10/19/95

SANDIA REPORT

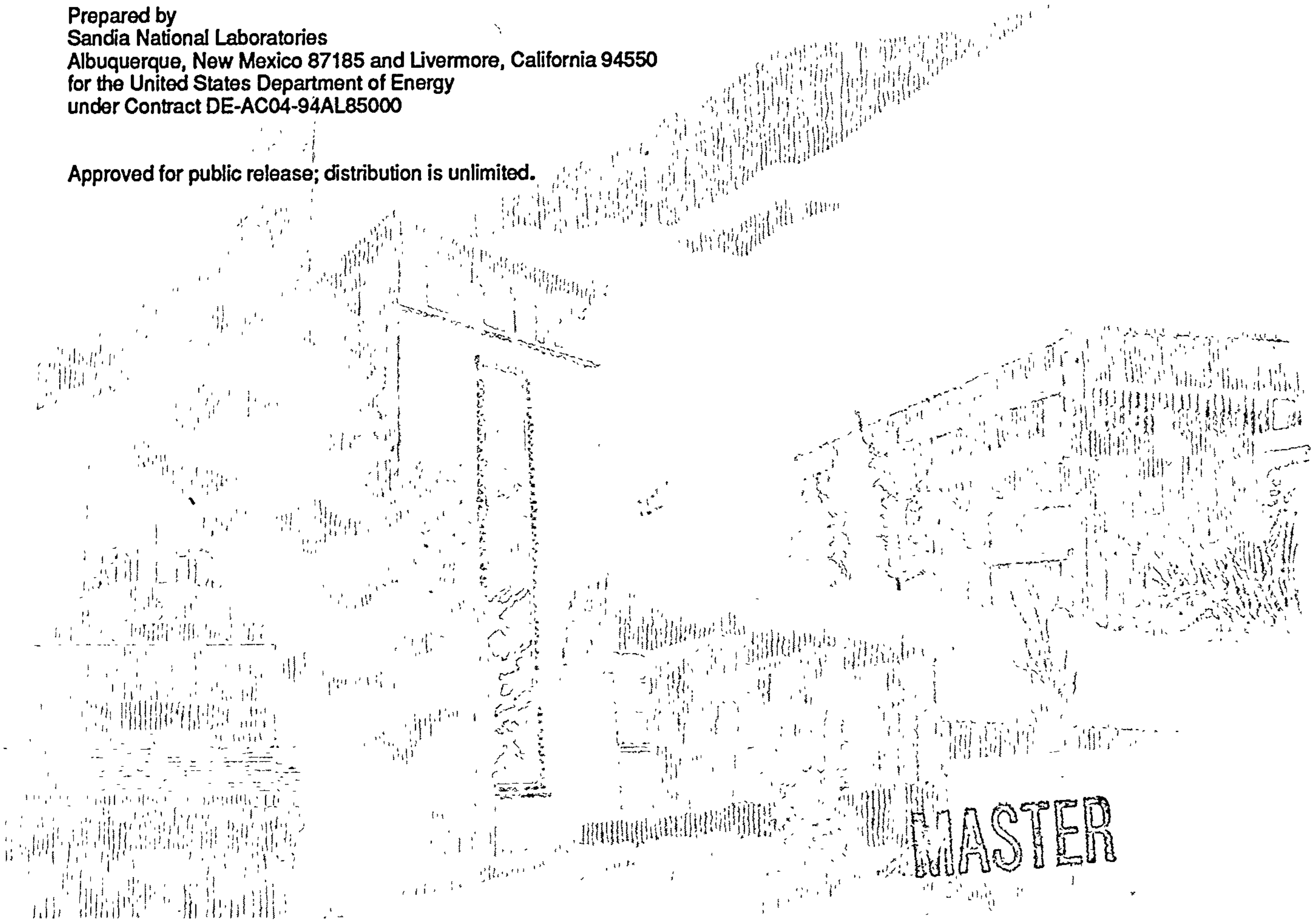
SAND85–2345 • UC–705
Unlimited Release
Printed September 1995

Sandia Software Guidelines

Volume 2
Documentation

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Approved for public release; distribution is unlimited.



DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

ar

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A07
Microfiche copy: A01

Distribution
Category UC-705

SAND85-2345
Unlimited Release
Printed September 1995

Sandia Software Guidelines

Volume 2
Documentation

Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

This volume is one in a series of *Sandia Software Guidelines* intended for use in producing quality software within Sandia National Laboratories. In consonance with the IEEE Standards for software documentation, this volume provides guidance in the selection of an adequate document set for a software project and example formats for many types of software documentation. A tutorial on life cycle documentation is also provided. Extended document thematic outlines and working examples of software documents are available on electronic media as an extension of this volume.

Blank Page

Foreword

This volume is one in a series of *Sandia Software Guidelines* for use in producing quality software within Sandia National Laboratories (SNL). These guidelines, when used in conjunction with IEEE standards and current software engineering methodologies, will help ensure that software developed within SNL is usable, reliable, understandable, maintainable, and portable. The series consists of the following documents:

- Volume 1, ***Software Quality Planning*** (SAND85-2344)

  Presents an overview of procedures designed to ensure software quality. Includes a sample software quality assurance plan for a generic Sandia software project.

- Volume 2, ***Documentation*** (SAND85-2345)

  Presents guidance in the selection of adequate documentation for a software project. Includes templates of documents needed for developing, maintaining, and defining software projects.

- Volume 3, ***Standards, Practices, and Conventions*** (SAND85-2346)

  Presents consensus standards and practices for developing and maintaining quality software at SNL. Includes recommended deliverables for major phases of the software life cycle.

- Volume 4, ***Configuration Management*** (SAND85-2347)

  Presents a discussion of configuration management objectives and approaches throughout the software life cycle for software projects at SNL.

- Volume 5, ***Tools, Techniques, and Methodologies*** (SAND85-2348)

  Presents descriptions and a directory of software tools and methodologies available to SNL personnel.

Acknowledgment

A consensus document, like this volume of the *Sandia Software Guidelines*, cannot be produced without the cooperation and hard work of a great many people throughout the Laboratories. The sponsoring Quality Engineering Department wishes to thank the members of the working group who contributed to this volume, as well as members of the balloting group who reviewed and refined it.

Working Group Members

Marianna Eisenhour	(02615)	Gerald McDonald	(12326)
Gregg Giesler	(06521)	Dave Peercy, Editor	(12326)
Pam Harris	(09617)	Joe Schofield	(13316)
Walt Huebner	(09421)	Eric Tomlin	(06613)
Dwayne Knirk	(12326)	Fred Trussell	(12334)

Balloting Members

Paul Attermeier	(09426)	Ann Hodges	(09432)
Lorraine Baca	(02615)	Laney Kidd	(02615)
Michael Blackledge	(12326)	Charleene Lennox	(02172)
Drayton Boozer	(12400)	Paula McAllister	(07901)
Ann Chipman	(05501)	Margaret Olson	(02122)
Nancy Freshour	(07901)	Patty Trellue	(02615)

The editor of this report also wishes to recognize this quote by John Aitken, a software manager and friend, who helped a few of us recognize (perhaps inadvertently) a basic truth about documentation:

> "Whenever there is a difference between source code and documentation, always believe the code -- unless it's wrong."

Table of Contents

List of Figures

List of Tables

1 Introduction

Software documentation is all the information that describes software processes and products. This volume presents guidelines on the use and form of software documentation that should be helpful to many different Sandia software projects. The full information on references[1] is contained in Appendix A.

Documentation is the time-dependent representation of software products as they evolve through life cycle activities: concept exploration, requirements analysis, design, implementation, test, operation, maintenance support, and retirement. Documentation is also the definition of those processes used to manage, develop, use, and support software. Documentation may exist on a variety of physical media, including hard copy paper and electronic media. Documentation may also exist in a variety of physical formats, including textual, graphical, and other combinations of information language forms, such as data bases and timing state relations. The software program (binary executable code and required data) is typically excluded from documentation but is in fact the most important representation of the software. The software program is sometimes denoted as the software [deliverable] product, and other supporting information as the software product associated documentation. The processes to create documentation may be manual or automated. Software is the broad umbrella under which the pieces of documentation and executable program code are collected. This interpretation of software and documentation is somewhat broader than definitions such as in the Institute of Electrical and Electronics Engineers (IEEE) Standard Glossary of Software Engineering Terminology [IEEE610]:

> "*Software : the computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.*"
>
> "*Documentation: any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results.*"

Within this volume documentation, documents, and document set will be differentiated as necessary. Documentation is the total set of all information about software processes and products. When certain software documentation is developed and organized in accordance with a predefined purpose and format and is a reasonably complete product entity under some level of official configuration control, it will be called a document. As with documentation in general, a document may be stored on a variety of media and may be represented in a variety of formats, but is most often associated with textual and graphical forms. Documents are more formal than just a written record and their intent is to capture the information considered the most important for description of the software. A document set is a collection of software documents that results from a software project. The documentation term will generally be used throughout this volume unless specific documents or document sets are being referenced, or the context makes more specific terminology useful.

1.1 Intent

This Sandia Software Guidelines volume provides guidance for selecting a software document set for a project; includes discussions of the content and outlines of several key software documents; and presents examples of software documentation to assist individuals and Sandia projects in the context of their involvement with the acquisition, development, use, and support of software. These guidelines can be flexibly tailored for use on most Sandia projects: small, large, research, information systems, weapon-related, energy and environment, and work for others.

[1] References throughout this volume are identified by a descriptive group of characters enclosed in brackets, such as [IEEE610] for IEEE Std 610.

1.2 Environment

The environment found throughout Sandia is one of innovation and change. Whether research or production-oriented design and development, Sandia projects are forging new state-of-the-art frontiers. Change and iteration are day-to-day parts of doing business. Software is a major part of this innovation and change.

The principles and techniques necessary to transform software activities from a creative art to a science are also in a state of rapid evolution. Since documents are intended to capture the important snapshots of the software life cycle, evolutionary changes to the software activities dictate the need for documentation methods that can be easily adapted to and coordinated with the changes to the software process activities.

Each Sandia project should document its software management, engineering, quality, operation, and support activities while balancing other factors such as performance, cost, complexity, quality, schedule, and customer expectations. The documentation appropriate for each project should be described in the project plan or an applicable organization software management plan as required by Sandia Laboratory Policy [SLP 1011]. This volume will provide Sandia line management organizations and functional program organizations with the necessary insight to make informed project documentation choices (see Chapter 3):

- what documentation is appropriate for a project;
- what documentation is appropriate for each software life cycle activity of a project;
- why the selected documentation is important to the project;
- how to control updates and changes to documentation during development and support;
- what automated methods are available to support documentation - and their limitations;
- how to improve project documentation quality.

1.3 Applicability

The material presented in this volume is intended to be applicable to software projects at Sandia. Most projects at Sandia can be regarded as consisting of either weapon-related or non-weapon-related activities. The former class of projects, termed War Reserve (WR), are undertaken in support of the design and development of nuclear weapon systems. We will refer to the latter class of projects as non-WR.

Within these two primary classes, WR and non-WR, there may be large or small projects, real-time and non real-time projects, projects that include commercial as well as internally developed software, research projects, information system projects, and projects that require customer-specific document sets. Where differences in documentation approach exist, examples are used to illustrate the differences as much as possible.

This volume is a set of guidelines, not directives. Enforcement of an application of these guidelines should be established at the organization level appropriate for the projects to which the guidelines are to be applied. This document is to be used with the other volumes of the Sandia Software Guidelines, references [SSGv1], [SSGv3], [SSGv4], and [SSGv5]. In addition, there are many other references such as the Sandia Laboratories Policy on Software Management [SLP1011], Sandia Preferred Processes for Software Development [PPSD], and Process Guidelines for Sandia WR Software Development [PGWR] that provide further details on software engineering guidelines for Sandia National Laboratories. The reader is also encouraged to consult the IEEE software engineering standards and guides [IEEEstds] for further information since many of the document outlines/templates provided within are derived from the IEEE standards.

1.4 Organization

This volume is divided into five chapters, five appendices, and continually evolving information on electronic media such as documentation templates and examples in various word processing formats. The chapters and appendices are bound in this physical volume.

Chapter 2 contains a description of some current and future software documentation concepts. An overview of documentation relationships across the software life cycle activities is presented with a more detailed tutorial included in Appendix C. Documentation of software processes and products as part of the Software Engineering Institute's concept of software process improvement is discussed. The potential use of information networks such as the National Information Infrastructure, World Wide Web, and the Continuous Acquisition and Logistics Support are described along with applications within Sandia for software documentation sharing at the corporate, organization, and project level.

Chapter 3 covers the practical aspects of making project documentation choices. A method for choosing the appropriate documentation and tailoring the documentation to the project is discussed. Application of the selection method to different example projects is illustrated in Appendix D. Choices for controlling the configuration of software documentation throughout the life cycle (development and support) are discussed. Tools to construct, update, and control software documentation are discussed along with some of the problems.. Options are presented for the documentation of existing software as part of reverse engineering and re-engineering activities. And, characteristics of better quality documentation are discussed along with some methods for building these characteristics into the documentation throughout the software life cycle activities.

Chapter 4 contains descriptions of a core set of key software documents. The descriptions include an outline for each key document and guidelines for tailoring and completing each outline. Documents are provided for software management and engineering activities. The complete document template for each outline is available on electronic media. For the most part, the document templates are similar to IEEE standards and can be tailored to most organization-specific documentation requirements. IEEE, Department of Defense (DoD), and International Standards Organization (ISO) standards related to documentation are briefly described in Appendix E along with points of contact and current access methods.

Chapter 5 provides specific focus on documenting software implementation at the unit level, in particular the source code and test activities. Use of a Software Development Folder to capture implementation documentation is described.

The appendices include a list of references; glossary of acronyms; tutorial on life cycle documentation relationships; examples of project document set selections; and documentation standards and contacts.

In addition to this hard copy volume, the templates described in Chapter 4 and a number of document examples (not always in a format identical to the associated template) are available on electronic media, generally in the original word processor format. For additional information on how to access the latest version of the electronic media, call the telephone listing for "Software Guidelines" under the services section in the Sandia National Laboratories telephone directory. The electronic media will be regularly updated to include revised and new templates, new examples of documentation, and other relevant information on documentation.

1.5 How to Use This Volume

A reader guide for locating information concerning various software documentation interests is summarized in Figure 1-1.

Concern	Read First	Read Second	Read Third
What's in this volume?	Chapter 1: Introduction	Review Table of Contents & Illustrations	Pursue interesting topics
What software documentation templates are available?	Chapter 4: Document Guidelines - format & content - electronic copy of templates & examples	Chapter 1: Introduction - section 1.4 last paragraph	Appendix C: Tutorial on Life Cycle Relationships - documentation forms
What software documentation is appropriate for my project?	Chapter 3: Making Documentation Choices - section 3.1 selection	Appendix D: Project Selection Examples	Chapter 2: Documentation Concept - section 2.1 life cycle
What if my code is already written -- how can existing software be documented?	Chapter 3: Making Documentation Choices - section 3.2 control - section 3.4 reengineer - section 3.5 quality	Chapter 5: Implementation Documentation - code & unit test - development folder	Appendix C: Tutorial on Life Cycle Relationships - documentation forms
What are some more modern concepts for documentation?	Chapter 2: Documentation Concepts - software life cycle - process improvement - information sharing - formal methods	Appendix C: Tutorial on Life Cycle Relationships - documentation - relationships - guidelines	Appendix E: Documentation Standards - trends in standards - contacts
Are there documentation tools that might help?	Chapter 3: Making Documentation Choices - section 3.3 tools	Chapter 4: Document Guidelines - format & content - example templates in electronic form	Chapter 5: Implementation Documentation - code & unit test - development folder
How do you control documentation versions?	Chapter 3: Making Documentation Choices - section 3.2 control	Chapter 2: Documentation Concept - section 2.1 life cycle	Sandia Software Guidelines: Volume 4
How does all this documentation work together over the complete life cycle?	Appendix C: Tutorial on Life Cycle Relationships - documentation forms	Chapter 3: Making Project Documentation Choices - section 3.1 selection	Appendix D: Document Set Selection for Example Projects
How does one obtain an electronic media copy of the document templates and other documentation examples?	Chapter 1: Introduction - section 1.4 last paragraph	Sandia Telephone Directory: Software Guidelines	Sandia Telephone Directory: Software Management Program

Figure 1-1. Guide to Using the Software Documentation Volume

2 Documentation Concepts

This chapter contains discussion of some traditionally important as well as more forward-looking documentation concepts. A brief overview of software life cycle activities and traditional supporting software product documentation is presented. Recent emphasis on software process improvement and the documentation that provides evidence of process definition and capability maturity is discussed. A guide to the Software Engineering Institute (SEI) Capability Maturity Model documentation is described along with a framework for a documentation process maturity model. One near-term documentation thrust is to provide methods for storing, handling, transporting, and displaying information (including software documentation) across internal and external networks. Some of these information sharing concepts are described. Some research concepts on formal methods for documenting software are also discussed.

2.1 Overview of Software Life Cycle and Associated Documentation

This section provides an overview of the relationship between software documents and the software life cycle activities and processes that produce these documents. The top-level software life cycle activities include those shown in Figure 2-1, organized for illustration purposes only into a Waterfall-like model, where the phases correspond to each of the activities. In addition to these activities, a Concept Exploration activity is usually conducted at the system level and may involve software activities.

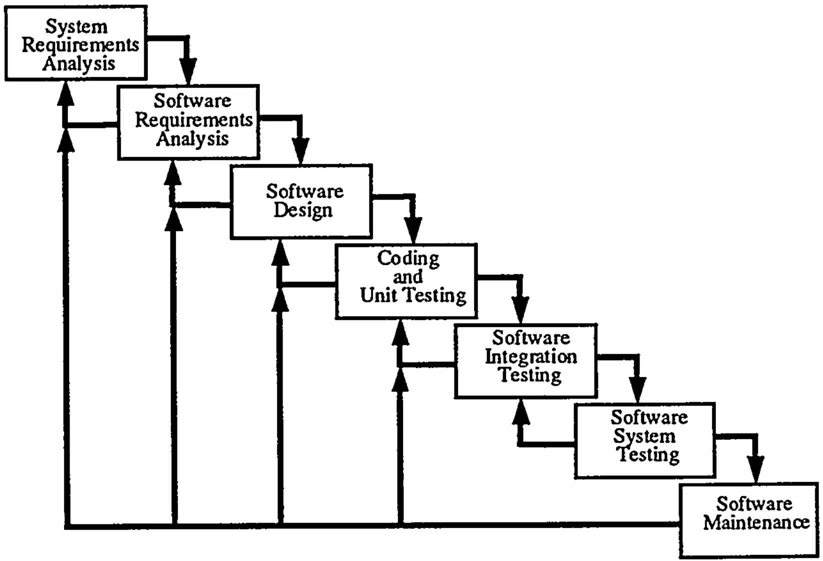

System Requirements Analysis
Software Requirements Analysis
Software Design
Coding and Unit Testing
Software Integration Testing
Software System Testing
Software Maintenance

Figure 2-1. Activities in the Standard Software Life Cycle

For each top-level activity, a set of lower-level tasks are identified that may be performed sequentially, in parallel, and/or iteratively. The Sandia Preferred Processes for Software Development [PPSD] provides a top-level description of the process tasks to accomplish software requirement analysis, software design, software coding and unit testing, and software integration testing activities. Each of these activities may involve development/support, metrics and measurement, reporting, reviews, approval, customer/user coordination, and so forth -- with the appropriate associated documentation. For example, the software requirements analysis process tasks are illustrated in Figure 2-2. A more detailed discussion is presented in Appendix C.

Tasks Identification	Tasks Description	Possible Associated Documentation
R1	Define and schedule Software Requirements Specification (SRS) activities	Updated Software Development Plan Updated Project Schedule Updated Resource Estimation SRS Thematic Outline
R2	Write SRS overview and get customer feedback	SRS Overview Updated SRS problem tracking list
R3	Prepare draft SRS	Draft SRS
R4	Schedule SRS inspection and distribute SRS	SRS Inspection Package
R5	Inspect SRS	SRS Inspection Defect Data & Action Items Updated SRS problem tracking list
R6	Resolve inspection issues	Configuration managed SRS Updated SRS problem tracking list Updated internal inspection data base Management inspection report
R7	Get customer feedback	Updated SRS problem tracking list
R8	Approve and distribute SRS	Approved and configuration managed SRS
R9	Maintain SRS	Changes to SRS through update process

Figure 2-2. Software Requirements Analysis Process Task Activities

A number of potential documents and reports that may be created during a development project or modified during software maintenance activities are shown in Figure 2-3. Documentation associated with a software project may include other elements in addition to the documents shown in Figure 2-3 and those identified in the preceding paragraphs. Such elements may include memoranda, electronic mail, results of design decisions and trade-off studies, minutes of technical interchange meetings, and so forth. The most important of these elements are further described in Appendix C.

