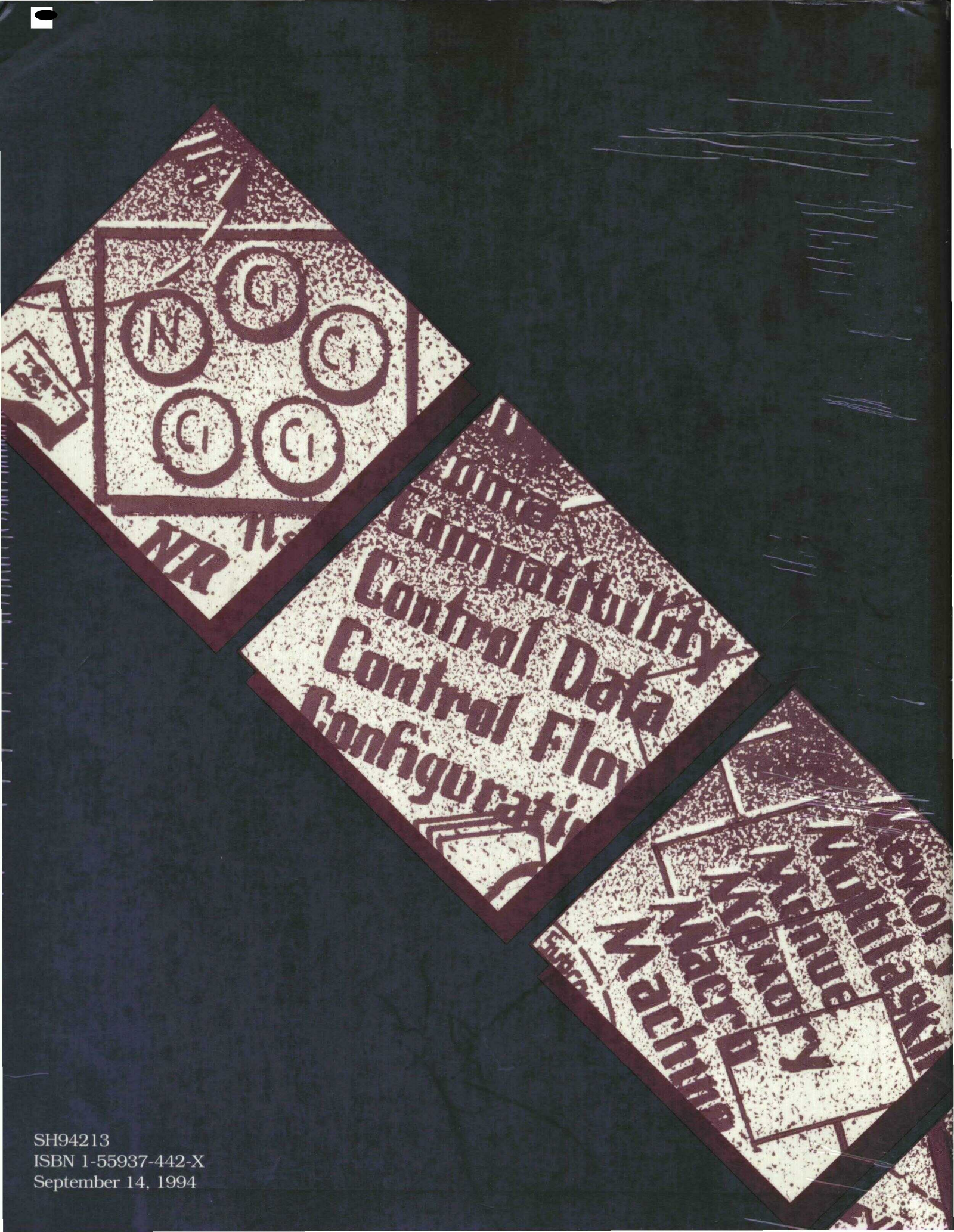


Fig C1
Process Interrelationships

Index



SH94213
ISBN 1-55937-442-X
September 14, 1994