<table>
<tr><th colspan="7">SOFTWARE DOCUMENTS RELATED TO THE STANDARD LIFE CYCLE</th></tr>
<tr><th>System Requirements Analysis</th><th>Software Requirements Analysis</th><th>Software Design</th><th>Coding and Unit Testing</th><th>Software Integration and Testing</th><th>System Testing</th><th>Software Support</th></tr>
<tr><td rowspan="4">Project Management Plan</td><td colspan="6">Software Management Plan</td></tr>
<tr><td colspan="6">Software Development (and Support) Plans</td></tr>
<tr><td colspan="6">Software Quality Assurance Plan</td></tr>
<tr><td colspan="6">Software Configuration Management Plan</td></tr>
<tr><td>System Requirements Specification</td><td>Software Requirements Specification</td><td>Software Design Description</td><td>Software Development Folders</td><td>Module Test Cases</td><td>Software System Test Plan</td><td>Software Support Plan</td></tr>
<tr><td>System Project Schedule</td><td>Interface Requirements Specification</td><td>Interface Design Document</td><td>Source Code Listings</td><td>Module Test Results</td><td>Software Test Descriptions</td><td>Software Support Manual</td></tr>
<tr><td>Software Project Schedule</td><td>Requirements Inspection Reports</td><td>Design Inspection Reports</td><td>Code Inspection Reports</td><td></td><td>Readiness Review Report</td><td>Software Users Guide</td></tr>
<tr><td rowspan="2">Software Verification & Validation Plan</td><td>Preliminary Design Review Reports</td><td>Design Review Reports</td><td></td><td></td><td>Software System Test Results</td><td>Changes to Product Baseline Documents</td></tr>
<tr><td colspan="6">Quality Assurance & Configuration Management CM Audit Reports</td></tr>
</table>

Figure 2-3. Software Products Associated With Activities and Processes in the Standard Software Life Cycle

The organized manner in which software life cycle activities, the process definitions, and the associated documentation components are integrated into project phases is the software life cycle model. This model defines the overall processes that will be used by a project and the software products and documentation that will be produced by the project. Documentation of software processes and integration of environments for information sharing are becoming key to the effective development of project documentation across the life cycle activities. For critical Sandia projects it may even be necessary to provide a more formal approach to software development and documentation to satisfy safety and security requirements. These emerging methods to support more effective software documentation are discussed in the next few subsections. Some internal Sandia projects at the organization and corporate level are beginning to address and use some of these methods.

2.2 Software Process Improvement

Software organizations can benefit from documentation management. As an organization begins to understand its essential processes, these processes will be documented, perhaps in accordance with the model of Key Process Areas such as promoted by the Software Engineering Institute (SEI). The SEI Capability Maturity Model (CMM) [SEI-CMM] identifies various documentation across 18 Key Process Areas and 5 levels of capability maturity. Documentation requirements of the SEI CMM Key Process Areas are based on the following area characteristics:

- goals;
- commitment to perform in the form of actions taken to ensure that a defined process is established and will endure;
- ability to perform in the form of preconditions that will allow implementation of the process;
- activities performed in the form of plans, roles, and procedures necessary to implement the key process;
- measurement and analysis in the form of measures and tracking the activity results relative to the defined goals and process definition; and,
- verification that the activities performed comply with the defined process.

The next subsections provide further guidance for software documentation related to the SEI CMM, the associated Key Process Areas, and the characteristics specific to each Key Process Area.

2.2.1 Software Process Capability Maturity Documentation

The documentation associated with the 18 SEI CMM Key Process Areas across the five CMM Levels is illustrated in Figure 2-4. Level 1 (Ad Hoc) is not represented since no defined processes and specific documentation are generally identifiable. Details concerning content of the Key Process Area documentation can be found in references [SEI-CMM] and [STSCdo].

Level 2 KPAs	Documentation
Requirements Management	Allocated requirements review Policy on managing allocated software requirements
Software Project Planning	Software development plan Software estimates procedures Project schedule Software life cycle definition Software planning data Software project activities and commitments Software engineering facilities and support tools plan
Software Project Tracking and Oversight	Revision of software development plan Peer review plans Change requests and problem reports procedure Formal reviews at selected milestones Project tracking and recording procedures
Software Subcontract Management	Statement of work Software subcontractor selection procedure Subcontractor software development plan Subcontract management policy Changes to subcontractor Statement of Work/Contract/Commitments Formal review of subcontractor accomplishments at selected milestones Monitoring of subcontractor CM/SQA activities procedure Formal subcontractor evaluation procedures Acceptance testing procedure for subcontractor's products
Software Quality Assurance	Software quality assurance plan Policy for implementation of SQA Deviations in software activities and software work products procedure SQA participation procedure SQA reports SQA reviews procedure
Software Configuration Management	SCM activities plan SCM policy Changes to baselines procedure Software baseline library product creation and release control procedure Configuration items/units status review procedure Standard software configuration management reports Baseline audits procedure

Figure 2-4. Repeatable Level Software Process Documentation: Part 1 of 4

Level 3 KPAs	Documentation
Organization Process Focus	Assessment finding action plan Software development, improvement, & training coordination policy Process database utilization policy New technology evaluation/transfer procedure
Organization Process Definition	Developing and maintaining standard software process and life cycle definitions procedure Software process tailoring guidelines Utilization of process database/library policy
Training Program	Development and revision of organization/project training plan Training requirements Development and maintenance of training courses Training waiver Training records
Integrated Software Management	Tailoring of standard software process procedures Software process revision procedure Costs/dependencies/risks/resources management procedure
Software Product Engineering	Methods and tools integration plan Software requirements, design, code, documentation, testing defined and integrated into software process
Intergroup Coordination	Resolution of intergroup issues procedure Intergroup commitments communication plan Critical dependencies tracking procedure Technical review and interchange policy
Peer Reviews	Peer review plan Peer review performance procedure

Figure 2-4. Defined Level Software Process Documentation: Part 2 of 4

Level 4 KPAs	Documentation
Quantitative Process Management	Quantitative process management plan Measurement and control of data procedure Quantitative control of process procedure Quantitative process management activities report Process capability baseline procedure
Software Quality Management	Software quality plan Quantitative quality goals document

Figure 2-4. Managed Level Software Process Documentation: Part 3 of 4

Level 5 KPAs	Documentation
Defect Prevention	Defect prevention activities plan Causal analysis meetings procedures Defect prevention data Defect prevention revisions to standard software process procedure Defect prevention feedback and activities report
Technology Change Management	Technology change management plan New technologies report Technologies selection and acquisition procedure Application of new technologies to standard software process procedure
Process Change Management	Total Quality Management program Process improvement plan Process improvement procedures Process improvement record Process improvement training plan

Figure 2-4. Optimized Level Software Process Documentation: Part 4 of 4

2.2.2 Documentation Process Maturity Model

Experience has shown that examination of the software documentation produced by an organization can provide an excellent indication of the software development capability maturity of that organization. A Documentation Process Maturity Model (DPMM) is referenced in [DPMM] , and defines a framework for assessing the maturity of an organization's documentation processes. The DPMM has four levels (corresponding approximately to the first four levels of the CMM) where the capabilities for each level are defined in terms of the following:

- keywords;
- description of documentation;
- key process area;
- key practices;
- key indicators; and,
- key challenges.

The DPMM is summarized in Figure 2-5.

DPMM Level	Level 1 Ad Hoc	Level 2 Inconsistent	Level 3 Defined	Level 4 Controlled
Keywords	Chaos, variability	Standards check-off list, inconsistency	Product assessment, process definition	Process assessment, measurement, control, feedback, improvement
Description	Documentation not a high priority	Documentation recognized as important and must be done	Documentation recognized as important and must be done well	Documentation recognized as important and must be done well consistently
Key Process Areas	Ad hoc, process not important	Inconsistent application of standards	Documentation on quality assessment; documentation usefulness assurance; process definition	Provide quality assessment and measures
Key Practices	Documentation not used	Check-off list has variable content	SQA-like teams for documentation quality and usefulness; consistent use of documentation tools	Minimum process measures; data collection and analysis; extensive use of documentation tools and integration with CASE tools
Key Indicators	Documentation missing or outdated	Standards established	SQA-like practices; consistent use of documentation tools	Data analysis and improvement mechanisms
Key Challenges	Establish documentation standards	Exercise quality control over content; assess documentation usefulness; specify process	Establish process measurement; incorporate control over process	Automate data collection and analysis; continue to strive for optimization

Figure 2-5. Summary of Documentation Process Maturity Model

2.3 Information Sharing Methods

As information sharing across communication boundaries is made more feasible by better defined processes and new technologies, the methods for development, review, delivery, use, and support of software documentation will likely be revolutionized. Because of the viability of high-speed access, the on-line inclusion of documentation with the operational software product will be realized for most systems. The electronic capability will soon exist to share current documentation on design, update the documentation based on electronic conferencing decisions, and maintain on-line the configuration managed view of all related (linked) documentation. Documents and document sets will be logically linked pieces of information stored on electronic media.

In order to create an algorithmic definition of the electronic links and documentation relations, documentation is being mathematically defined by its unique object sets and associated operations and relations. Processes that develop or use documentation are being standardized for application across project, organization, corporate, national, and international communication boundaries. Tool sets are being developed as an integrated part of these processes to facilitate documentation development, use, and communication. The following subsections provide an overview of some of the potential information sharing methods that should lead to new software documentation form, content, and use. Although the future evolution may be somewhat different than described herein, the key is that process improvement and technology change are likely to significantly alter the form and use of system, software, hardware, and all forms of documentation in the future.

2.3.1 Information Network

The international network is typically referred to as the Internet. This is a complicated collection of computer systems, communication protocols, and physical communication lines that began to take shape in the 1970s. Today, there are many different aspects to the Internet, but one key trend is the increase in the number of users (individuals, groups, companies) that are able to communicate basic electronic information.

With some effort it is possible to share specific documents such as those prepared by standard desk-top publishing software like word processors, graphics packages, and spreadsheets. Corporate groups that are physically separated can share project and product documentation. In an effort to make the location of information more transparent and multimedia information access more seamless, there have been some recent interesting developments that will be important for software documentation. One of these concepts is the Asynchronous Transfer Mode (ATM) network.

In short, the ATM network has an open architecture that will allow increased flexibility and efficiency in tomorrow's high-speed, multi-service, multimedia networks. The current telephone, television, Internet, and other communication networking will be integrated into a logical network concept that digitally connects all of these voice, video, and data services. Multimedia software documentation (hypermedia) may exist in physically distinct parts of the network, linked through a set of access points embedded within each documentation part called a hyperlink. The owner of a document can specify the access privilege and provide hyperlink access on a network page to document objects that are to be made visible to other potential viewers. A project might create a network page to provide access to project, process, and product documentation. An organization might create a network page that would be available to other organizations as well as projects within the organization. A corporation might create one or more network pages for sharing of corporate level information with external customers and internal organizations. The effect is a relational communication linkage scheme.

The World-Wide Web (WWW) is a term being applied to one of the current protocols for accessing and viewing Internet information, employing windows-based browsers with user-friendly click and go interfaces. On the WWW, home pages are created by the host organization as a top-level screen display with links to lower level pages containing information the host wishes to present. Sandia is in the process of creating a Sandia Internal Web for use by organizations and projects within Sandia and an access interface to the external WWW for linkage with other corporations around the world. Sharing voice, video, and data as documentation of software product within an information network infrastructure is within process and technology reality. This technology most certainly will alter the software acquisition, development, use, and support paradigm with regard to documentation of software processes and products. The merging of video, graphic, and text documentation is illustrated in Figure 2-6.

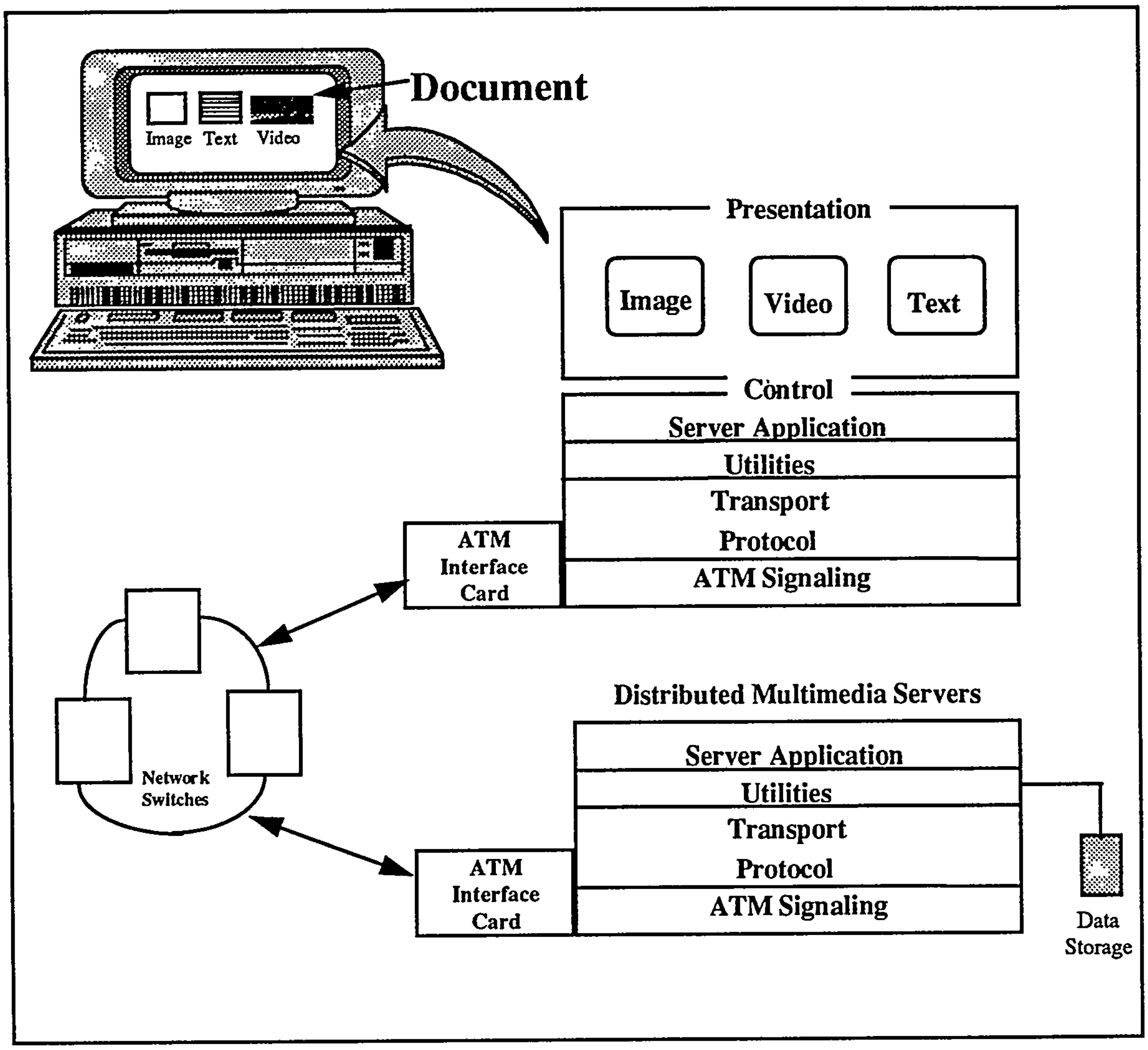


Figure 2-6. ATM-based Multi-Media Documentation System

2.3.2 Continuous Acquisition and Life-Cycle Support

Continuous Acquisition and Life-Cycle Support (CALS) was established by the Department of Defense in 1985 and is a DoD/Industry strategy for the transition to automated interchange of technical data and to the process improvements enabled by automation and integration. CALS will facilitate business integration and promote an electronic commerce environment that will enhance industrial competitiveness and economic growth through process improvement, information technology, and international product data exchange standards. CALS is applicable for all forms of product documentation, including software.

The CALS Software Product Committee has developed a plan to apply CALS strategy to the development, delivery, and maintenance of software products developed under DOD-STD-2167A [DOD2167A] and its successor MIL-STD-498 [MIL498]. Many military standards are gradually being replaced by equivalent commercial standards, as is currently the case with MIL-STD-498. These standards and their accompanying documentation concepts will influence Sandia software organizations. The benefits of CALS to the software life cycle process are to reduce the total cost and time attributed to documentation, and more importantly, to improve access to the information contained in the documents. The reader can contact the Software Technology Support Center [STSC] for the most recent information on CALS and the following supporting standards:

Delivery Media/Format/Organization: these standards address the magnetic tape, CD-ROM, telecommunications, and other applicable media standards for formatting and organizing header file and data file types.

Initial Graphics Exchange Specification (IGES): this standard includes three dimensional technical illustrations and engineering drawings.

Standard Generalized Markup Language (SGML): this standard is for the identification of textual (unfielded) data and defines the following:

- tags - embedded information specifying processing details of sections including references to raster and vector illustrations;
- document type definition - specifies the organization, structure and content of the document and the meaning of the tags;
- formatting output specification instance - specifies document format and style; and
- page description language - specifies how the page is produced (e.g., PostScript).

Requirements for Raster Graphics Representation in Binary Format: this standard includes Type I (untiled) and Type II (tiled), and is simply the format for a facsimile. Tiled refers to the logical cutting of a raster image into rectangular sections, an approach used for more efficient handling of images larger than the normal 8.5" x 11" page.

Computer Graphics Metafile (CGM): this standard includes two-dimensional technical illustrations and graphic art.

Hypermedia/Time-Based Structuring Language: this standard describes a language and syntax for representing objects, including hypermedia, in documents. Standardized linking, alignment, and addressing methods allows objects to be made available in a standardized way.

STEP International Standard for the Exchange of Product Data: This standard is an evolving technology progressing through new development and enhancements. There are many parts to the standard, some of which are standards while others are in development. STEP can be viewed as an overall umbrella standard for most of the above standards whose implementation will be through integrated tool kits based on standard Computer-Aided Drawing

2.3.3 Automated Document Imaging System

Systems are available that allow a completed document to be scanned into an automated document imaging storage and retrieval system. These systems may also permit the storage and retrieval of electronically generated information. If a particular document type has been scanned into such a system, it might be possible to retrieve the document and use text editing tools to create a new document. This use of an imaging system might save significant time if there are similarities in the different software projects or similarities in the types of applications.

There are two basic types of document imaging systems: (1) those that allow the user to input information about a document and store the information in an accessible database, with the document bit image only being scanned and stored; and (2) those that allow the entire document to be scanned and stored in an electronic manner such that text retrievals, searches, and editing can be performed. Because the first type is usually significantly cheaper and quicker than the second type, some hybrid systems have been developed which allow you to select either method for document storage and retrieval.

Sandia's Technical Library is currently in the process of establishing a document imaging system into which many of Sandia's technical reports and documents will be scanned and made available to Sandia employees. Part of the advantage of this system may be the ability to look for key words, either in the document title, in a database of keywords established at the time the document was scanned, or in the actual documents themselves. If the keyword search is successful, then the user may be able to download the document (given the proper access authority) to their computer or some other transferable electronic medium. Depending on the type of scanning system implemented by the Technical Library, it may also be possible to scan a paper document and have the document converted to the word processor or other application format appropriate for the users.

2.3.4 Integrated Development Environment and Assistant

Sandia is developing a system to provide easy and guided access to electronically accessible knowledge, information, and tools in an integrated development environment. This system is called Integrated Development Environment and Assistant (IDEA) and may be a valuable resource for sharing software documentation developed using various tools.

Currently, IDEA provides an electronic facility for access both to general Internet resources and to much of Sandia's unclassified information: project files, lessons learned, administrative and technical processes, corporate documentation (e.g., standards, policies, processes), and product line resources. For example, IDEA lets the user view and navigate archived project files (e.g., schematics) developed in a particular tool even if the user doesn't own a copy of that tool. IDEA also offers an assistant to guide the user to those resources that are relevant to the user's needs. The basic structure of IDEA is illustrated in Figure 2-7.

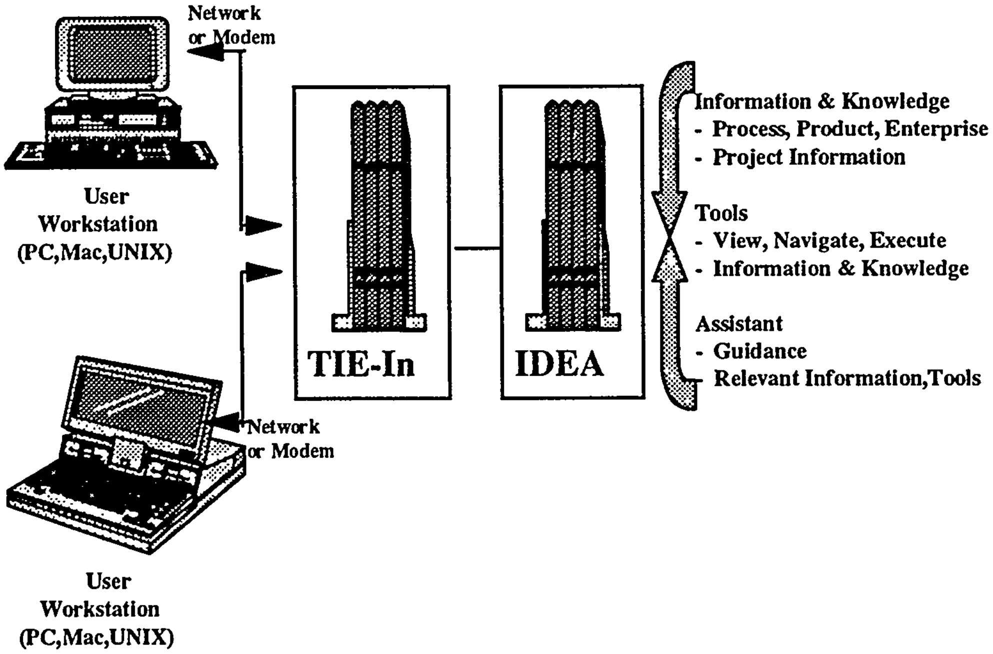


Figure 2-7. IDEA Information Management

When fully developed, IDEA will provide a complete integrated development environment with a smart assistant. Integration of tools, information, and expertise will make IDEA more than the sum of its parts. For example, IDEA won't just instruct a user to author a software requirements document but will be able to bring up the word processor of choice with a template that will aid the user in developing the document.

Users may connect to IDEA via network or modem from most PC, Macintosh, or UNIX workstation platforms. From IDEA, users may access resources originally developed on PC, Macintosh, and UNIX workstation platforms. The Technology Information Environment for Industry (TIE-In) will provide a front end to IDEA allowing both external (United States Industry) and internal (Sandia) access to IDEA.

2.4 Formal Methods for Documentation of Software

Many software developers have found that current documentation standards are not as useful as desired. A common complaint is that current narrative documentation in natural language is not as easy for software engineers and programmers to read and understand as the source code language version of the final code. As a result, a number of concepts have been explored as potential approaches to more formal mathematical representation of documentation. One such highly mathematical approach is described in detail in [PARNAS]. The following paragraphs provide an overview of these formal documentation concepts.

Design Through Documentation: The approach accepts the premise that documentation of a computer system must include a Systems Requirements Document that consists of both systems requirements and system design. Such a document is considered to be a description of the system as a whole, in the form of a description of the hardware structure and a black-box description of the software. From that point, particularly for large systems, the software development is split into several smaller work assignments. The goal for each assignment is to design and implement a group of programs that, collectively, are called a module. Programs in such a module share access to a data structure and implement one or more abstract objects. The documentation proposed for this approach consists of the following:

- **Software Module Guide** - for the overall software system; describes the structure of the software system by indicating the design decisions hidden within each module;
- **Module Interface Specification** - for each module; provides a black-box description of the behavior of objects created by that module; and
- **Module Internal Design** - for every implementation of a module interface specification; describes the internal data structures and the effect of the module's access-programs on the state of the structure.

Design and Review Responsibilities: The approach follows the premise that the reviewer or maintainer of a program should never have to guess at its structure. Designers systematically record information that reviewers and maintainers would otherwise have to discover for themselves. Programs should be presented to reviewers and maintainers as a collection of small parts, each with a precise description of its function. This method will allow a review of each small part separately, with the understanding that, if each component is correct, the whole program is correct. Reviewers must check both the structure and each small fragment against the description of its function. To facilitate the review and maintenance effort, designs are formally documented using mathematics.

Display Method of Documentation: The Display Method for mathematical documentation of programs requires designers and implementers to present their programs through the use of tables to describe mathematical functions, relations, and sets in the form of displays. The method is based on an understanding that a well-structured program can always be written as a short text in which the names of other programs may appear, and the programs named may also be short programs. Many short programs will be the result, and in order to understand one program it will be necessary to understand a number of other programs. The display method overcomes this problem by presenting a program description in such a way that the program's correctness can be examined without looking at other displays.

3 Making Project Documentation Choices

This chapter provides guidance on requirements for the structure and information content of a minimum document set based upon project characteristics and constraints. Using this guidance, the reader should be able to make some intelligent choices on the cost and benefits of selecting, tailoring, constructing, controlling, and improving the quality of a software project's document set.

The document set that the project develops may consist of one or more documents, depending on the size of the project and the needs of the intended user. Each document of a document set may be one or more volumes depending on the amount of information to be presented. The media form of the documents will depend on the intended use of the documents and the technology available for representing the documentation information (see Chapter 2 for some more advanced documentation concepts).

To determine appropriate project documents, the software product, its application, and the audience that will use the product must be identified. A document set should contain the information needed by the intended user. Depending on the nature of the software, the document set may need to be integrated with system-level documentation. The presentation style and level of detail should be tailored for the intended audience. When a document set must be prepared for audiences with widely differing needs, then the materials should be appropriately separated to meet the specific needs of each audience. For example, installation procedures may be required for field personnel who have many duties besides installing software. The documentation in this case should specifically address the skill and background of the installers and be written in clear step by step procedural language. Software operating guidelines will be written in a style more oriented to simple computer operating input and output information and less software engineering terminology. Software maintenance documentation may assume the user will possess more software engineering skills and address the documentation style in an engineering language.

3.1 What Document Set is Adequate for My Project?

Documentation Suggestion # 0
Keep it simple: a small amount of very useful documentation is preferred.

This section provides a step-by-step process for making selection choices for a project's document set. Several examples of applying the process to some Sandia-like projects is presented in Appendix D. Every software project is unique but there are characteristics of each software project that are in common with other historical software projects. Experience with these historical projects provides guidance for the level and type of documentation that would be most appropriate for your software project. The types of documents and rigor of documentation content are a function of many factors: project budget, schedule, complexity of software, quality constraints, performance requirements, and customer needs and specified requirements. The function of documentation is to capture snapshots of information, perhaps in various media forms (electronic and paper), about the operational software products and the processes and intermediate products upon which the operational products are dependent. The precise document set that is adequate for your software project will clearly depend upon the process that is followed for your project.

The documentation that is adequate for your project can generally be determined by asking the following questions:

Question 1: What documentation is contractually required by my customer(s)?

Documentation Suggestion # 1
Negotiate with the customer to deliver the minimal sufficient document set.

Contractual requirements often specify a minimal set of documentation. The requirements may be stated in different terms depending upon the software project customer. If the Department of Defense is the customer, then documentation requirements are usually based upon DoD standards such as MIL-STD-498 (recent replacement for MIL-STD-2167A). Documentation deliverables are specified in terms of Data Item Descriptions (DIDs) that identify documentation formats, section thematics, and any tailoring requirements for the sections within. The documentation formats and thematics are similar to what is presented in Chapter 4 but may differ in name and/or scope. If the Department of Energy is the customer, then DOE Orders and Sandia-related procedures such as the Engineering Procedures (see references [EP401040] and [EP401045]) may be applicable. If a commercial industry is the customer, then customer-specific standards may apply, including the customer's internal standards and national professional standards such as published by IEEE or other such organizations. If possible, it is best to work with the customer and negotiate what documentation is most appropriate for development on the software project, including documentation that will be internally used as well as delivered to the customer. This negotiated document set is then the contractual basis for the project. Once the contractually required documentation is identified, the next step is to make sure adequate descriptions exist of the document format and content. It is also necessary to ensure that the software processes to be used have activities in place to develop new and update existing documents in accordance with the contractual requirements.

Question 2: What documentation is required by Sandia?

Documentation Suggestion # 2
Use a Software Management Plan to specify your documentation approach.

Sandia Laboratories Policy 1011[SLP1011] requires all software to be managed through definition of a Software Management Plan. Each organization should be covered by at least one such Plan that is tailored to the type of software being managed by the organization. The Plan should indicate what software models for acquisition, use, development, and support are to be followed by the organization. These models will in turn reference appropriate customer and Sandia-specific requirements for documentation. Again, the documentation format and content required by any specific organization standards may be similar to what is discussed in Chapter 4.

Sandia has several general purpose documents that provide software development and support life cycle model and process guidance:

- Sandia Software Guidelines, Volumes 1 through 5 [SSG]
- Preferred Processes for Software Development [PPSD]
- Process Guidelines for Sandia WR Software Development [PGWR]
- Engineering Procedure, Definition of Computer Software Configuration Items [EP401045]
- Software Development/Support Methodology [SDSM]

Question 3: If there are no contractual requirements, what documentation does the customer require?

Documentation Suggestion # 3
Use Chapter 4 guidelines for format & content of documentation when the customer has no preferences.

There are four basic customer types that the project group should consider:

- user: what documentation is needed to simply use the software?
- developer: what documentation is needed to facilitate developing the software?
- supporter: what documentation is needed to make changes to the software to correct defects, make enhancement changes, and adapt the software to changes in the operational environment?

- manager: what documentation do I need to understand the nature of the project, including progress, risks, and decision making aids?

The types of customers that the software needs to serve will determine any additional documentation requirements beyond contractual and Sandia corporate requirements.

Question 4: What documentation does the project require?

Documentation Suggestion # 4
Develop documentation as if you were the user, supporter, and manager.

Once you have satisfied the requirements of your customers and Sandia good software engineering practices, the most critical group to satisfy is your own project group. If your project group can view the software project as if they were the customer, the documentation needs will be much clearer. Your project group should view itself as the customer in terms of using the software, supporting the software, and understanding the conduct of the project from the viewpoint of the end customer. What would you need to know and in what form would you need it in order to perform your function as a customer?

Documentation Selection Process

The next few figures provide a step-by-step description of a documentation selection process that can be applied by each software project group, whether the group is one person or a large multi-functional organization. The general documentation selection process is described in Figure 3-1. Risk scoring guidelines are described in Figure 3-2 and guidelines for selecting a document set based on the computed risk score are described in Figure 3-3. Although the factors identified in Figure 3-2 should apply to all software projects, individual organizations may wish to tailor the factors and assigned risk values to the organization's own historical data. The selection process, risk factors, and associated risk scores are discussed in more detail in Appendix D where a variety of examples are provided.

The time and effort to perform the documentation selection process will depend upon the size of the software project, available dollars and personnel to perform the selection process, customer and contract requirements, schedule, and other project specific factors. A reasonable effort to perform the selection procedure outlined in Figure 3-1 for a small size and low cost project is about one to three person days. A very large size and high cost project may require two or three person weeks. These should be the extremes, with projects in intermediate categories requiring varying degrees of resources within these ranges. If you are tempted to skip this selection process, then you should be prepared to pay the costs for either inadequate or excessive software project documentation.

1. **Objective:** Determine appropriate software documentation for a software project.

2. **Abstract:** This process enables the Project Manager, Leaders and Software Personnel to quickly determine a minimal set of documentation for development/update by the software project. The process steps can be used by any type of project. This process should occur sometime between the initial discussion of the project with the customer and the development of a project plan by the project manager. A recommended document set is derived using a practical software documentation risk scoring guidelines. Further analysis and tailoring can be done by project personnel to make sure the selected documentation is satisfactory. This process can be reentered any time the basis for the document set selection changes during the project.

3. **Special Responsibilities:**
Project Manager: Establishes working group. May be part of working group.
Project Leaders/Key Project Software Personnel: May be chair or member of working group.
Selection Working Group: 1 to 4 key software project personnel.

4. **Inputs:** Customer(s) and Customer(s) Requirements, Type of Project (WR, non-WR), Project Description, Project Tasks, Software-Based Tasks, Possible Life Cycle Model, Existing Documentation (department, project, customer); Document Templates and Documentation Guidelines (e.g., this volume). Not all inputs may be available or complete depending upon when the negotiation for the document set selection is initiated.

5. **Entry Criteria:** Resources available for selection process; preliminary project and software function descriptions available; preliminary customer and project requirements specified to appropriate level of detail.

6. **Procedure:**
Step 1: Project manager forms documentation specification working group.
Step 2: Working group reviews inputs.
Step 3: Working group determines software risk levels for size, budget, schedule, customer needs, project lifetime, expected customer growth, usage growth, support & maintenance, personnel skills. Working group identifies previous similar projects.
Step 4: Working group completes risk range scoring from guidelines and any other analysis.
Step 5: Working group selects minimal document set based on risk level.
Step 6: Working group reviews guidelines for format of documents in the selected set and finalizes format.
Step 7: Working group establishes consensus on tailoring/expansion of minimal selection set based on risk range.
Step 8: Project manager signs off on document set and includes set in project plan.

7. **Exit Criteria:** Document set is selected; risk range change scenarios have been analyzed; selection decision criteria have been documented in project notebook.

8. **Output:** Selection Decision Criteria; Software Document Set Selections; Risk Range Change Scenarios; Any Risk Range Migration Plan

9. **Auditability:** Copies of decision criteria used for documentation selection and the final software documentation selection decision for the project should be maintained in a project notebook.

Figure 3-1. Document Set Selection Process

1. Process:

10 - acquisition: purchase of S/W for use as part of project
documentation - use vendor documentation (operation guide, user's guide)

100 - development: S/W developed to be used in system application
documentation - developer, supporter, user

10 - use: operation/installation of S/W for use in system application
documentation - installation, user support

50 - support: S/W modified for use in existing system application
documentation - updates to existing documentation, development of new supporter and user documentation

2. Size (NCSLOC):

10 - low (<1000): documentation set can vary from Source and Load to SDP, SRS, SSTP, Source, Load

50 - medium (1000-19,999): documentation set can vary from Source and Load to SDP, SRS, SDD, SSTP, Source, Load

100 - high (20,000-100,000): documentation set can vary from SDP, SQAP, SRS, SDD, SSTP, Source, Load, SSMP to Extended Set

200 - xhigh (>100,000): documentation set can vary from SDP, SQAP, SRS, SDD, SSTP, Source, Load, SSMP to Extended Set

3. Cost: (Life Cycle Cost to Sandia and its customers - don't include commercial vendor development cost, do include any vendor support cost passed on to SNL).

10 - low: life cycle cost for S/W is estimated to be below $25,000

50 - medium life cycle cost for S/W is estimated to be between $25,000 and $50,000

100 - high life cycle cost for S/W is estimated to be between $50,000 and $100,000

200 - xhigh life cycle cost for S/W is estimated to be between $100,000 and $1,000,000

500 - xxhigh life cycle cost for S/W is estimated to be over $1,000,000

Caveat: This selection risk matrix should be tailored to an organization's historical data when available.

Acronyms Used Above:

NCSLOC	Non-Commented Source Lines Of Code
S/W	Software .
SDD	Software Design Description
SDP	Software Development Plan
SQAP	Software Quality Assurance Plan
SRS	Software Requirements Specification
SSMP	Software Support/Maintenance Plan
SSTP	Software System Test Plan

Figure 3-2. Document Set Selection Risk Scoring Guidelines: Part 1 of 3

4. Customer Characteristics:

The following characteristics are somewhat overlapping, so it is more difficult to provide generic guidelines on how documentation could be allocated. Some guidance includes:

10 - no customer: S/W is acquired, developed, supported, used solely by a single person or small group with no intentions of using the software or results from the software execution for external or other Sandia internal customers. Documentation requirements should be set by the single person or small group based on their own internal consideration.

50 - internal Sandia customer: normal documentation guidelines follow from other risk areas depending on how the customer intends to use the S/W. There should be a Sandia review procedure for release of S/W to internal or external customers, and this release procedure might require different levels of documentation consistent with this volume's guidelines. Count 50 points plus the following depending upon the project:

100 - WR project
10 - Non-WR project
10 - Research
100 - Non-research

100 - external Sandia customer: must assume the external customer intends to use the S/W for operational use and perhaps critical decision making. Count 100 points plus the following additional points depending upon the project:

100 - WR project
10 - Non-WR project
100 - Work For Others
10 - Non-WFO
10 - Research
100 - Non-research

5. Life Cycle Characteristics:

Prototype Activities/Production Product Activities
10 - Prototype
100 - Production

Expected Life Cycle Length
10 - Short (less than 1 year)
50 - Medium (from 1 to 5 years)
100 - Long (greater than 5 years)

Activities of Concern
(Cumulatively add the scores depending on the activities to be performed)
10 - Concept Definition/Requirements Analysis
50 - Design
20 - Implementation
30 - System Test
20 - Installation/operation
40 - Support

Acronyms Used Above:

S/W	Software
WFO	Work for Others
WR	War Reserve

Figure 3-2. Document Set Selection Risk Scoring Guidelines: Part 2 of 3

6. Support Environment Characteristics:

Platform (planned or selected)

10 - Personal computer workstation
50 - Network workstation
100 - Mainframe

Automated Tool Support (planned or selected)

100 - Basic set
50 - Intermediate set - some Computer-Aided Software Engineering
10 - Integrated Computer-Aided Software Engineering

Personnel experience with the selected or planned support environment

50 - Inexperienced
20 - Medium experience
10 - Experienced

7. Historical Project Experience Characteristics:

Experience Level

50 - Low: none or very few similar projects
20 - Medium: several similar projects
10 - High: extensive experience

Personnel Stability

50 - Low: significant personnel changeover is expected
20 - Medium: moderate personnel changes are expected (no greater or less than reasonable)
10 - High: no personnel changes are expected, particularly for key personnel

8. Sandia Customer and Political Risk Characteristics:

Operational Reliability and Safety

10 - Low risk to human life, property, and/or political catastrophe
100 - Medium risk to human life, property, and/or political catastrophe
500 - High risk to human life, property, and/or political catastrophe
1000 - Extra high (catastrophic) risk of great magnitude (thousands killed, billions($) damage, and/or political chaos)

Customer Satisfaction

10 - Low risk to project success if customer is not satisfied with software
100 - Medium risk to project success if customer is not satisfied with software
200 - High risk to project success if customer is not satisfied with software

9. Other Possible Questions

- Is it possible the software risk level might change in the future?
- What are the major impacts on documentation if the risk level does change?
- How would the software be reengineered to satisfy new requirements of the new risk level? Can the impact of future reengineering be affected during the current project?
- Identify the most likely risk level change scenario within the next 5 years - what documentation would be required to support the risk level change? Should that additional documentation be part of the current documentation selection set?

Figure 3-2. Document Set Selection Risk Scoring Guidelines: Part 3 of 3

1. Document Sets:

Minimal Document Set:	CODE, SDF
Basic Document Set:	SRS, CODE, SSTP, SDF
Small Document Set:	SDP, SRS, CODE, SSTP, SDF
Medium Document Set:	SDP, SRS, SDD, CODE, SSTP, SSMP, SDF
Large Document Set:	SDP, SQAP, SCMP, SRS, SDD, CODE, SSTP, SSMP, SVVP, SDF
Complete Document Set:	SMP, SDP, SQAP, SCMP, SRS, SDD, CODE (source, object, load, command language), SSTP, SSMP, SVVP, SDF, SUG, SSP, Vendor User Documentation, and Contract Documentation

2. Map of Risk Score to Document Set:

xLow Risk Range < 500:	Select Minimal Document Set
Low Risk Range 500 to 999:	Select Basic Document Set
Medium Risk Range 999 to 1499:	Select Small Document Set
High Risk Range 1500 to 1999:	Select Medium Document Set
xHigh Risk Range > = 2000:	Select Large Document Set

3. Some Additional Guidelines: If the project statement of work from the customer includes documentation requirements then map the contractual documentation to these guidelines, determine which documentation is either included or not included per these volume 2 guidelines, identify the risks to the customer of either having too much or too little documentation, and establish final customer documentation requirements. Use MIL-STD-498 documents mapped to volume 2 documents if the customer is a Department of Defense agency. Include any other major Sandia customer document sets appropriately mapped to the volume 2 documents.

- If security is a major issue: add SSP.
- If a user interface is a critical issue: add SUG.
- If there are special installation and/or support characteristics, add SSMP.
- If operational reliability or customer satisfaction is critical, add SVVP.
- If S/W is being purchased, add Vendor User Documentation.
- If the organization is not covered by a software management plan per SLP1011, add SMP.
- This selection risk matrix should be tailored to an organization's historical data if possible.

Acronyms Used Above:

SCMP	Software Configuration Management Plan
SDD	Software Design Description
SDF	Software Development Folder (also known as Software Development File)
SDP	Software Development Plan
SLP	Sandia Laboratories Policy
SMP	Software Management Plan
SQAP	Software Quality Assurance Plan
SRS	Software Requirements Specification
SSMP	Software Support/Maintenance Plan
SSP	Software Safety/Security Plan
SSTP	Software System Test Plan
SUG	Software Users Guide
SVVP	Software Verification and Validation Plan
S/W	Software

Figure 3-3. Guidelines for Selecting a Document Set Based on Risk Score

3.2 How Can I Control the Document Version?

Uniquely identifying successive versions, keeping a change history, and marking sections that have changed from one version to the next provide document version control. These methods and some additional considerations that may influence how documentation is controlled are described in the following subsections.

3.2.1 Configuration Control of Documentation

Sandia Software Guidelines, Volume 4: Configuration Management [SSGv4] addresses not only several control options but the underlying management aspects of evolving engineering information as well. While the focus in this section is controlling documentation versions, the reader should consider how changes to documentation and other software engineering information are to be managed for the entire project.

3.2.1.1 Great Beginnings

Documentation Suggestion # 5
Establish a document configuration control plan when the project begins.

As the project begins, control may be limited to saving copies of each iteration of the document or document set. These copies can quickly become voluminous and untraceable, especially if changes are not marked or some sort of unique identification is not used to differentiate the iterations. Another option is to just keep the latest iteration. This option can present severe complications if the actual changes are not identified, even when the date or some other document version identification is used. Also, if the reason for making the changes is not identified, even single-person projects can unintentionally return to previously discarded ideas and fruitless solutions. Therefore, regardless of how a project is started, uniquely identifying each iteration, maintaining a history of the substance of the changes, as well as marking the changed sections is recommended.

3.2.1.2 Life Cycle Considerations

Documentation Suggestion # 6
Control documents so all participants can have a common baseline.

The purpose of controlling documentation should be to ensure the engineering information accurately reflects the evolving product. Such control should be part of a configuration management scheme designed to met the needs of the project. As the project proceeds from concept to requirements definition to design to implementation, information contained in the various documents must progressively be more tightly controlled. That is, documents containing the information should have greater restrictions placed on them in terms of who may make changes and why changes should be made. At selected points in the development, some information may need to be frozen into a baseline from which future work can confidently proceed. Changes to baseline information should be approached with great caution due to the potentially high impact on project schedule and costs . However, having a baseline does not preclude changes. It is intended to assure that changes are not made without the notification and concurrence of those persons affected by the changes. See reference [SSGv4] for additional information on the concept of Libraries as a means to control software documentation.

3.2.2 Examples

3.2.2.1 Internal Project: Manual vs On-Line Control

Documentation Suggestion # 7
If an on-line capability to mark changes is not available, then provide a capability to compare documents and print differences.

For many small projects, most documentation can be maintained manually. That is, version identification nomenclature is manually updated, documents and other information are stored and archived within the framework of the computer system hosting the word processor, design tools, and compiler. Many of today's processors and other software development tools have the capability to automatically mark changed sections of the item being processed. Updating a change history must be performed manually whether the control is primarily manual or through an on-line (automated) tool.

On-line control may be part of a Computer-Aided Software Engineering (CASE) tool or stand-alone software configuration management tool. In these cases, version identification updating may be built into or scripted into the tool; storing, archiving, and cross-referencing documents or other software development information is performed within the scope of tool control. Even on small projects though, making use of an automated tool to consistently update version identification and provide cross-referencing information is extremely useful, particularly for code generation. As the project increases in size or people, on-line control becomes more important. The management aspect of controlling software documents is more fully discussed in SSGv4, sections 2.1, 2.2, 3.1, and 4.1.

3.2.2.2 Controlling External Customer Documentation Releases

Documentation Suggestion # 8
Control the document set configuration over the entire software life cycle.

Release of software to an external customer requires that the person or group responsible for development and/or support of the software maintain adequate control of the documentation describing the design of the software in use by the customer until such time as the customer has been notified that it will no longer be supported. Even then, an archive copy of all documentation should be kept until all official copies of the software are retired. If multiple versions of the software are in current use by customers, careful attention to maintaining the integrity of each version almost mandates the use of an on-line, automated configuration control tool. Such a tool will provide a higher level of confidence that any changes made to one version that affects another version will in fact be successfully implemented.

3.2.2.3 Maintaining Documentation

Documentation Suggestion # 9
Use documentation tools that are likely to be supported over the software's life.

Although keeping track of documents and document versions over the life of a software product is important, it is just as important to maintain version control of the tools used to create the documents. Fortunately, most tools provide upward compatibility between versions. In addition, the most popular tools often recognize the competition and provide ways of importing data from them. For example, Microsoft Word provides help in importing files created by the Word Perfect word processing tool and vice versa. The greatest problems may occur if a special software tool is used to create specific information for a document, and the software vendor for that tool goes out of business or is merged with another business during the software life cycle. Unless a copy of the tool is maintained by the project staff, it may be very difficult to maintain software documents. These issues are more important for software projects having a long life cycle since the tools are more likely to have different versions over time, and there are more opportunities for bad things to happen to tool vendors. Another consideration is that successful documents may be useful to other or future software projects. These projects will likely want automated access to the documents to reduce duplicate effort. Although scanners may be useful in solving these problems, it is probably better to have access to documents through the tools that were used to generate them originally.

This caution is just as important for automated tools used to maintain configuration control over the documents. For instances where a CASE tool is used for this function, it is very important that the tool be available not only for the most efficient maintenance of software code and documentation but also to control it in the manner used throughout the software development. Since CASE tools are not as standard as document generation (word processing) tools, their version control is important because there will be less likelihood for compatibility if the CASE tool suddenly becomes unsupported. When the life cycle is long or follow-on efforts require the reuse of documentation, attention should be given to license expiration dates and the use of escrow agreements for software purchase options.

3.2.2.4 Using the Sandia Drawing System

Documentation Suggestion # 10
Use the Sandia drawing system identification scheme; archive software documentation in the drawing system.

Sandia has a corporate configuration management system that facilitates configuration control of engineering information. Principally used by weapon development projects, this system is available to any software development project. The system provides an identification scheme to differentiate versions of the documentation that describe a software product. Documents are referred to as Drawings in the Sandia Drawing System, a legacy from drawings of hardware components. Each document type has a prefix slaved to a 6-digit part number that is unique for each unique software product. A two- or three-digit suffix defines the version migration. For software, the drawing system represents a library in which stable (baselined) information is stored. SSGv4, Chapter 3, contains a more detailed description of the Sandia Drawing System and references the Engineering Procedures that define the Sandia Drawing System. In particular, Engineering Procedure EP401045, Definition of Computer Software Configuration Items [EP401045] defines the nomenclature, types of documents, and minimum document set for War Reserve software product.

3.2.2.5 Energy Science and Technology Software Center (Export Control)

Documentation Suggestion # 11
Review the requirements of DOE Orders for applicability to the export of software documentation.

Sandia is required to submit certain types of software products to the Energy Science and Technology Software Center (ESTSC), in accordance with the Policy Statement from DOE Order 1360.4B [DOE1360.4B]. "Scientific and technical computer software programs developed and/or modified during work supported by DOE or during work carried out for others must be provided to the ESTSC for inclusion in DOE's information announcement and dissemination systems and, when appropriate, for dissemination outside the Department. Such software shall be made available to the scientific, technical, academic, and industrial communities, and to the public, only through approved channels as security, patent, contractual, and other DOE policy considerations permit." The term approved channels includes Specialized Information Analysis Centers which are activities that collect, evaluate, announce, and provide scientific and technical computer software in specialized subject areas. The Specialized Information Analysis Centers are defined in DOE Order 1360.4B and are listed as: the Engineering Physics Information Center of Oak Ridge National Laboratory; the National Nuclear Data Center of Brookhaven National Laboratory; and the Bartlesville Project Office.

Certain types of computer software are excluded from these provisions. The exclusions are discussed in DOE Order 1360.4B. In addition, DOE Order 1360.4B requires that all scientific and technical computer software developed by Departmental Elements, contractors, and financial assistance recipients under their purview contain specific Distribution Statements and Disclaimers on the cover and title page of all documentation developed for that software and any computer-readable medium used for transmitting that software. These statements currently read as follows:

DISTRIBUTION STATEMENT

"This computer software has been developed under sponsorship of the Department of Energy. Any further distribution by any holder of this software package or any data contained therein outside of DOE offices or other DOE contractors, unless otherwise specifically provided for, is prohibited without the approval of the DOE Energy Science and Technology Software Center. Requests from outside the Department for DOE-developed computer software shall be coordinated with the Energy Science and Technology Software Center, P.O. Box 1020, Oak Ridge, TN 37831."

DISCLAIMER

"This material was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights."

3.3 Are There Tools to Assist Documentation?

Documentation Suggestion # 12
Make sure you have a defined process into which selected software documentation tools can be effectively integrated; tools are not a panacea for a poor software process.

Many tools aid in all activities and phases of software documentation. A distinction, however, needs to be made between tools one might use to generate and support software documentation versus methods used to write a document. It is not the specific intent of this volume to describe how to write accurate and clearly understood software documentation. Some thoughts on producing quality software documentation are covered in section 3.5 of this volume. If you need or desire additional information on how to write software documentation, a good reference is [BROCKMAN]. This book also has an extensive bibliography of other materials related to the philosophies of writing software documentation. The Software Technology Support Center [STSC] is a good source for the latest tool technology to support software documentation.

This volume, and particularly this subsection, will concentrate on providing information on generic tools and methodologies associated with those tools that can assist you in preparing quality software documentation products. To this end, a documentation tool taxonomy is discussed. In addition, things you may wish to consider in selecting and using tools and other available resources and software documentation aids will be discussed. One key observation is to remember that tools can only assist you in preparing a quality software documentation product. The software engineering process, of which the documentation is an integrated part, will likely be of more influence on the quality of the product than any specific tools.

3.3.1 Documentation Tool Taxonomy

A taxonomy of commercial documentation tools is illustrated in Figure 3-4 [STSCdo]. The tool applications listed in this figure are examples only. Every year brings the introduction of additional tool applications and specific tools to support the applications, so this list may be outdated fairly rapidly. In spite of this caveat, the tools that are likely to be of most interest over the next decade include those in the categories of desktop publisher, graphics processor, and tools that form software integrated tool environments. Additional information about these tools is provided in the following paragraphs. In addition, Chapter 2 provides some futuristic documentation concepts for which a new generation of documentation tools may be required. Other references that discuss Computer-Aided Software Engineering tools include [SSGv5], [IEEE1209], [IEEE1348], and [SQAS-CASE].

3.3.2 Word Processors and Desktop Publishers

It is now possible to integrate text and figures on the same file in a word processing system. It is even possible to identify key words that may be used to automatically build a document's index. Some word processors also have special modes for various programming languages (e.g., to assist with the syntax features of the language). Where these word processors have allowed the author (or an editor, if assigned) to integrate all of the capabilities of text, graphics, tables, and so forth, and to produce professional looking documents printed on a laser printer, the systems have been called Desktop Publishers (DTPs). Many DTPs provide an option to import graphics generated using graphics processors not already a part of the DTP itself.

It is now possible to export or import most data between compatible computers. Therefore, various parts of a document may be prepared on many different media and still integrated with a single word processing tool to form a complete document. Be aware, however, that not all of these conversions may be necessarily clean, and some effort may be required to convert special characters or handle capabilities in one tool that are not recognizable on another tool.

3.3.3 Graphics Processors

Many of the modeling techniques used in software development use graphics. There are many tools that provide utilities for creating, editing, and managing graphics. Some common examples of methods that use graphical techniques and thus have a need for graphics processor tools include: Natural Language (Nijssen) Information Analysis Method (NIAM), Structured Analysis, Structured Design, Object-Oriented Analysis, and Object-Oriented Design. Such tools include generic drawing packages, special purpose application packages, and software engineering technique-specific tools. These tools are usually part of a Computer Aided Software Engineering tool set to assist in developing the software. There are a variety of commercial tools that may be purchased to assist in developing required graphics.

Tool Subdomain	Applications
Color Publishing	Color publishing products may be useful in producing color portions of technical orders or corporate brochures.
Database Publishing	Direct application of these tools to software projects may increase as word processors and DTPs develop the functionality needed to access databases of objects.
Desktop Publisher	Software development projects with resources and a life cycle that justify a higher end tool; projects that use other CASE tools.
Document Management	Used in implementation of libraries or repositories where paper is being replaced by electronic media. May also be used in the configuration management and archiving of software documentation associated with software maintenance projects with a long life cycle and frequent and/or significant updates.
Electronic Distribution	Allows documents to be transmitted electronically complete with graphics and formatting; recipient's display is independent of application or platform. Receiver can view a WordPerfect document, for example, without having WordPerfect available.
Extract from Source Code	These tools may be useful if source code documentation is non-existent or of poor quality. Reengineering tools fall into this category.
Filter/Translator	Translation of an input file (text or graphic) to an output file in another desired format.
Flowcharting	Requirements analysis, design, process documentation, either individually supported or supported through an integrated CASE environment.
Forms	Provide framework forms with the facility to create, edit, modify the forms and the entry of data on the forms.
Graphics/Presentation	May be useful for complex figures and tables, flowcharts, data flow diagrams, raster and vector images of the computer hardware, and project status.
Hypermedia	Allows for the integration of several forms of information including text, graphics, video, animation, music, voice and sound. Hypertext is a related term for textual information that is linked, usually through pointers. Products can be applied to most standard document types: requirements, design, test, maintenance.
Page Layout	May provide the DTP functions that a specific word processor lacks without incurring the expense of a high-end DTP. Good for newsletters and short documents.
Scanning	Software maintenance projects whose software product has a long life cycle with frequent and/or significant updates, and whose documentation is not in electronic form.
SGML-Based/CALS	SGML and CALS tools act on an input file or object to perform a conversion or validation to ensure that the output is compliant with the desired standard.
Technical Drawing	CAD-like tools for drawing straight lines, curves, and for dimensioning.
Text Processor/Batch Compiler	If a project lacks the funds for a high-end DTP but has a requirement for high quality, complex documentation involving multiple authors, a text processor/batch compiler may be useful. Trend is away from this subdomain and toward DTP.
Word Processor	Creation of input files to desk top publishing and text processors; creation of documents of moderate length.
Workflow	Provide means for electronically implementing project workflow without extensive text manipulation. May implement procedures such as electronic distribution/review.
Workgroup	Includes products with text manipulation features that run on a local area network and emphasize seamless sharing of text and graphic data among project members. Products provide access to databases, spreadsheets, text files, graphics, forms, and e-mail.

Figure 3-4. Documentation Tool Taxonomy

3.3.4 Integrated Tool Environments

Tools in this category include workbench environments that integrate many tools together. Environments include: UNIX platforms, "C" environments, "Ada" environments, and method-based environments such as an Object-Oriented, Structured Design, and so forth. An integrated tool environment includes tools for software management and software engineering. Documentation within such an environment is essentially paperless, although the need for paper copies of almost any of the standard documentation information could be produced through a documentation report generator capability. Almost all other tools are candidates for inclusion in the integrated tool environment.

3.3.5 What to Look For In Selecting/Using Tools

Fairly obvious factors to consider when selecting software documentation tools for your project include:

- cost versus benefit analyses;
- applicability to your software documentation issues;
- availability of and ease of obtaining the product;
- how easy it is to use the product (human engineering);
- hardware/software requirements and constraints;
- staff training time;
- acceptance by those who must use the tools;
- compatibility relationships with other project hardware, software, and project standards;
- supportability issues; and
- version control of the tools.

Included in the supportability issues are such considerations as what kind of technical service support will you receive if you have questions about tool usage or error messages, an evaluation of the vendor's history in supporting previous version releases of a tool, an assessment of the upward compatibility record for the tool vendor as new versions are released, and an understanding of the financial and other sound business aspects of the vendor. The best way to get this information is to obtain as much vendor literature as possible, and to perform an extensive search of various technical sources (such as magazines) which tend to rate various vendors and their products regarding value, performance, reliability, repair service, and technical support. Another good source of information is to ask those who have used the tool(s) on other projects to determine their overall satisfaction with the factors mentioned above. Extensive guidance is available in reference [IEEE1348] for the adoption of CASE tools, and in references [IEEE1209] and [SQAS-CASE] for the evaluation and selection of CASE tools.

Version control of documentation tools is almost as important as configuration control of the software documents themselves that were generated using those tools. The reason this is important is the need to maintain documentation for existing projects and to use information from existing documentation to create documentation for new projects. If you do not know which version of a documentation tool was used to create a specific document, you may have great difficulty making future changes to that document. In addition, if there is ever a need to use that documentation or parts of it for another project, you may not be able to electronically reproduce it, requiring it to be retyped, scanned, or otherwise recreated These activities can be very costly. Consider putting a note in the documentation that states what tools, and versions thereof, were used to produce the document during its development or modification history.

3.3.6 How Can I Use the Templates In This Volume?

Document templates are a form of tool you can use to facilitate development of consistent software documentation. Chapter 4 of this volume provides templates that can be used as guides to produce the most common types of software documentation. The templates are primarily used by higher risk projects as guided by the process and selection criteria previously described in Figures 3-1 through 3-3.

In addition to this hard copy volume, the templates described in Chapter 4 and a number of document examples (not always in the format suggested by the template) are available on electronic media, generally in the originally created word processor format. For any current updates to the templates and examples, the reader is referred to these electronic media. See the end of section 1.4 for information on how to obtain the electronic media.

3.4 Documenting Existing Software

In some cases a software program is operational but appropriate documentation either does not exist or must be upgraded to be effective. The primary reasons for documenting existing software are the same as for documenting software being newly developed: appropriate documentation improves understanding for operational use and for modification to correct defects, add enhancements, and adapt to operational or support environment changes. This subsection describes some methods for documenting requirements, design information, and test cases and test results from existing software. A general context for these methods is then described in terms of reverse engineering software and software re-engineering. In addition, there is a description of some pitfalls to avoid when documenting existing software.

3.4.1 Documenting Requirements

Often situations arise in which an existing software system must be upgraded to correct existing problems, provide system enhancements, or convert from one hardware platform to another. In these cases, it may be necessary to create requirements documentation from scratch or update existing system products. Some possible methods for capturing and documenting software requirements under these conditions include:

- conduct questionnaire survey or poll users;
- interview current users;
- derive requirements from existing software source, execution of operational software, and other possible documentation; and
- reverse engineer design and requirements information from source code.

Frequently it is useful to combine several of these methods to accomplish the necessary capture of the requirements information. Documenting the requirements can be accomplished by mapping the results into the sections of an appropriately tailored Software Requirements Specification as described in section 4.1. Traditional requirements gathering techniques are usually necessary even when documentation is current and correct. As an example, the software may perform exactly as the documentation describes and may execute as expected 100 percent of the time; however, the software still may not do what the customer requires of the software. Traditional techniques such as polling and interviewing can help.

When there are a large number of users (customers) or the users are geographically dispersed, it may be necessary first to poll the users to identify key aspects of the software and its use. The polling mechanism, such as a questionnaire or survey, can establish a general view of the software use, its primary customers and their satisfaction, and the functional requirements. With a large number of users the results might be analyzed using statistics. The polling information might also identify more experienced users, the functional distribution of software use among the users, and what issues need to be addressed in more detail through the interview technique described in the next paragraph.

Interviews of current users can be conducted in a one-on-one setting in serial fashion to capture what an existing software system is required to do. Often the interviewer is responsible for reconciling inconsistencies across interviews. This responsibility is time consuming, filled with opportunity for errors, and often frustrating. A better approach to interactive planning and requirements gathering is to convene developers and their customers for an intense co-development (or in our case co-redevelopment) of the software. In this approach, developers and customers become mutual partners in the requirements gathering (and defining) process and exchange the serial process of interviewing for a parallel process of joint system development. This is most effective when there are only a few knowledgeable users, developers, or maintainers who need to be involved in the interactive process. Functions, performance timing and response, and user interface specifications can be determined and documented in appropriate sections of a traditional requirements documents.

An advantage to documenting the requirements of existing systems is the presence of source code and potentially other documentation that may be used to supplement requirements gathering techniques associated with software development. It is an advantage to have an operational software system when trying to derive its requirements -- assuming the software system satisfies its current or expected customers. However, existing documentation is often outdated, and determining which pieces of the documentation are current may require significant time investment. Also, it may be necessary to use reverse engineering methods to derive basic design information from the software source code in order to begin a detailed derivation of the software's requirements. Existing documentation from which requirements can be captured may come in many forms. A few are described below:

- Software development standards/guidelines/practices may also provide insight regarding how the software was developed, what tools normally would have been used, and what documentation deliverables might be available. Within the Sandia Software Guidelines, these management documents are referred to as the Software Development Plan, the Software Quality Assurance Plan, and the Software Configuration Management Plan.

- Development documentation such as requirement specifications, design criteria, and test plans, while least likely to be found or be current, may provide many details for components of the system where few changes have been introduced. Within the Sandia Software Guidelines, these development documents are referred to as the Software Requirements Specification, Software Design Description, and the Software System Test Plan. There also may be individual software development folders from which lower-level module requirements can be derived.

- Graphical models such as Data Flow Diagrams, State Transition Diagrams, Entity-Relationship Diagrams, Association Matrices, Action Diagrams, and others are documentation sources.

- Customer User Guides, while providing a less technical description of the system, may offer a roadmap for system functionality.

- Corporate policy and business practices describe the current business environment; these should be consistent with engineering and business software.

Existing systems have executable code, and typically the source code as well. If the source code is consistent with the executable code, both the source code and any current comments are a likely source of documentation for use in capturing requirements through reverse engineering or re-engineering methods. This option is discussed in more detail in section 3.4.4.

3.4.2 Documenting Design Information

Unfortunately, there are situations in which software code is written but design documentation for that software either does not exist or is insufficient for maintenance activities. In such cases, the use of manual or automated methods may be useful to derive graphical models such as Data Flow Diagrams, State Transition Diagrams, Entity-Relationship Diagrams, Association Matrices, Action Diagrams, and other design documentation information.

If the only existing part of the software is executable code there may be an existing support tool to de-assemble the machine code into an assembler code appropriate to the computational processor. Once the software is in assembler code there may be support tools to construct basic flow diagrams that identify control and data information. Whether the assembler code groups into recognizable modules will depend on the application. The next step would be to try and identify the functional module groups from the entry points and assembler jumps. This decomposition would provide the basic design functional hierarchy and together with the flow diagrams could be the basis for the software's design information. Without the indicated support tools, it may be more useful to consider a re-engineering effort of the software.

If software source code exists in a higher order language than assembler code, then it is an important documentation (and verification) activity to recompile the software and determine if the recompiled software execution agrees with the current executable software. This verification is necessary in order to preclude wasting time deriving design information from source code that does not agree with the executable code. Results of the verification activity, the compilation information, and any symbolic execution traces should be documented in a Software System Test Plan and appropriate software development folders (see section 5.3). If there is confidence that the source code matches the executable software product, then additional methods can be applied to derive design information.

Reverse engineering tools exist for most common higher order languages (e.g., Ada, C++, C, Pascal, FORTRAN, COBOL). These tools accept source code as an input, organize the source code modules into a library-like internal format, and provide design information as an output. Design information includes such information as a data dictionary, call forward and backward tree chains, data flow diagrams, control flow diagrams, and individual module design templates with entry points, parameter variables, and exit points identified. This information is usually available in written reports, on-line files, or repeatable user-queries. In addition, the tool capability often exists to group common design information such as the identification of all locations where a variable is used (and its use by reference or modification). Such information can be used for further reverse engineering and documentation activity. For example, once the indicated design information is available, other design methods can be used to represent the design information (and requirements information) in a standard format appropriate to the organization. This information can then be the basis for a complete re-engineering effort or simply for supporting the current software operation and maintenance. Without the automated tools a manual approach can be used to identify the most critical software modules and construct higher level versions of the same design information as indicated above. Once the design information is derived the information can be captured in a written form such as a Software Design Description or an electronic media format appropriate to the organization standards and available automated tool support.

3.4.3 Documenting Test Cases and Results

Assuming the existing software includes executable software, it is very useful to document what the software does by executing a series of test cases. The test case setup, description, inputs, outputs, and any failures become the basis for satisfying other documentation needs. For example, the test cases and results can be documented in an appropriately tailored Software System Test Plan. The test cases become the basis for specifying the software requirements that can then be documented in an appropriately tailored Software Requirements Specification.

The focus of the test cases should be on the performance of user tasks and the use of externally observable inputs and outputs while the system containing the software is exercised in an operational-like environment. During the execution of the test cases it would be very valuable to capture what part of the software is exercised (e.g., paths executed). Automated software tools and module instrumentation methods are available to trace the paths in the software that are exercised during system execution. From this test path trace information, static analysis can be conducted to identify which non-exercised parts of the software may not be required. Further test cases can be designed for unit and other specialized test environments to confirm the function of the non-exercised software paths. Documenting this correspondence between test results and existing software code provides insight into what the software requirements specification should be. Through this process it should be possible to identify those parts of the software whose failure is critical to the software's performance. More exhaustive testing of critical software components could then be conducted. Other criteria for identification of further tests might be modules that have a high number of control branches or implement a complex algorithm.

The Software System Test Plan that results from these efforts becomes the baseline for future regression testing as the software is modified. Other software documentation developed as a result of the testing process should also be updated as needed.

3.4.4 Reverse Engineering Software Documentation

The term reverse engineering means to build products normally constructed during an earlier life cycle phase from products and other information available from a later life cycle phase. If design documentation does not exist for a software product but source code does exist, then to create design information from the source code (and perhaps information available from available personnel or operational use) would be an example of reverse engineering software documentation. The reverse engineering process may result in a graphical model or a textual representation of the product. Reverse engineering is often used in conjunction with forward engineering techniques to either verify the reverse engineering process or reengineer the existing product.

Reengineering is the reconstitution of a product either from the combination of reverse and forward engineering or altering of a product within the same phase of the software life cycle. Code that is restructured without generating data or process models is an example of re-engineering within the same phase of the life cycle. Code re-structuring tools exist with varying capabilities. The capabilities range from a simple physical rearrangement of the code in accordance with style guidelines to graphically depicting process flows and regenerating the code using updated syntax and coding structures. The capability to re-design the system or add functionality is not typically part of this class of tools.

Keeping detailed design information current as development proceeds can be a tedious job. The detailed design information may be considered extremely important when modifications to the software are considered either later in the initial development or during maintenance. Unfortunately, this design information is useless, even harmful, if it is inaccurate. A reverse engineering tool can be used to advantage by deriving as-built design information and facilitating the comparison of this information to the as-built code.

The following are desirable characteristics of a reverse engineering tool:

- the tool should generate graphical models which the software engineer can manipulate;
- the tool should generate data and process models; generating the data model is far easier than generating process models yet much of the work required to re-engineer software is process-driven;
- the tool should allow or facilitate forward engineering of modified models (preferably within the same suite of tools);
- the tools should generate the same level of code that was submitted for reverse engineering; some code generators only generate code for the data in the system; other generators produce skeletal programs which the engineer is responsible for completing; higher level generators produce complete source code; upon completion, a re-engineering tool should not transform a source program into any less functional form than what it was; and
- the tool should populate a data dictionary, repository, or encyclopedia, depending on the sophistication of the tool; a data dictionary might contain data names, sizes, types (of data), descriptions, and edit criteria; repositories and encyclopedias are usually perceived as having greater capability capturing process and sub-process models, where-used information, and project planning criteria.

The use of reverse engineering to generate as-built software documentation has certain advantages. However, it can also lead to dangerous development practices. The processes and tools used to develop documentation for existing systems require the same rigor required of products when originally engineered. This rigor implies that processes and tools be consistent, complete, and correct. Training and knowledge in the software application and the techniques selected are prerequisites to success. Also, a software project should not decide that it can skip documentation of a development phase since it can later reverse engineer the needed documentation from source code. A design phase serves a larger purpose than just to produce a Software Design Description. More information about reverse engineering and reengineering software documentation is contained in references [ARNOLD] and [WCRE].

3.4.5 Some Potential Problems to Avoid

An abbreviated discussion of some possible pitfalls in creating documentation for existing software is contained in the following paragraphs.

Limitations of Tools

Some tools are independent of any particular methodology and hence make enforcement of a project's specific methodology very difficult. Many tools are totally dependent on one or a few specific methodologies. This dependence requires that the project use one of the methodologies supported by the tool. This dependence may limit the type and form of software documentation that can be created. The likelihood that a software engineer will find a tool, outside an integrated environment, that aligns precisely with a project's development methodology is rather low. Empirical evidence suggests that the market for documentation tools is reactive rather than proactive. That is, a problem exists for which a product is developed. Ensure that the tools and techniques selected do indeed match the documentation problem to be solved and the needs of your development and maintenance processes.

Reliance on Tools

Depending on the tools available, different levels of sophistication are often required by the software engineer. All tools have strengths and weaknesses. An unfortunate side effect of automated software tools is the perception that the tool can be used by almost anyone with little or no formal training. The opposite is typically much closer to the truth. Sophisticated tools generally require sophisticated users who can differentiate among the features of the tool and the needs of the project. Understanding the terminology of the processes being used by the project will greatly increase the likelihood that tools can be selected with capabilities that support the project processes and with the appropriate level of software engineering sophistication. Often vendors will offer to demonstrate their solution on a piece of an existing system. Vendors may accommodate an in-house evaluation for the cost of shipping and training. Maximize the use of both these types of opportunities.

Reliance on Existing Documentation

Documentation that has not been maintained is defective documentation. Approximately 25 percent of all software defects have been identified as being related to the documentation. While these guidelines suggest plans and documents to ensure the supportability of the software, including the documentation, reality may be different. Prevention, through diligent upkeep, remains the best prescription. The use of Computer-Aided Software Engineering tools that require changes to data and process models and not to source code is a practical way to ensure software product documentation is current.

Emerging Technologies

As certain as change is itself, so is the dating of this section by emerging technologies. Object-oriented techniques, client-server architectures, and network communication advances promise to rewrite how software is developed and supported. The use of networks and electronic media to communicate project and software documentation information will have a major impact on the form and content of software documentation. The automation of software development will have a powerful impact on what software information is of value for both the developer and supporter. The distinction between software development and support will be in the way integrated software engineering environments are used to manipulate software documentation (representation) information.

Software engineers can now develop good or poor systems faster than ever. Despite these advances, software development productivity has not soared. Relying on tools and technology to solve scientific, engineering, and business applications avoids the challenge of re-using and re-engineering application practices. The most significant advances in productivity still await the maturation of how software information can be efficiently represented for reuse.

3.5 Are There Methods to Improve Documentation Quality?

3.5.1 Documentation Quality Characteristics

Throughout this guide there are descriptions of what each type of document should contain and some hints on which processes produce the documentation. However, what characterizes the quality of documentation? Documentation can make everyone's (developer, customer, and supporter) software-related tasks much easier or harder -- depending on the quality of the documentation. Some of the more important documentation quality factors are:

- **correctness:** the extent to which documentation is free from defects;
- **maintainability:** the ease with which documentation can be modified for software corrections, functional and performance enhancements, and adaptations to changes in the system environment;
- **readability:** the ease with which documentation is understandable to the reader;
- **simplicity:** the extent to which documentation is free from extraneous information, cumbersome organization, and complex descriptions of the software product; and
- **usability:** the extent to which documentation can be used to facilitate the operation and maintenance of the software.

3.5.2 Documentation Review Methods

One of the easiest ways to ensure documentation quality is to review the documentation for defects relative to quality factors such as listed above. The lower level characteristics and measures that precisely define each of the quality factors will depend somewhat on the software application domain and the documentation product being reviewed. There are several methods of review:

- **informal review:** someone else in your group looks over the documentation to see if it has problems;
- **walk through:** similar to an informal review except one leader typically guides the discussion or presentation, while the reviewers listen, ask questions, and provide suggestions;
- **technical (peer) review:** several technically competent people review the documentation and then hold a semi-formal meeting to discuss the document and what they have found;
- **formal review:** specific reviewer roles are defined and process steps followed; documentation defects are identified, recorded, classified, and corrected using well-defined procedures; the documentation is signed-off and released into configuration management control; Sandia provides training on this method of review which is also called a software inspection;
- **inspection:** same as formal review; and
- **audit:** an independent agent checks the as-built documentation for compliance of the format, content, development process, and/or perhaps the verification of any processes associated with the review or production of the documentation; compliance may be to external or internal organization standards and guidelines.

The extent of the documentation review is dependent on the formality of the software process to be followed on the project. If the software project is to develop a 200 source line program that will be used only within the organization, then the documentation need not be extensive nor require much more than an informal review. If the project is a Sandia-wide software project or is a weapons-related software project, then a more formal process needs to be followed for documentation development and review to ensure the appropriate level of quality.

3.5.3 Documentation Inspection Process

Software inspections are formal peer reviews of software documentation such as requirements specifications, design descriptions, test plans, and source code. Any product generated during the software life cycle can be inspected. Software inspections are formal in that there is a set agenda. A limited amount of material is covered in a set amount of time. Those attending the review have clearly defined roles. All issues raised at the inspection are rigorously recorded and resolved. Software inspections are peer reviews that management should not attend. This restriction allows those in attendance to concentrate on analyzing the material at hand rather than getting involved in political maneuvering that might occur if supervisory personnel were present. It also eliminates a human tendency to look to management for leadership and approval. The software inspection process has output that is also a part of the software's documentation.

A software inspection's goal is to find defects by evaluating whether the software documentation products conform to quality-related factors. Characteristics that are checked include: lack of ambiguity, verifiability of requirements, correctness of logic and data information, completeness of the information, adherence to documentation standards, and consistency within the product being inspected and with other products . This technique can be applied to text, graphical information, and source code. It can be used to examine source listings in any language, text-based documentation, and specifications that use formal languages. It is this broad applicability that makes software inspections such a powerful tool to employ during software development and support.

4 Document Guidelines: Format and Content

The elements and content of a software documentation set are dependent upon the type and complexity of the software to be developed. A project to develop a database to track office supply inventories within a department is vastly different than a project to provide the software for use-control functions of a nuclear weapon. Some documentation is recommended as part of every software project no matter how small or large. Small software projects may expand into programs that are used lab-wide or, through technology transfer, are used by external organizations. Documenting the software processes and resulting products for a project is also a recognition that developers cannot always remember everything and may not even be around to maintain the programs they create. As a result, development projects need to select the types and extent of documentation that will produce a reasonable set of documents. Approaches to selection of software documents for a project are provided in Chapter 3 and its associated Appendix D.

This chapter describes the key software documents from which a project document set can be selected. Specifically, this chapter establishes guidelines for the development of key software documents. Each section describes a single document from the following points of view and provides an outline identifying the potential contents of that document:

- **Abstract** - includes purpose of the document and an overview of its contents;
- **Audience** - identifies the potential users of the document and how it might be used;
- **Responsibilities** - describes who is responsible for writing, reviewing, and approving the document;
- **Content** - describes the requirements for and typical content of the document, considerations relative to its production, and relationship of the document to the project's software development activities;
- **Maintenance** - describes considerations relative to keeping the document current, who is responsible for such efforts, and an overview of how the maintenance is to be performed; and
- **Other Alternatives and Issues** - describes additional considerations relative to creation, maintenance, and distribution of the document.

Sometimes, the same kind of information is called out in the content descriptions of more than one of the key software documents. Only one project document should include such information, while other project documents should reference this information. In addition to the suggested content outline, each document should have a cover page, sign-off page to indicate initial release responsibility, change page for recording document update information, table of contents, and list of illustrations as appropriate. The control version of each document should be clearly indicated. Development versions can be controlled through a date located on each document page. Release versions can be controlled through an identification scheme such as required for War Reserve software (see references [EP401040] and [EP401045]). For each document described in this chapter there is a corresponding thematic template available on electronic media (see subsection 1.4) that provides an expanded set of guidelines for completing each section of the document. The documents addressed in this chapter are summarized in the table below.

Section	Document Title
4.1	Software Management Plan (SMP)
4.2	Software Development Plan (SDP)
4.3	Software Quality Assurance Plan (SQAP)
4.4	Software Configuration Management Plan (SCMP)
4.5	Software Requirements Specification (SRS)
4.6	Software Design Description (SDD)
4.7	Software System Test Plan (SSTP)
4.8	Software Support/Maintenance Plan (SSMP)
4.9	Software Safety/Security Plan (SSP)
4.10	Software Users Guide (SUG)
4.11	Software Verification and Validation Plan (SVVP)

4.1 Software Management Plan (SMP)

4.1.1 Description

An organization's Software Management Plan (SMP) may be implemented using a multi-tiered approach. This approach allows for definition of software management policies and strategies at the organizational level (see reference [SLP1011]) while accommodating the specific management approach for the types of software acquired, developed, used and supported at the project level. The software managed by an organization is identified in that organization's SMP.

4.1.1.1 Abstract

The purpose of a Software Management Plan is to describe how an organization manages its software resources consistent with: Sandia Laboratories Policy 1011 [SLP1011]; Engineering Procedure 401045, Definition of Computer Software Configuration Items [EP401045]; and DOE Order 1330.1D, Computer Software Management [DOE 1330.1D]. A SMP, in conjunction with other plans such as the Line Practices for Environmental Safety & Health (Conduct of Operations), software standards, and project plans constitute the software management practices for that organization. An organization may be a specific Sandia line element (e.g., center, department), a combination of such line elements, a functional area that cuts across line elements to accommodate a specific functional need (e.g., information systems, nuclear weapons), or an individual project.

A Software Management Plan is used to both document and guide an organization's software engineering environment. It addresses management of the software life cycle with respect to development, maintenance, customization, cost, release, security and safety, software sharing and reuse, and software management cost factors. In addition, it describes the methods, tools, and techniques to be used to ensure that an organization's software assets are properly acquired, developed, used, and supported. More detailed information and specific project variances may be provided in a software project's selected document set.

Preferably, an organization would first develop a Software Management Plan that represents their current software engineering management practices. Through a continuous software process improvement effort, the SMP would evolve to document the improved software engineering management practices.

4.1.1.2 Audience

The Software Management Plan is intended for the following audiences:

- **Software Program Management** - provides an overall plan for conduct of software projects while meeting the goals and standards for a specific organization, project types, and customers;
- **Software Development Personnel** - provides conventions, methodologies, and techniques approved for use on software projects within the organization; as such, it also provides a ready reference for use by software personnel who are new to the organization or an on-going project;
- **External Customers** - provides clarification concerning software management approaches used on projects by the development organization; and
- **Auditors** - provides a basis against which compliance with DOE Order 1330.1D and Sandia Laboratories Policy 1011 may be verified.

4.1.1.3 Development, Review and Approval Responsibilities

An organization's management is responsible for allocating the resources necessary to develop and maintain a Software Management Plan. The SMP should be written by members of the organization who are familiar with the practices and conventions used by that organization's software personnel. The organization's management and senior software engineers should review and recommend changes to the draft plan. The SMP should be approved by the head of the organization covered by the plan; e.g., the center director, department manager, or project leader. It is important to establish interfaces with other organization personnel and ongoing activities while developing the SMP including:

- project leaders to ensure coverage with software-related projects already underway;
- conduct of operations management to eliminate redundancy with software processes; and
- strategic planners to anticipate future directions.

4.1.1.4 Content Guidelines

The Software Management Plan describes the overall approaches to development, documentation, and maintenance of software within the organization. The plan addresses management of software programs, life cycles, release, and security.

4.1.1.5 Maintenance

The Software Management Plan defines the current software engineering practices within the organization. Conceivably, these practices will continue to change as technology and software improves. The organization covered under the SMP needs to review and modify the SMP at least on an annual basis. Adoption of new methodologies, preferred processes, tools, and techniques, as well as the release of new software products, may occur more frequently. Any substantial changes to the operation or development environment need to be incorporated into the SMP as soon as it is reasonably possible.

4.1.1.6 Other Alternatives and Issues

The Software Management Plan may consist of a paragraph or section in the organization's Conduct of Operations.

4.1.2 Template Outline

An outline of a Software Management Plan, based on reference [SLP1011], is illustrated in Table 4-1.

Table 4-1. Software Management Plan

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms, and Abbreviations
   - 1.4 References
2. Implementation
   - 2.1 Plan Dependencies
   - 2.2 Assumptions and Responsibilities
3. Program Management
4. Life Cycle Management
5. Inventory Management
   - 5.1 Function
   - 5.2 Degree of Customization
   - 5.3 Cost
   - 5.4 Degree of External Impact
6. Release Management
7. Security Management
8. Software Sharing
9. Software Process Improvement
10. Software Management Cost Factors

4.2 Software Development Plan (SDP)

4.2.1 Description

The execution of most software projects should be based on a set of practices, procedures, and processes defined by management for use by team members during the software development efforts. A properly prepared Software Development Plan conveys these management controls and a project schedule. The schedule itself should be based on milestones and deliverables (documents and products) and a work breakdown structure defined to achieve the development schedules and meet the deliverable dates. This section does not specify any particular techniques to be used during the software development process but does provide a template as an aid for preparing the SDP.

4.2.1.1 Abstract

The purpose of a Software Development Plan is to document what methods and techniques will be used to manage a software project. The SDP defines the technical and managerial processes employed during a project, a detailed schedule including milestones and deliverables, a detailed budget, a categorization of the personnel resource needs and distribution as a function of time, the support ènvironment and tools to be used, and the perceived risks relative to the schedule and technical hurdles. The SDP provides a detailed refinement of the applicable Software Management Plan to a specific project. The SDP should be a living document, continually updated so as to provide current information as well as comparisons with the previously generated project estimates.

4.2.1.2 Audience

The Software Development Plan is intended primarily for the following three audiences:

- **Software Project Management** - provides an overall plan and schedule for the implementation of software development processes, utilization of resources and tools to be used, and identification of potential problems that may be encountered; the SDP also serves as a basis against which to measure progress during the software development effort;
- **Software Development Personnel** - software designers, programmers, analysts, testers, and maintainers may benefit from the overall program picture provided by the SDP in the form of clearly defined goals, expected work products, team interface coordination, standards to be followed, and tools that are planned for use; and
- **External Customers** - the SDP will give customers an estimate of how and when project goals will be met and whether the project risks have been properly identified and addressed.

4.2.1.3 Development, Review, and Approval Responsibilities

Software project management is responsible for preparing an appropriate Software Development Plan for the project; however, managers cannot do it in a vacuum. The responsibility for writing the SDP is a combined effort of the software project manager and the software developers knowledgeable of the software technologies and resources necessary to prepare the required product for a customer within projected budgets. The software manager or project manager should be responsible for sign-off of the SDP to indicate the initial release. Once the Software Development Plan is reviewed/inspected and the sign-off process completed, the SDP should be placed under project configuration management control.

4.2.1.4 Content Guidelines

A Software Development Plan typically contains the following information:

- technical activities/tasks;
- management activities/tasks;
- activity/task descriptions;
- product qualification and control;

- resource assignments;
- schedule and schedule dependencies;
- major technical and schedule risk areas; and
- evolution and working document concepts.

4.2.1.5 Maintenance

The Software Development Plan is a living document, representing the evolution of the software project from the planning information to as-built information. Revisions to the Plan should be published periodically throughout the software development phase, particularly if the phase lasts for a year or more. The Plan necessarily tracks changes to the estimates and events/reasons for the changes. The SDP is controlled within the guidance of the Software Configuration Management Plan.

4.2.1.6 Other Alternatives and Issues

The Software Development Plan also documents lessons learned in the evolution of the plan and the project. The Software Development Plan may contain references to separate, more detailed plans such as an SQAP or SCMP. This is particularly true for large projects, where configuration management and quality organizations typically require more resources.

4.2.2 Template Outline

An outline of a Software Development Plan, based on the Software Development Plan in reference [IEEE1498], is illustrated in Table 4-2.

Table 4-2. Software Development Plan

1. Introduction
1.1 Identification
1.2 System Overview
1.3 Document Overview
1.4 Relationship to Other Plans
2. References
3. Overview of Required Work
4. Plans for Performing General Software Development Activities
4.1 Software Development Process
4.2 General Plans for Software Development
5. Plans for Performing Detailed Software Development Activities
5.1 Project Planning and Oversight
5.2 Establishing a Software Development Environment
5.3 System Requirements Definition
5.4 System Design
5.5 Software Requirements Definition
5.6 Software Design
5.7 Software Implementation and Unit Testing
5.8 Unit Integration and Testing
5.9 Software Item Qualification Testing
5.10 Software/Hardware Item Integration and Testing
5.11 System Qualification Testing
5.12 Preparing for Software Use
5.13 Preparing for Software Transition
5.14 Software Configuration Management
5.15 Software Product Evaluation
5.16 Software Quality Assurance
5.17 Corrective Action
5.18 Joint Technical and Management Reviews
5.19 Risk Management
5.20 Software Management Indicators
5.21 Administrative Security and Privacy Protection
5.22 Managing Subcontractors
5.23 Interfacing With Software IV&V Agents
5.24 Coordinating With Associate Developers
5.25 Project Process Improvement
6. Schedules and Activity Network
7. Project Organization and Resources
7.1 Project Organization
7.2 Project Resources
8. Notes
Appendices

4.3 Software Quality Assurance Plan (SQAP)

4.3.1 Description

The purpose of a Software Quality Assurance Plan is to document the methods and techniques that will be used to verify and validate that software products have the requisite quality characteristics necessary to satisfy customers and any regulations and standards applicable to a specific software project.

4.3.1.1 Abstract

The Software Quality Assurance Plan for a project includes identification and descriptions of: reference documents; management structure and responsibilities; software documentation products to be developed and how they are to be checked for accuracy; standards, practices, and conventions that are applicable to the project and its products; reviews and inspections that are to be performed; tools, techniques, and methodologies that will be utilized by the project team; configuration management and controls; records collection, maintenance, and retention requirements; and verification methodologies that will be applied.

4.3.1.2 Audience

The Software Quality Assurance Plan is intended primarily for use by the following four audiences:

- **Software Project Management** - provides an overall plan for ensuring that approved software quality techniques are being applied throughout the software project;
- **Software Development Personnel** - provides an identification of quality standards expected for the software products and documents to be produced; as such, it is a reference for use by software personnel who are new to the organization or an on-going project;
- **External Customers** - provides clarification concerning the software quality standards that are to be enforced by the development organization; and
- **Auditors** - provides a basis against which compliance with software quality assurance standards and requirements may be verified.

4.3.1.3 Development, Review, and Approval Responsibilities

Software project management is responsible for chartering and allocating the software project or Quality Department personnel resources necessary to develop and maintain a SQAP. Responsibility for developing the Software Quality Assurance Plan rests with the organization developing the software, preferably someone familiar with software quality assurance techniques. Software engineers, their management, and when appropriate their customers, review the plan. The SQAP is approved by the head of the organization developing the software; e.g., center director, department manager, or project leader. The plan assigns responsibility for its implementation, required resources, scheduling, and distribution.

Several interfaces to other ongoing activities within the organization are encouraged while developing the SQAP including:

- project leaders to ensure consistency with system development projects already underway;
- organization managers to ensure consistency with other software projects and eliminate unnecessary redundancy among them; and
- strategic planners to anticipate and incorporate future software development/documentation concepts and technology/environment upgrades in the SQAP for an ongoing project.

The Software Quality Assurance Plan should be developed and approved during the initial period of the software project. Once the SQAP is reviewed/inspected and the sign-off process completed, the SQAP should be placed under project configuration management control.

4.3.1.4 Content Guidelines

The reference [SSGv1] provides guidelines for the content of a Software Quality Assurance Plan based on tailoring of the plan in reference [IEEE730]. The following information should be provided in the SQAP:

- identification of software product to which it applies;
- standards, regulations, and measures that provide quality criteria to be satisfied by the software products;
- project organizational structure and personnel responsible for conducting the quality assurance activities;
- software product reviews and the review mechanism (e.g., software inspections) that will be used;
- software control mechanisms for project information, problem reports, and deliverable products;
- project records collection and maintenance information;
- project management and technical training plans; and
- identification of areas of risk and possible strategies to mitigate the risks.

4.3.1.5 Maintenance

The Software Quality Assurance Plan should reflect the current software quality practices within the organization. Conceivably, these practices will change during an on-going project as technologies and methodologies improve. Consideration should be given to the following throughout the development and maintenance of software:

- identifying options for changes to the plan;
- recommending specific changes;
- reviewing changes;
- implementing approved changes; and
- managing the release of those changes.

4.3.1.6 Other Alternatives and Issues

When an independent quality organizations is used for qualification of the software product, the SQAP can be an input to the qualification process but generally will not be a substitute for the qualification plan. The SQAP information may be included in the Software Development Plan. Project SQAPs may be tailored from an organization level SQAP. See reference [SSGv1] for more information on SQAPs.

4.3.2 Template Outline

An outline of a Software Quality Assurance Plan, based on reference [IEEE730], is illustrated in Table 4-3.

Table 4-3. Software Quality Assurance Plan

1. Introduction
    1.1 Identification
    1.2 System Overview
    1.3 Document Overview
    1.4 Relationship to Other Plans
2. References
3. Management
    3.1 Organization
    3.2 Tasks
    3.3 Responsibilities
4. Documentation
    4.1 Purpose
    4.2 Minimum Documentation Requirements
5. Standards, Practices, Conventions and Metrics
    5.1 Purpose
    5.2 Content
6. Reviews and Audits
    6.1 Purpose
    6.2 Minimal Requirements
7. Test
8. Problem Reporting and Corrective Action
9. Tools, Techniques, and Methodologies
10. Code Control
11. Media Control
12. Supplier Control
13. Records Collection, Maintenance, and Retention
14. Training
15. Risk Management

Appendices

4.4 Software Configuration Management Plan (SCMP)

4.4.1 Description

Software is subject to constant change during its life cycle. Such changes are normally due to changes in user requirements, defects discovered in code, or changes in technology such as hardware or software support upgrades. Establishing appropriate controls is critically important to ensure that all affected configuration items are corrected according to established, approved, change-control procedures. These controls are described in a Software Configuration Management Plan (SCMP).

4.4.1.1 Abstract

Configuration management procedures are established at the beginning of the software development cycle. Procedures include identification, control, reporting, and auditing of software configuration items. Configuration items include elements such as specifications, design documents, test documentation, and all software code and tools whether purchased or developed.

4.4.1.2 Audience

The Software Configuration Management Plan is intended primarily for the following five audiences:

- **Software Project Management** - provides an overall plan for the conduct of software configuration management efforts to support the establishment and management configuration baselines and the procedures that will be followed to control changes to those baselines;
- **Software Development Personnel** - software engineers, programmers, and are provided with a source of information about the tools and methods that will be used to perform the configuration management functions; of particular interest are the policies and procedures that will be followed in reporting and tracking problems and managing control of changes;
- **External Customers** - provides clarification concerning the approaches that will be used to manage the configuration of the software and identification of the configuration baselines that will be established and controlled; of particular interest are the policies and procedures that will be followed in reporting and tracking problems reported in the operational use of the software by the customer;
- **Software Configuration Management Personnel** - provides software configuration management personnel with a reference that identifies the policies and procedures that they will follow in carrying out their functions; discussions will also address subjects such as problem reporting and tracking, configuration control boards, software libraries, control and updating of library documents, configuration management tools and support facilities, version builds, and so forth; and
- **Software Quality Assurance Personnel** - software quality assurance personnel will have access to the information necessary to measure the effectiveness, accuracy, and consistency of the software configuration management and change control processes.

4.4.1.3 Development, Review, and Approval Responsibilities

Usually an individual familiar with configuration management procedures writes the SCMP. This person may be a member of the software development team or quality assurance. The project leader reviews and approves the SCMP. In some cases, customer approval may also be required. The project leader, customer representative(s), SCMP author(s), and software quality personnel sign the SCMP after final review/inspection. Once the sign-off is complete the SCMP is placed under project configuration management control.

4.4.1.4 Content Guidelines

Configuration Management establishes a comprehensive set of activities that are intended to capture the essence of the design and its definition at any point in time. Configuration Management also provides a basic framework that facilitates change approval, implementation, and tracking throughout the life of the product. The major components of a Software Configuration Management Plan include:

- overview of plan;
- structure of the configuration management organization and membership of the Configuration Control Board;
- policies and requirements relative to software configuration management and control; and
- identification of configuration management tools and methodologies.

4.4.1.5 Maintenance

Once the Software Configuration Management Plan is approved by program management it should not significantly change during the course of the project. Should changes be required, section 6 of the Plan provides the necessary steps.

4.4.1.6 Other Alternatives and Issues

The Software Configuration Management Plan may also be incorporated as part of the Software Quality Assurance Plan or, for smaller projects, as part of the Software Development Plan. Project SCMPs may be tailored versions of a general organization level SCMP. See reference [SSGv4] for more information on SCMPs.

4.4.2 Template Outline

An outline of a Software Configuration Management Plan, based on reference [IEEE828], is illustrated in Table 4-4. The guidance in references [IEEE1042] and [SSGv4] is very helpful.

Table 4-4. Software Configuration Management Plan

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions and Acronyms
    - 1.4 References
2. Management
    - 2.1 Organization
    - 2.2 SCM Responsibilities
    - 2.3 Applicable Policies, Directives, and Procedures
3. SCM Activities
    - 3.1 Configuration Identification
        - 3.1.1 Baselines
        - 3.1.2 SCM Libraries
    - 3.2 Configuration Control
        - 3.2.1 Software Change Request/Requesting Changes
        - 3.2.2 Functions of the Configuration Control Board
        - 3.2.3 Evaluating and Implementing Changes
    - 3.3 Configuration Status Accounting
    - 3.4 Audits and Reviews
    - 3.5 Interface Control
    - 3.6 Subcontractor/Vendor/Supplier Control
    - 3.7 Records Collection and Retention
4. Schedules - SCMP Implementation
    - 4.1 Configuration Control Board
    - 4.2 Baselines
    - 4.3 Change Control
    - 4.4 Status Reporting, Reviews and Audits
5. Resources
    - 5.1 Tools, Techniques and Methodologies
    - 5.2 Personnel
    - 5.3 Training
6. Plan Maintenance

Appendices

4.5 Software Requirements Specification (SRS)

4.5.1 Description

The Software Requirements Specification is the document that captures the software product functional and performance specifications agreed to by the customer and the developer. The SRS is the most important document a developer produces and is the one most likely to impact the quality of software product items.

4.5.1.1 Abstract

The Software Requirements Specification defines the functions, performance, interfaces, and constraints to be implemented by a particular software product, program, or set of programs. In addition, it provides the basis for validating that the software satisfies the requirements for the end product.

4.5.1.2 Audience for the Software Requirement Specification

The Software Requirements Specification is intended primarily for the following seven audiences:

- **System Project Management** - provides system project managers with software-related descriptions of the system requirements that have been allocated to software for their satisfaction; this information is useful in evaluation of proposed changes to system and/or the hardware being procured or developed for use in that system;
- **Software Project Management** - provides software project management with detailed documentation of the specific requirements that must be designed, implemented, and tested; this information supports estimation of and planning for the time and resource allocations to complete the expected work;
- **Software Development Personnel** - provides software developers precise documentation of the software specifications and technical definitions for the software functions that are to be implemented;
- **External Customers** - provides clarification of and a basis for agreement on the software functions that are to be developed and implemented and the methods (demonstration, test, inspection, or analysis) by which satisfaction of each of the specified requirements will be validated;
- **Software Configuration Management** - provides software configuration management with the starting point for evaluating the accuracy of reported problems and the impact of proposed changes resulting from evaluation of those problems;
- **Software Test Personnel** - provides software testers with technical descriptions of each of the software requirements and the methods by which their satisfaction will have to be validated; and
- **Auditors** - the effectiveness of the system can be audited against known requirements prior to or after customer delivery.

4.5.1.3 Development, Review, and Approval Responsibilities

It is the responsibility of software analysts to derive the SRS from the system requirements defined by the customer. Software testers review the derived requirements for testability and understandability. A formal software requirements review (SRR) may be performed for the program management and the customer. The project leader, customer representative(s), and software quality assurance sign the SRS after final review/inspection. Once the sign-off is complete the SRS is placed under configuration management control in accordance with the Software Configuration Management Plan.

4.5.1.4 Content Guidelines

A Software Requirements Specification defines the functionality of and accept/reject criteria for the software to be developed. The SRS provides the basis for derivation of the design and development of the test descriptions. The basic content includes:

- General description of the software;
- Specific requirements such as those for:
  - ⇒ external interfaces to other software and hardware;
  - ⇒ functionality to be provided;
  - ⇒ performance;
  - ⇒ specialty attributes such as security, safety, reliability, and maintainability; and
  - ⇒ design constraints.

4.5.1.5 Maintenance

The Software Requirements Specification may need to evolve as the development of the software product progresses. It may be impossible to specify some details at the time the project is initiated and this may lead to prototyping. Additional changes may ensue as deficiencies, shortcomings, and inaccuracies are discovered during development of the prototype software product items. All changes to the requirements should be reflected in the SRS. Two major considerations in this process are:

- requirements should be specified as completely as possible, even if changes can be foreseen as inevitable; and
- a formal change process should be initiated to identify, control, track, report, and incorporate changes.

4.5.1.6 Other Alternatives and Issues

The requirements in a Software Requirements Specification may be explicitly stated by the user or may be allocated to computer software through a system requirements analysis process. A prototype is also a form of requirements specification and can be used to supplement, but not replace, an SRS. The developer would still be responsible for obtaining formal approval from the customer for requirements developed as the result of prototype efforts. An SRS, or equivalent, is required by reference [EP401045] for War Reserve software projects.

4.5.2 Template Outline

An outline of a Software Requirements Specification, based on reference [PPSD], is illustrated in Table 4-5. This format is a slight variation of the outlines provided in reference [IEEE830] that includes eight template options for section 3 of the SRS. The eight template options include section 3 organized by (version 1) mode, by (version 2) mode, by user class, by object, by feature, by stimulus, by functional hierarchy, and by multiple organizations. The mode refers to the way in which the software is used: for example, for training, normal use, emergency use, and so forth.

Table 4-5. Software Requirements Specification

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms, and Abbreviations
   - 1.4 References
   - 1.5 Overview
2. General Description
   - 2.1 Product Perspective
   - 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and Dependencies
3. Specific Requirements
   - 3.1 Functional Requirements
     - 3.2.1 Functional Requirement <10.00>
     - 3.2.2 Functional Requirement <20.00>
     - ...
     - ...
     - 3.2.*n* Functional Requirement <n.00>
   - 3.2 External Interface Requirements
     - 3.2.1 User Interface Requirements
     - 3.2.2 Hardware Interface Requirements
     - 3.2.3 Software Interface Requirements
     - 3.2.4 Communication Interfaces
   - 3.3 Performance Requirements
   - 3.4 Design Constraints
     - 3.4.1 Standards Compliance
     - 3.4.2 Hardware Limitations
   - 3.5 Attributes
     - 3.5.1 Security
     - 3.5.2 Maintainability
     - 3.5.3 Availability
     - 3.5.4 Transferability and Conversion
   - 3.6 Miscellaneous Requirements
     - 3.6.1 Data Base
     - 3.6.2 Operations
     - 3.6.3 Site Adaptation

Appendix 1: Glossary
Appendix 2: References
Appendix 3: Numbered List of Requirements
Appendix 4: Data Dictionary

4.6 Software Design Description (SDD)

4.6.1 Description

A Software Design Description contains the complete design for a major software component. Typically it describes that component in terms of the underlying software modules and lower level units. A SDD may also reference detailed design information that is maintained on electronic media. Multiple SDDs may be created.

4.6.1.1 Abstract

The purpose of a Software Design Description is to describe how program components are to be structured to meet the requirements specified by the Software Requirements Specification. It is the link between requirements and source code implementation. The SDD maps every requirement into one or more entities in the implementation. In this way the SDD shows how the requirements will be met. It may also be used to explain why certain design decisions were made. Typically the SDD employs both text and graphics to convey design information.

Two levels of design description documents are commonly written. These levels are often referred to as a top-level preliminary design and a low-level detailed design. The top-level design provides a description of the system as a whole and its major functional components and interfaces (data and control). The low-level detailed design includes specifications of individual functional components or objects. The detailed design is developed such that the implementation may proceed directly to source code with little additional analysis. Detailed design information can include program design language representations, a cross-reference of function calls, a listing of all global data in the program, which functions access global data, and definitions of user-defined data types. The Software Design Description can evolve to include both top-level and low-level design information, perhaps physically separated into two distinct documents.

4.6.1.2 Audience

The Software Design Description is intended primarily for the following three audiences:

- **Software Development Personnel** - software designers and quality assurance personnel will be able to trace software requirements to software design and implementation at the component level; software engineers (developers) will have specific knowledge of interfaces, functions, and inputs/outputs expected from each of the software components prior to actually writing software code;
- **Software Test Personnel** - software testers may use the design information to aid in the development of test cases; and
- **Software Maintenance Personnel** - Software maintainers may use the SDD to understand specific operations of the system.

In addition, the software design may be reviewed with the external customer. The SDD is a supporting document for the design review but is not usually reviewed directly with the customer. Software project management may find it useful to reference the SDD for more detailed information on the activities necessary to implement the software requirements. This information could permit better planning for allocation of time and resources to complete the software development task.

4.6.1.3 Development, Review, and Approval Responsibilities

The Software Design Description is derived from the requirements specification by the software design team. It is reviewed or inspected using the requirements specification as the basis against which it is verified. Systems engineers who are not directly familiar with the software design may also review the SDD. Such a review can further ensure that the SDD does not contain incorrect assumptions about how the software will support the system mission. The SDD may be formally reviewed with program management and the customer at a preliminary design review and at a critical design review during the appropriate stages of the design. The software project leader, key software design personnel, and a software quality assurance representative complete sign-off of the SDD after final review/inspection. When sign-off is complete the SDD is placed under configuration management control in accordance the Software Configuration Management Plan.

4.6.1.4 Content Guidelines

The level of detail included in the Software Design Description may vary according to the needs of the project and the stage of design development for individual modules. The modules for which only an initial preliminary design has been developed may only be documented in terms such as purpose and function. Later efforts in preliminary design may result in decomposition of such a module into lower level units and the documentation expanded to include additional top-level design information such as identification of subordinate units, dependencies and interfaces, and processing requirements. The detailed design may be documented in program design language or pseudo code. This level of detail will allow for direct transformation of the detailed design into programming language statements.

4.6.1.5 Maintenance

Typically the Software Design Description is placed under informal configuration management prior to a preliminary design review or inspection. The principal object of such control is to ensure that the design elements being reviewed or inspected are the latest version that have been developed. After completion of a detailed/critical design review or inspection, the SDD may be brought under more formal configuration control or it may be kept under informal control. The principal difference in these two levels of control is the formality involved in the analysis and approval of changes to the design during coding, module integration, and testing. In either control case, the software design team is responsible for seeing that the final version of the SDD accurately reflects the computer programs at the completion of formal software test efforts. If the Software Development Plan calls for a Physical Configuration Audit, one objective of that audit is to ascertain that the latest version of the SDD accurately describes the as-built product.

4.6.1.6 Other Alternatives and Issues

The specific design representation form and associated documentation should be closely aligned with the design process used by the developers. The level of detail in the Software Design Description should be necessary and sufficient for the developers to construct software source code. It should be a goal to make the design representation as simple and informative as possible, not as complex and detailed as possible. An SDD, or equivalent, is required by reference [EP401045] for War Reserve software projects.

4.6.2 Template Outline

An outline of a Software Design Description, based on reference [PPSD], is illustrated in Table 4-6. This outline is based on a tailoring of the design description described in reference [IEEE1016]. This outline is primarily suited to documenting the results of a structured design process. Since the design representation documentation is so closely aligned with the design process, there are other outline forms that may be more appropriate for other design processes.

Table 4-6. Software Design Description

- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions, Acronyms, and Abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Design Rationale
- 3. Design Description
  - 3.1 Module Decomposition
    - 3.1.1 Module 1 Description
      - 3.1.1.1 Purpose
      - 3.1.1.2 Function
      - 3.1.1.3 Subordinates
      - 3.1.1.4 Dependencies
      - 3.1.1.5 Resources
      - 3.1.1.6 External Interfaces
      - 3.1.1.7 Processing
      - 3.1.1.8 Internal Interfaces
      - 3.1.1.9 Data
    - 3.1.2 Module 2 Description
      - ...
    - 3.1.n Module n Description
      - ...
  - 3.2 Data Decomposition
    - 3.2.1 Data Entity 1 Description
      - 3.2.1.1 Purpose
      - 3.2.1.2 Subordinates
      - 3.2.1.3 Dependencies
      - 3.2.1.4 Resources
      - 3.2.1.5 Data Entity 1 Detail
    - 3.2.2 Data Entity 2 Description
      - ...
      - ...
    - 3.2.m Data Entity m Description
      - ...
      - ...
- 4. Unit Testing Strategy

Appendix 1: Glossary
Appendix 2: References
Appendix 3: Data Dictionary
Appendix 4: Matrices of Numbered Requirements vs. Design Entities

4.7 Software System Test Plan (SSTP)

4.7.1 Description

Formal software test documentation is necessary for all but extremely small, non-critical, and non-complex software programs. Such documentation is captured in a Software System Test Plan and normally addresses an approach for carrying out the testing, descriptions of the tests to be conducted, expected results of the tests, and actual results of the tests.

4.7.1.1 Abstract

The Software System Test Plan provides details of activities required to prepare for and conduct system tests, defines sources of the information used to prepare the plan, describes tests to be conducted and results expected from each test, defines the test tools and environments needed to conduct the tests, and includes the test results in an appendix.

4.7.1.2 Audience

The Software System Test Plan is intended primarily for the following five audiences:

- **Software Project Management** - provides a roadmap for testing the system; this roadmap enables program management to ensure that all system requirements are adequately tested;
- **Software Development Personnel** - provides insight into the tests to be conducted and the accept/reject criteria that will be applied for each test;
- **Software Test Personnel** - provides a specific plan, including test data, test environment, test procedures, and expected results and accept/reject criteria for each test; every requirement specified in the SRS is subjected to one or more tests or validations;
- **Software Mâintenance Personnel** - provides a starting point for testing changes and enhancements to the software during its operational life cycle; and
- **Software Quality Assurance Personnel** - provides a basis for validation that the programs satisfy each and every requirement in the SRS.

4.7.1.3 Development, Review, and Approval Responsibilities

Normally the SSTP is written by those who will perform the formal validation of the software. Depending upon the project, the testers may be a separate group, possibly even a different organization, or they may be the actual software developers. On small software projects, the software developers are often the software testers. The SSTP should be reviewed/inspected by software developers and testers. The project leader, test leader, and quality assurance personnel complete sign-off of the SSTP after final review/inspection. When sign-off is complete, the SSTP is placed under configuration management control in accordance with the Software Configuration Management Plan.

4.7.1.4 Content Guidelines

The content of the Software System Test Plan is derived from the SRS and tests are designed to verify that each requirement is satisfied. The requirements must be traceable, by identification/paragraph number in the SRS, to one or more program components, and to one or more test cases. This traceability should be captured first as a "Numbered List of Requirements" appendix in the SRS. The contents of that matrix should be traceable to the "Matrices of Numbered Requirements versus Design Entities" appendix in the SDD. The contents of those matrices should be traceable to one or more test identifier sections and to elements in the "Matrix of Numbered Requirements versus Test Identifiers" appendix in the SSTP.

The SSTP also needs to identify the testing strategy (for example, tools, techniques, metrics, and completion criteria) for each test as well as the test environment for hardware and software, test data, communications needs, security, and other factors that may affect a successful test. The SSTP should describe in detail the testing procedures, test data, and expected results and/or accept/reject criteria for each test case. The SSTP should also provide guidance on how to document (usually through a test log) and report the test results. It is possible that some software requirements are not testable in the given environment. These cases also need to be identified and explained in the SSTP.

4.7.1.5 Maintenance

The SSTP is maintained by software project management within the guidance provided by the Software Configuration Management Plan. All changes in software test activities should be incorporated into a configuration controlled SSTP.

4.7.1.6 Other Alternatives and Issues

In some situations, it may be useful to separate the plans, test case descriptions, and test results into three physically separate documents. For small projects, the Software System Test Plan information may be included in the System Test Plan. An SSTP, or equivalent, is required by reference [EP401045] for War Reserve software projects.

4.7.2 Template Outline

An outline of a Software System Test Plan, based on reference [PPSD], is illustrated in Table 4-7. This outline is based on a tailoring of the test documentation described in reference [IEEE829].

Table 4-7. Software System Test Plan

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions, Acronyms, and Abbreviations
    - 1.4 References
    - 1.5 Overview
2. Interactions and Dependencies
    - 2.1 Requirements
    - 2.2 Internal Interactions and Dependencies
    - 2.3 External Interactions and Dependencies
3. Requirements Not Tested
4. Testing Strategy
5. Test Environment
    - 5.1 Hardware
    - 5.2 Software
        - 5.2.1 Communications
        - 5.2.2 System
        - 5.2.3 Usage Mode
    - 5.3 Security
    - 5.4 Tools
    - 5.5 Documentation
    - 5.6 Environment Usage
6. Test Descriptions
    - 6.1 Test Identifier <1>
    - 6.2 Test Identifier <2>
    - 6.n Test Identifier <n>

Appendix 1: Glossary
Appendix 2: References
Appendix 3: Matrix of Numbered Requirements vs Test Identifiers
Appendix 4: Test Results

4.8 Software Support/Maintenance Plan (SSMP)

4.8.1 Description

The Software Support/Maintenance Plan provides a description of information useful to software maintainers once the software has been released and accepted. A significant aspect of this plan is to identify the process required to change and/or reproduce the software product.

4.8.1.1 Abstract

The Software Support/Maintenance Plan (SSMP) provides guidance on how to support specific software products that are covered by the plan. The SSMP documents software version identification information, build or regeneration procedures for compiling and linking the software from source libraries, regression testing methods, acceptance procedures, and maintenance procedures that might be useful to the support organization responsible for incorporating changes to the software in the future. The SSMP may also contain program-unique software delivery and installation and/or removal instructions if these are not addressed in other project documentation.

4.8.1.2 Audience

The Software Support/Maintenance Plan is intended primarily for the following four audiences:

- **Software Development Personnel** - provides information on the operational and support environments for which the software is designed to operate and be maintained;
- **Software Configuration Management Personnel** - provides information on generation of the software from source program files;
- **Software Maintenance Personnel** - provides information about changing software products during software maintenance activities; and
- **External Customers** - defines the support facilities and maintenance training that will be required to maintain the software.

4.8.1.3 Development, Review, and Approval Responsibilities

Software project management is responsible for allocating resources to develop and maintain the Software Support/Maintenance Plan. The software development team is responsible for developing the SSMP. If the supporting organization and the using organization are not the same, coordination between these two entities is essential to the development of an acceptable SSMP. Upon acceptance by the developing, supporting, and using organizations the SSMP is approved by the software project manager. During development, the SSMP is updated and maintained by the software development organization in accordance with the Software Configuration Management Plan. During support, the SSMP is updated and maintained by the software support organization using software configuration management processes for acceptance and approval of plan changes in accordance with the support organization's Software Configuration Management Plan (which may be the same plan as used by the development organization).

4.8.1.4 Content Guidelines

The Software Support/Maintenance Plan summarizes resource, support agreements, and environmental assumptions into a single document providing software maintenance engineers with an important source for maintenance information. Software maintenance includes life cycle phases similar to development but has a significantly larger emphasis on the change processing activities, such as problem reporting and change analysis, and regression testing to verify change implementation. The maintenance task originates with an approved request for service and terminates upon acceptance by the customer.

4.8.1.5 Maintenance

The SSMP is updated, reviewed, and approved as major changes are introduced to the plan. Changes are made in accordance with the software configuration procedures defined for development and support. Changes in resource allocation, product size, product type, and the operating environment should be reflected in the plan as soon as possible.

4.8.1.6 Other Alternatives and Issues

The SSMP information may be included with other software support documentation such as operator manuals or user's guides or, for small projects, may be included with the Software Development Plan.

4.8.2 Template Outline

An outline of a Software Support/Maintenance Plan, based on the Software Transition Plan in reference [IEEE1498], is illustrated in Table 4-8.

Table 4-8. Software Support/Maintenance Plan

1. Introduction
   - 1.1 Identification
   - 1.2 System Overview
   - 1.3 Document Overview
   - 1.4 Relationship to Other Plans
2. References
3. Software Support Resources
   - 3.1 Facilities
   - 3.2 Hardware
   - 3.3 Software
   - 3.4 Other Documentation
   - 3.5 Personnel
   - 3.6 Other Resources
   - 3.7 Interrelationship of Components
4. Recommended Procedures
   - 4.1 Software Modification
   - 4.2 Software Regression Testing
   - 4.3 Software Generation
   - 4.4 Software Quality Evaluation
   - 4.5 Corrective Action System
   - 4.6 Configuration Management
5. Training
6. Anticipated Areas of Change
7. Transition Planning
8. Notes

Appendices

4.9 Software Safety/Security Plan (SSP)

4.9.1 Description

The Software Safety/Security Plan is concerned with documenting methods and techniques that will be used to ensure that software does not fail in a way that contributes to the unsafe operation (e.g., results in personnel injury or loss of life) of the system or unauthorized access to protected resources (e.g., classified or proprietary material). Software requires special attention whenever it controls safety- or security-related functions or provides a backup function for safety or security features that are primarily implemented in hardware. An SSP can be used to provide a plan for ensuring that safety, security, or safety and security requirements are met.

4.9.1.1 Abstract

A Software Safety/Software Security Plan contains information relative to management of safety risks and security vulnerabilities during the design and implementation/maintenance of computer programs for critical systems. Technical efforts involve identification and analysis of potential problems, analysis of alternative solutions, implementation of the selected alternatives, and analysis of these implementations with respect to validity, completeness, and accuracy.

4.9.1.2 Audience

The Software Safety and Software Security Plans are intended primarily for the following eight audiences:

- **System Project Management** - provides this level of management with insight into the safety and security efforts that are to be carried out during the project;
- **Software Project Management** - provides plans by which safety- and security-related software efforts can be implemented and tracked by software managers;
- **System and Software Safety/Security Personnel** - provides plans by which safety- and security-related software efforts can be implemented and tracked;
- **Software Development Personnel** - software designers and programmers need an understanding of the safety and security activities to be carried out; safety and security requirements must be implemented by software designers and programmers;
- **Software Test Personnel** - provides test personnel with an understanding of the high level of reliability required in the execution and analysis of safety and security features; testing must be appropriately planned to complement planned safety- and security-related analyses activity;
- **Software Quality Assurance Personnel** - provides a basis for auditing compliance to the safety and security plan requirements;
- **Software Configuration Management Personnel** - provides an understanding of the criticality of configuration efforts related to safety- and security-critical software components; and
- **External Customers** - provides clarification concerning software safety and security standards that are to be enforced by the development organization.

4.9.1.3 Development, Review, and Approval Responsibilities

Not all software projects need a Software Safety/Security Plan. The need for such a plan should be determined by project management and the external customer. When such a plan is required, it is the responsibility of project management assisted by safety/security and software management personnel. The system project leader, customer representative, security/safety managers, and software quality assurance personnel should sign the SSP after a final review/inspection and recommended changes have been implemented. Once the sign-off is complete, the SSP should be placed under configuration management in accordance with the controls defined for such documents in the Software Configuration Management Plan.

4.9.1.4 Content Guidelines

The goal of the Software Safety/Security Plan is to reduce the risk of negative safety or negative security aspects being designed or programmed into critical software during development or maintenance activities. Normally, the SSP will be developed within the context of system safety and security programs. Safety or security becomes an issue when software is embedded/installed within a high consequence system. The content, level of detail, and resources required by a safety/security plan are determined by the type and level of risks associated with the software, the complexity of the software (and the system), and by external forces such as the threat environment and contractual or regulatory requirements. An SSP might typically address:

- management of activities such as documentation and records, configuration management, quality assurance, verification and validation, software reuse, and certification;
- analysis activities during software development and maintenance; and
- post-development activities such as user training, installation and startup, and maintenance.

4.9.1.5 Maintenance

The Software Safety/Software Security Plan requires updating if the management approach changes, new or additional technical activities are identified, or if documentation requirements change due to contractual or programmatic changes. The SSP may also require updating if changes are introduced to the system's threat environment, hardware, or the software.

4.9.1.6 Other Alternatives and Issues

A closely related concept to software security is Software Use Control. Classified software is protected according to the rules and regulations pertaining to its classification level. In addition, unclassified software may be subject to certain constraints such as export limitations or other release limitations based on its sensitivity or proprietary nature. Of special consideration is software, whether classified or not, that is used in nuclear weapons. This software, as well as the hardware that comprises the system, must be included in a Use Control Security Theme for the nuclear weapon being developed. The Use Control Security Theme is an independent document that may incorporate some aspects of a Software Safety/Security Plan.

4.9.2 Template Outline

An outline of a Software Safety/Security Plan, based on reference [IEEE1228], is illustrated in Table 4-9.

Table 4-9. Software Safety Plan/Software Security Plan

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Application
   - 1.4 Disclaimer
2. Definitions, Acronyms, and References
3. Software Safety/Security Management
   - 3.1 Organization and Responsibilities
   - 3.2 Resources
   - 3.3 Staff Qualifications and Training
   - 3.4 Software Life Cycle
   - 3.5 Documentation Requirements
   - 3.6 Software Safety/Security Program Records
   - 3.7 Software Configuration Management Activities
   - 3.8 Software Quality Assurance Activities
   - 3.9 Software Verification and Validation Activities
   - 3.10 Tool Support and Approval
   - 3.11 Previously Developed or Purchased Software
   - 3.12 Subcontract Management
   - 3.13 Process Certification
4. Software Safety/Security Analysis
   - 4.1 Software Safety/Security Analyses Preparation
   - 4.2 Software Safety/Security Requirements Analysis
   - 4.3 Software Safety/Security Design Analysis
   - 4.4 Software Safety/Security Code Analysis
   - 4.5 Software Safety/Security Test Analysis
   - 4.6 Software Safety/Security Change Analysis
5. Post-Development
   - 5.1 Training
   - 5.2 Deployment
     - 5.2.1 Installation
     - 5.2.2 Startup and Transition
     - 5.2.3 Operations Support
   - 5.3 Monitoring
   - 5.4 Maintenance
   - 5.5 Retirement and Notification
6. Plan Approval

Appendices

4.10 Software Users Guide (SUG)

4.10.1 Description

The Software Users Guide describes how users of application software can produce desired results using the software. The SUG provides the syntax, commands, key strokes, and pointing device operations necessary to execute all options the software provides. The SUG should address the range of operator/user knowledge and experience backgrounds appropriate for the software's operational use.

4.10.1.1 Abstract

The format and content of a Software Users Guide depends upon the software application and the operational use (modes) of the software. An SUG includes all information necessary for the user to operate the software: data inputs, operational procedures, expected outputs, and error processing instructions. Operational procedures may include installation, initialization, backup and recovery, as well as procedures to be conducted to cause the software to perform its intended operational functions. Some of the information for an SUG might be provided as an on-line capability and simply referenced by the SUG. User interfaces such as display screen and keyboard interactions may be prototyped early in the software project to facilitate customer acceptance of the interfaces and support the requirements gathering process. The prototype may then evolve into information that is documented in an SUG.

4.10.1.2 Audience

When prepared early in the software development cycle, the Software Users Guide (or a prototype of the user interface part of the SUG) may be used by the software developers to ensure the system works the way it was advertised to the customer. The preparation of the SUG also provides the customer an opportunity to understand and have input into how the system is intended to operate before the software is written. The SUG is intended primarily for the following three audiences:

- **Software Development Personnel** - provides insight into the design of user interfaces;
- **Software Maintenance Personnel** - provides insight into the design of user interfaces, error recovery and backup maintenance procedures, and the functional capabilities of the software; and
- **External Customer/Users** - provides a source of information for initially learning how to use the software and a ready reference for looking up features and operational information.

4.10.1.3 Development, Review, and Approval Responsibilities

The software development team is responsible, possibly with the help of outside technical writing expertise, for producing the Software Users Guide. The customer is responsible for reviewing early views of the user interface functions and formats as part of the software requirements gathering process. The project leader, customer representative(s), and software quality personnel complete sign-off of the SUG after final review/inspection. When sign-off is complete, the SUG is placed under configuration management control in accordance with the Software Configuration Management Plan.

4.10.1.4 Content Guidelines

The Software User Guide provides a background of the system operation and a step-by-step walk through of the system features and user operational procedures. For menu-driven systems, the document should contain a menu-tree diagram. For graphic-user-interfaces (GUIs) it should show diagrams of the screen at two levels. The first level should be a description of the basic components of the screen, such as tool and menu bars. The second level shows screen configurations having data displayed on the screen to represent the type of functions described in the associated text. The SUG must be able to address issues from the novice to the expert user of the system. The ability to quickly find answers to questions requires both an in-depth table of contents and a key word index. The SUG may reference on-line user documentation for more specific details.

4.10.1.5 Maintenance

The Software Users Guide should be prepared electronically using a word processing system, hypertext system, or on-line documenting tool. The maintenance of the SUG can then be upgraded as each phase in the life cycle is revisited and changes are made. Changes to the SUG should be controlled through development and support configuration management procedures.

4.10.1.6 Other Alternatives and Issues

The format and content of the Software Users Guide should be prepared to be usable and maintainable. The SUG may be a combination of written document information, on-line information available to the user during operational use, and electronic interface screens that graphically define the format, inputs, and outputs that the user must understand.

4.10.2 Template Outline

An outline of a Software Users Guide, based on the Software Users Manual in reference [IEEE1498], is illustrated in Table 4-10.

Table 4-10. Software Users Guide

1. Introduction
    - 1.1 Identification
    - 1.2 System Overview
    - 1.3 Document Overview
    - 1.4 Relationship to Other Plans
2. References
3. Software Summary
    - 3.1 Software Application
    - 3.2 Software Inventory
    - 3.3 Software Environment
    - 3.4 Software Organization and Overview of Operation
    - 3.5 Contingencies, Alternate States, Modes of Operation
    - 3.6 Security and Privacy Protection
    - 3.7 Assistance and Problem Reporting
4. Access To The Software
    - 4.1 First-Time User Of The Software
        - 4.1.1 Equipment Familiarization
        - 4.1.2 Access Control
        - 4.1.3 Installation and Setup
    - 4.2 Initiating a Session
    - 4.3 Stopping and Suspending Work
5. Processing Reference Guide
    - 5.1 Capabilities
    - 5.2 Conventions
    - 5.3 Processing Procedures
        - 5.3.n (Aspect of Software Use)
    - 5.4 Related Processing
    - 5.5 Data Backup
    - 5.6 Recovery From Errors, Malfunctions, and Emergencies
    - 5.7 Messages
    - 5.8 Quick-Reference Guide
6. Notes

Appendices

4.11 Software Verification and Validation Plan (SVVP)

4.11.1 Description

The Software Verification and Validation Plan describes the approach for carrying out the software tasks to verify that the products from successive life cycle phases and activities are correct and validate that the final software product satisfies the original operational intent as captured in system and software requirements. Verification is the evaluation of a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. For example, one verification activity might determine whether the content of a Software Design Description fully and accurately addresses each of the requirements contained in the Software Requirements Specification. Validation is the evaluation of a system or component during, or at the end of, a development process to determine whether the documented requirements are satisfied. In short, verification answers the question of whether the software is being developed correctly and validation answers the question of whether the correct software has been developed.

4.11.1.1 Abstract

Verification and validation efforts are carried out by a team that is often independent of the software development and software test teams. The Software Verification and Validation Plan is principally a set of task descriptions that address verifying the results of each stage of software development and validating the intermediate and end products against the requirements. Typically these tasks are initiated during concept development of the system.

4.11.1.2 Audience

The Software Verification and Validation Plan is intended primarily for the following four audiences:

- **Software Project Management** - project management will have a plan which defines the software verification and validation processes, resources, and tools to be used for the project being implemented; the plan also serves as a measure of progress during the software development effort; there is additional assurance, assuming the plan is good and well executed, that the desired product will meet customer requirements;
- **Software Development Personnel** - all members of the software development team (software designers, programmers, analysts, testers, and maintainers) benefit from the overall emphasis on verifying and validating that software requirements are being met and that any defects are being eliminated earlier in the development process; the plan may be used by these individuals to understand how the software might be evaluated.
- **Software Verification and Validation Personnel** - provides an overall plan for the verification and validation efforts, and describes the tasks that are to be performed at each stage of the software life cycle process; verification and validation personnel include all reviewers, inspectors, and testers in the project.
- **External Customers** - the SVVP provides customers a more refined estimate of how program goals will be met and whether the program risks have been properly identified; the SVVP will also provide customers with insight concerning the quality being built into the software product.

4.11.1.3 Development, Review, and Approval Responsibilities

The Software Verification and Validation Plan is the responsibility of the software manager, project leader, and the verification and validation team. It is normally developed by the verification and validation team in cooperation with the software manager and quality assurance personnel. The SVVP is normally signed by the software project manager, unless the verification and validation activities are performed as an independent activity. In this case the document may be signed by both the project manager and the manager of the independent activity. Once the SVVP is reviewed/inspected and the sign-off process completed, the SVVP should be placed under project configuration management control.

4.11.1.4 Content Guidelines

A Software Verification and Validation Plan is usually required for complex and critical software development projects. This latter criteria may be safety or security related but could also be driven by risk of financial loss or loss of reputation. An SVVP should be developed to complement other quality related plans and activities such as safety, security, configuration management, and quality assurance. Each task defined in the SVVP should contain information concerning resources needed to complete the task, risks and assumptions relating to the task, and roles and responsibilities for completing the task.

The SVVP is written to fit the life cycle model being used to develop the software. The SVVP outline provided uses the traditional waterfall model as a basis for discussion. The SVVP tasks each have entrance and exit criteria that are predominately product engineering information in the form of requirements specification, design descriptions, test designs and test plans. Therefore, the emphasis should be to model the plan based on the predominant transitions from one major activity to another. For example, one transition might be from design to implementation. At this transition, design information should be reasonably complete to satisfy the entrance criteria for tasks associated with the verification of the design activity. The completion of the tasks may sequentially address the design information of software components in the order the components are completed.

4.11.1.5 Maintenance

The Software Verification and Validation Plan is normally initiated early in the development process but must be able to be modified as development proceeds. Many factors change between concept development and requirements analysis. The SVVP should be updated to be consistent with these changes. Like the software requirements, however, the development approach, required tasks, and management issues should stabilize and the SVVP should need less updating as development moves into design.

4.11.1.6 Other Alternatives and Issues

When a Software Verification and Validation Plan is required, it normally becomes a driving factor in the development process. An SVVP is usually needed when, either by customer demand or because of the critical nature of the software (and system), a warrant is required that all possible attention has been given to the correct development of the software. It should also be recognized that a formal verification and validation program does not come without considerable additional cost. A formal verification and validation program requires a substantial documentation effort to successfully complete the tasks and provide for reporting requirements.

It is useful to distinguish between an SVVP and an SQAP. Both plans have the intent of ensuring software is developed with attention to quality. Perhaps the primary difference is in the scope of activity and responsibilities. The SVVP is developed and implemented primarily by an independent organization, perhaps one reporting to the customer. The SVVP typically covers a broader range of life cycle activities and at greater depth than the SQAP. The SQAP is developed and implemented by a combination of personnel who have quality and project-specific responsibilities. The quality personnel may be completely independent or totally integrated with the project. The scope of activities is integrated with the software development process.

Within the nuclear weapon development community, there exists a requirement to perform an engineering qualification evaluation on the weapon system components, including the software. This requirement is accomplished by the development and issuance of a Qualification Plan (QP) that specifies tasks and analyses to be performed in the evaluation. The QP is reissued regularly with revised information reflecting the progress of the evaluation. Upon completion of the evaluation, a Qualification Evaluation Release (QER) is issued stating whether, and to what extent, the component can be used. The QP and QER accomplish the spirit of what the SVVP is intended to do. When a QER has been issued on a software component of a nuclear weapon stating that it is fit for use in that system, then that software component has successfully completed a verification of the development processes and has been validated as meeting the requirements placed upon it.

4.11.2 Template Outline

An outline of a Software Verification and Validation Plan, based on reference [IEEE1012], is illustrated in Table 4-11.

Table 4-11. Software Verification and Validation Plan

1. Introduction

    1.1 Identification

    1.2 System Overview

    1.3 Document Overview

    1.4 Relationship to Other Plans

2. References

3. Definitions

4. Verification and Validation Overview

    4.1 Organization

    4.2 Master Schedule

    4.3 Resources Summary

    4.4 Responsibilities

    4.5 Tools, Techniques, and Methodologies

5. Life-Cycle Verification and Validation

    5.1 Management

    5.2 Concept Phase

    5.3 Requirements Phase

    5.4 Design Phase

    5.5 Implementation Phase

    5.6 Test Phase

    5.7 Installation and Checkout Phase

    5.8 Operation and Maintenance Phase

6. Software Verification and Validation Reporting

7 Verification and Validation Administrative Procedures

    7.1 Anomaly Reporting and Resolution

    7.2 Task Iteration Policy

    7.3 Deviation Policy

    7.4 Control Procedures

    7.5 Standards, Practices, and Conventions

Appendices

5 Software Implementation Documentation

This chapter describes software source code standards and unit software documentation in more detail, and provides information on using software development folders to organize and capture software implementation documentation. A typical small, non-critical software project may have no implementation documentation other than source code. For software development efforts that involve several programmers or more critical applications, the necessity for additional and more formally organized implementation documentation increases. Implementation documentation includes unit software source code, unit software verification test documentation, and implementation standards and guidelines such as coding standards or styles. The unit software source code may include application source code, control language files, data files, and test software developed specifically to accomplish the unit software verification tests.

5.1 Source Code Implementation Standards

Standards establish a common ground that enhances communication. Standards exist for many programming languages. The syntax of a programming language is established so that a compiler or interpreter can perform the tasks described by a software developer. This basic syntax is sufficient when it is a computer program trying to read the task description. However, it is often desirable for a human being to be able to read this description as well. Requirements and conventions placed on top of the basic syntax of a programming language, in order to improve the readability for humans, are commonly referred to as coding standards or style guidelines.

Source code needs to be readable in order to enhance supportability and reuse. In order for source code to be easily supported or reused, it must be well documented and consistently written. By following coding standards, developers are more likely to correctly implement the intended task or requirement in the software. Even if the source code is not initially correct, by following coding standards it will be easier to locate and eliminate software defects prior to delivery. See reference [K&P] for more on elements of programming style and coding standards.

5.1.1 Source File Documentation

Coding standards often specify that certain information must be included in a header at the beginning of every source code file. This information serves as a table of contents for the file. A person reading the source file can immediately see the most pertinent information about the contents of the file. File headers typically include the following information:

- file name;
- short statement of.purpose;
- list of functions, global variables, and data types defined;
- list of dependencies for the file; and
- change history.

In addition to file headers, each function in the file should have a header with certain information which is pertinent to that function. Some ideas for what to include are:

- function name;
- short description;
- names and types of parameters;
- return value;
- global variable usage;
- other functions used;
- file dependencies;
- change history; and
- special considerations.

For uniformity, the coding standard should specify templates for these headers so that the software developers can copy the format into their favorite source code editor.

5.1.2 Naming Conventions

Files, functions, variables, constants, and data types all need names. The more care with which these names are assigned, the more readable the source code will be. In addition to a general rule requiring that names be meaningful, the use of lower and upper case letters is often specified to distinguish between variables, constants, and data types. For example, the coding standard may specify that variable names be in lower case, constants in upper case, and data type names use title case.

Another useful standard is to require that all names visible outside of the file in which they are defined have a part of the name that links to the file name. A prefix can be assigned to each file and then used as a part of the function, global variable, constant, and data type names. The use of a prefix as part of global names allows a person reading the code to more quickly identify where a particular name is defined.

5.1.3 Language Constructs

Most programming languages provide a rich set of constructs for specifying programs. Unfortunately, just because the syntax of a language allows the programmer to write something a certain way doesn't mean it should be written that way. Some constructs of the language may be easier to understand or less error prone than others. For example, in languages that contain a "goto" construct, it is common for a coding standard to discourage its use when another language construct is sufficient. Similarly, a "switch" or "case" construct is usually considered better than a long string of nested "if-then-elses". By necessity, rules on the use of language constructs will be very language specific. Consult style guides for the individual languages to get ideas for what are considered good standards.

5.1.4 Style and Layout Guidelines

A programmer may use standard file and function headers, assign meaningful names, and avoid confusing language constructs but the code could still be very difficult to understand. Just imagine a function that is several hundred lines long, or worse, a function that is just as long but consists of a single line. Some subjects which could be addressed in this section of a coding standard are:

- use of white space;
- use of comment lines;
- function sizes;
- use of global data versus parameter passing; and
- indentation schemes.

5.1.5 Maintenance of Coding Standards

A coding standard, like any standard, should not be considered carved in stone. It should evolve as experience is gained with the language and programming in general. A coding standard is a repository of the pearls of wisdom that are acquired over time and a place to record these pearls so that they are not forgotten. The organization responsible for development of software using particular source languages should develop (or reference) coding standards for each language. If the coding standards are developed, then changes to the coding standard should be controlled in a formal manner in accordance with organization-level configuration management procedures.

5.2 Unit Development Documentation

This section describes a detailed approach to the documentation activities associated with unit software development and testing. Usually, a unit is the lowest level of separately compilable software structure. It may be a subroutine, module, function, procedure, or other such nomenclature depending on the computer language in which the software is written. The following paragraphs describe the elements of unit documentation, and discuss the types of unit documentation that are most useful to other software developers and software testers. This documentation should be captured in the unit's Software Development Folder (see section 5.3).

5.2.1 Developing Unit Documentation of Known Quality

If the unit developer's job were just to produce source code for delivery, it would require only the following work:

- develop an understanding of what the execution of the unit is to have accomplished when it returns from an invocation of each specified entry point;
- design internal data structures and algorithms (sequential procedures, knowledge frames, or rule bases) to implement the functions for the specified entry points; and
- encode these understandings in a specified compilation or scripting language.

Of course, most developers also do some testing, so when they pass their units to their customers, they feel it will work. In fact, most unit developers have other developers and the system testers as their customers. Consequently, the criterion for a good job in unit development documentation is that those other developers and testers can readily use the units produced by a developer. Documented units and the delivery of unit documentation to other developers and system testers is among the cheapest and the most valuable parts of the unit development job. The bullet list below is a suggested task list for a unit developer integrated with the suggested unit documentation, including test documentation. This list provides some perspective for how unit documentation is used within the unit development and test process.

- develop and record the catalog of unit actions;
- develop and record the unit software design and testing objectives;
- record a test design for test software and test cases that will achieve the testing objectives;
- record a code design for the unit software that will implement the catalog of unit actions;
- prepare test cases and test software for testing the unit;
- encode the unit software;
- build a test that is executable from the test software and the unit software and then execute the associated test cases; and
- evaluate the test results for correctness; evaluate the test execution trace for coverage; locate and correct defects in the unit software and test software.

Corresponding to the two kinds of unit customers, developers and system testers, the associated unit documentation is summarized in the following two subsections.

5.2.2 Unit Documentation for Developers

Unit development should produce a catalog of unit actions, code design, and unit software that will be needed at a later time by any other developer (including the unit's developer) for understanding and reusing or modifying the unit.

Catalog of Unit Actions

The catalog of unit actions records the understanding of what the execution of the unit is to accomplish when it has returned from an invocation of each specified entry point of the unit. Usually, each entry point has several distinct actions associated with it. There is, of course, the intended action performed by the unit when the entry point is called at the right time with valid inputs and sufficient resources. In addition, there are often numerous other actions to be accounted for; for example, the action performed in response to calling the entry point:

- at the wrong time (the unit is in the wrong state);
- with invalid input data values;
- with input data which does not satisfy particular relationships; and

- when there are insufficient resources for its job.

A good programmer usually accounts for these situations, but in many programming teams that programmer is (unfortunately) the only person who knows just what actions are performed. For each entry point, the catalog of unit actions should list all the distinct actions, giving for each:

- a unique label which enables reference to the action;
- a list of unit states in which it is allowed to occur ;
- a list of the inputs expected (names, types, interpretations) and their valid domains;
- a list of input relationships expected and the conditions which make them valid;
- a list of needed resources and their sufficiency (often seen as a success or fail response from a resource management utility, e.g., a dynamic memory allocator or a communication link); and
- a definition of unit response in terms of the outputs produced (names, types, interpretations) and the resulting unit state, if different than the initial state in which the action starts.

The catalog of unit actions is evidently not redundant with a software system's design documentation. In most cases, system designs are not refined this well before unit development is assigned.

Unit Software Design

A useful unit software code design gives other developers hints about internal data structures, algorithms (sequential procedures, knowledge frames, or rule bases) which are used to implement the functions for each specified entry point, and packaging (identifying internal functions/procedures in the unit and references to external units and libraries).

For data structures, document the computational or control role it performs, when and why it is set, and what its values represent. For algorithms, document any recursion (especially termination and error recovery), any numerical errors of truncation or approximation, and control of error propagation. For packaging, document subordinate units, internal subordinates having no external visibility, and allocation of responsibilities among the unit and its subordinates.

Designing unit software without thinking in advance about how the unit will be tested may lead to code that is difficult to test. Generally, don't design what can't be tested and don't code what can't be tested. More specific guidelines for design testability include the following:

- package data and code structures into logical units;
  ⇒ encapsulate data structures to control scope and visibility
  ⇒ localize and standardize data accesses, data validation, and constraint enforcement
  ⇒ add entry points with get and set functions for data and action states
  ⇒ alias data and code structures with reference names
  ⇒ localize and protect critical sections
  ⇒ maximize cohesion, minimize coupling
- enforce required function call sequences by constructing a finite state machine implementation of the valid sequences;
- implement error detection, notification, response, and recovery behavior;
  ⇒ validate inputs
  ⇒ validate outputs
  ⇒ arrange for continued execution after failures during testing
- include functions to reset states and display internal data structures;
- parameterize capability and resource limits;
- allow a disconnect from external hardware if possible (e.g., by redirecting memory-mapped I/O from external device to local RAM);
- use design inspections; and
- use classical software architectures.

layers	pipes	remote procedure calls
hierarchies	transactions	abstract data types
callbacks	blackboards	object messaging

A useful design validation technique is to provide a high level description of the mechanism which performs each action in the catalog of unit actions. Such a description indicates the sequence of calculations and transformations occurring in the internal data structures when the entry point is invoked under the specified conditions. It includes resource usage (time, memory, communications) and sequence of references to both internal and external subordinate units.

Unit Software Code

The encoding of the mechanisms from the code design into a specified compilation or scripting language produces unit software files. Often these files are supplemented with other files of data and function declarations, unit configuration parameters, and compilation instructions (notes or configuration and build files for automatic compilation). Variations and extensions of those files may be further defined through a variety of standard editing facilities in the compilers or interpreters which process those files (e.g., conditional compilation, pragmas and code generation modifiers). The preceding section 5.1 on Coding Standards addresses the documentation of the unit software code files.

5.2.3 Unit Documentation for Testers

Unit development should produce a variety of documentation for any tester including a developer who is modifying or reusing the unit and needs regression tests. This documentation should address what behaviors and properties of the unit software were tested at the unit level and how they were tested. This documentation should include testing objectives, test design, test software, test cases, and test results. These documentation elements are summarized in the paragraphs below.

Testing Objectives

The testing objectives list of all the operational things about the unit to be shown by unit testing. This list consists of selected information from the catalog of unit actions and information derived from code design. At the unit level, the testing objectives should include every action listed in the catalog of unit actions. In addition, the following kinds of things should also be shown:

- the correct actions are performed at each of the boundaries of each input data domain;
- the correct action is performed for input data meeting preconditions and failing to meet preconditions; preconditions include validity or invalidity of individual input data and satisfaction or violation of input data interrelationships;
- the correct action is performed for each possible combination of independent preconditions in a decision table or decision tree for action selection;
- the correct action is performed for each possible failure in the unit's environment (e.g., device time-out, insufficient resource, or missing or corrupt file) and any special combinations of failures demanding unique response;
- actions are performed in all sequences which cause all allowed transitions between all allowed unit states; all actions are executed in each unit state allowing them;
- every code segment, every conditional branch, every multi-way switch are executed, and (when appropriate) every loop is executed zero, one, and two times; and
- paths forcing all variable def-use pairs are executed; a def-use pair is a statement assigning a value to a variable followed by statements later in the path which use the value from that assignment statement.

All test objectives except the last two are called functional (or behavioral) testing objectives. Accomplishing all functional testing objectives provides assurance that the unit does what it should when it should. The code flow and data flow objectives are called structural testing objectives. Accomplishing the structural testing objectives provides assurance that there isn't untested or unexpected functionality in the unit.

Other aspects of the unit may provide testing objectives. For example, developers may understand likely errors which other developers may make in using a unit as a subordinate capability to their units. In that case, the code design and testing objectives may include either structural or execution checks for those kinds of unit misuse by another developer. Simple examples of misuse include a caller providing an invalid address at which the unit is to place output and a caller failing to call a unit initialization function before using one of its functions.

Test Design

Distinguished from the testing objectives which contain a selection and specification of things to be shown by unit testing, is the test design which indicates what kind of test cases will be used to show that the unit software correctly implements each of the objectives. Every functional testing objective is explicitly tied to one of the unit's actions. Therefore, designing test cases for a particular functional testing objective results in test cases which are explicitly traceable to the unit's requirements in the catalog of unit actions.

The unit testing approach in test design is documented as a record of five classes of decisions:

- what samples and combinations of input data will be used in test cases;
- what initial situations (unit states, resources) will be used in test cases;
- in what sequences test will be administered;
- how test cases will be documented; and
- how tests will be executed.

Test Software

The term test software is just a collective name for driver and stub code. A driver is code that sets up unit input data in executable form and invokes entry points to execute the test. From the unit's perspective, a driver is a surrogate for all other units that may invoke entry points in the unit under test. A unit test driver can be largely standardized so the individual developer has very small amount of extra work in adapting it to any particular unit. A stub is simple code that has the same invocation interface as a real subordinate for receiving and returning calls or messages from the unit. Of course, real subordinates may also be used as long as they have already been tested and shown to execute properly. The documentation should clearly describe the test execution environment architecture of driver and stubs for the unit under test as illustrated in Figure 5-1.

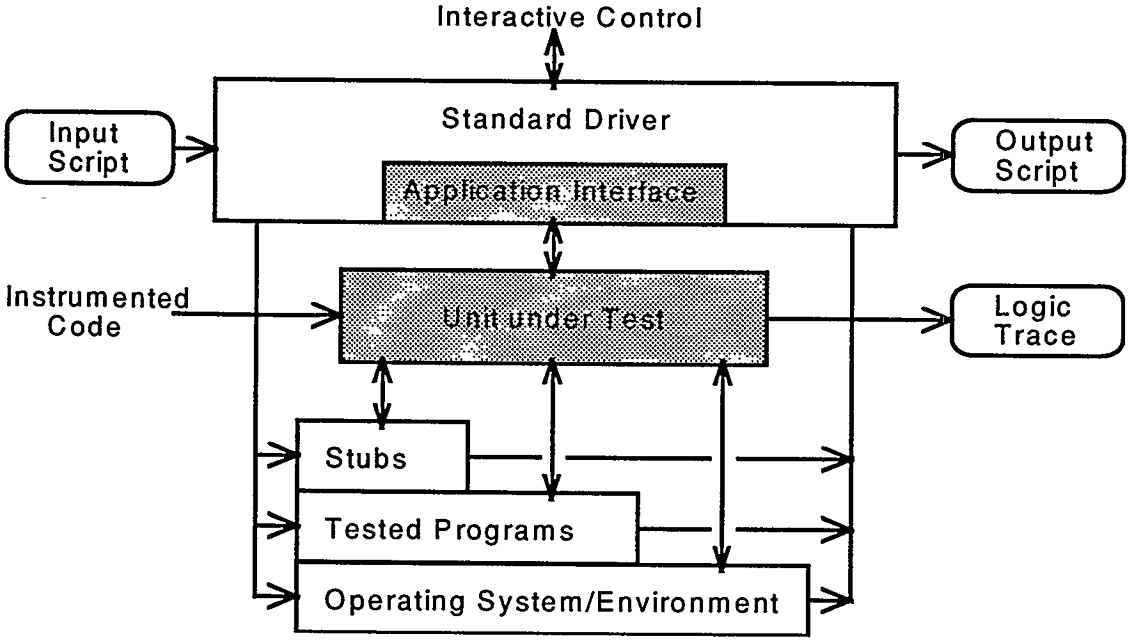


Figure 5-1. Framework for Test Software

Test Cases

Test cases are identified when test design decisions have been applied to one or more testing objectives for a particular action of a particular entry point. Each test case has particular values specified for each input variable and an initial situation specified in terms of the unit's state (if any) and resource supply (if needed). A test case is completed when the catalog of unit actions has been used to determine the expected outcome of the test execution, giving expected values for each output variable and a specification of the final situation in terms of unit state and resources.

When the actual test cases are devised according to the test design, they should be recorded in some form. Ideally, this form would be some kind of executable script file which could be used directly in test execution. A combination of comments and data in an executable script form is suggested. Whether the test case documentation is recorded for human or machine administration, it should provide the following information:

- a unique test case identifier;
- a reference to the particular action and entry point being executed;
- any required setup of the unit into a particular state (an appropriate sequencing of test cases is usually used to achieve this);
- the purpose of the test's input sample combinations in terms of subdomain body or boundary being probed, structure size or structure alternative being probed, array or table characteristic being probed, or data relationship being violated;
- a list of names and values of the input data;
- the particular precondition being satisfied or violated;
- the method of causing unit execution of the test;
- the method of determining end of unit execution;
- a list of names and expected values of the output data;
- the particular postcondition expected to be satisfied; and
- any state transition expected to a different unit state;

Test Results

A test occurs when a test case, administered through the test software, is executed by the unit software, resulting in an outcome. The test results are the actual output data values and, when appropriate, the resulting unit state and resource situation. Since a test case may be executed multiple times, there can be multiple test results for a particular test case. Labeling test results would require:

- the unique identifier for the test case which was executed;
- an identification of the unit software version which executed the test;
- an identification of the test software version which administered the test; and
- a time stamp for the particular execution of the test case.

The test results may or may not be the same as the expected results documented in the test case. If they are the same, to within some tolerance, the test case is marked passed. Otherwise it is marked failed. It is common at the unit development level not to regard the development task as complete until all the test cases have passed. Therefore, such detailed labeling and marking of test cases is unnecessary, except for knowing which test case produced which results. The test results are simply saved in their observable form. The unit software, the test software, and the test results are all consistent and can be labeled with the unit's configuration component identifier.

5.3 Software Development Folders

This section describes the format and content of a Software Development Folders (SDF), also called a Software Development File, Software Development Notebook, or Unit Development Folder. A Software Development Folder is a collection of material pertinent to the development of a given software unit or set of related units. Contents typically include the requirements, design, technical reports, code listing, test plans, test results, problem reports, schedules, and notes for the units [IEEE610].

The Software Development Folders are started in skeletal form at the start of each program unit and become an important organizational tool for implementation documentation during software development. The SDFs provide management with a mechanism for monitoring individual software unit implementation progress. The SDFs provide software maintainers with invaluable insight into the design and implementation of software units when changes to the units are required. Content requirements for project SDFs should be specified in the Software Development Plan and controlled in accordance with the Software Configuration Management Plan. Some times SDFs are only used for internal project unit management and are not delivered to the customer. Other times SDFs are more formally controlled as a customer deliverable.

The SDFs have been historically composed of code documentation on hard copy and executable code on machine-readable media but the trend today encourages increased use of automated tools to support the creation and maintenance of SDFs. The SDFs should be managed in accordance with the scope of the project, the tools available, and the budget. General content of the SDF is illustrated in Figure 5-2.

Purpose: Record software development activites associated with a program unit.
Provide basis for final as-built documentation

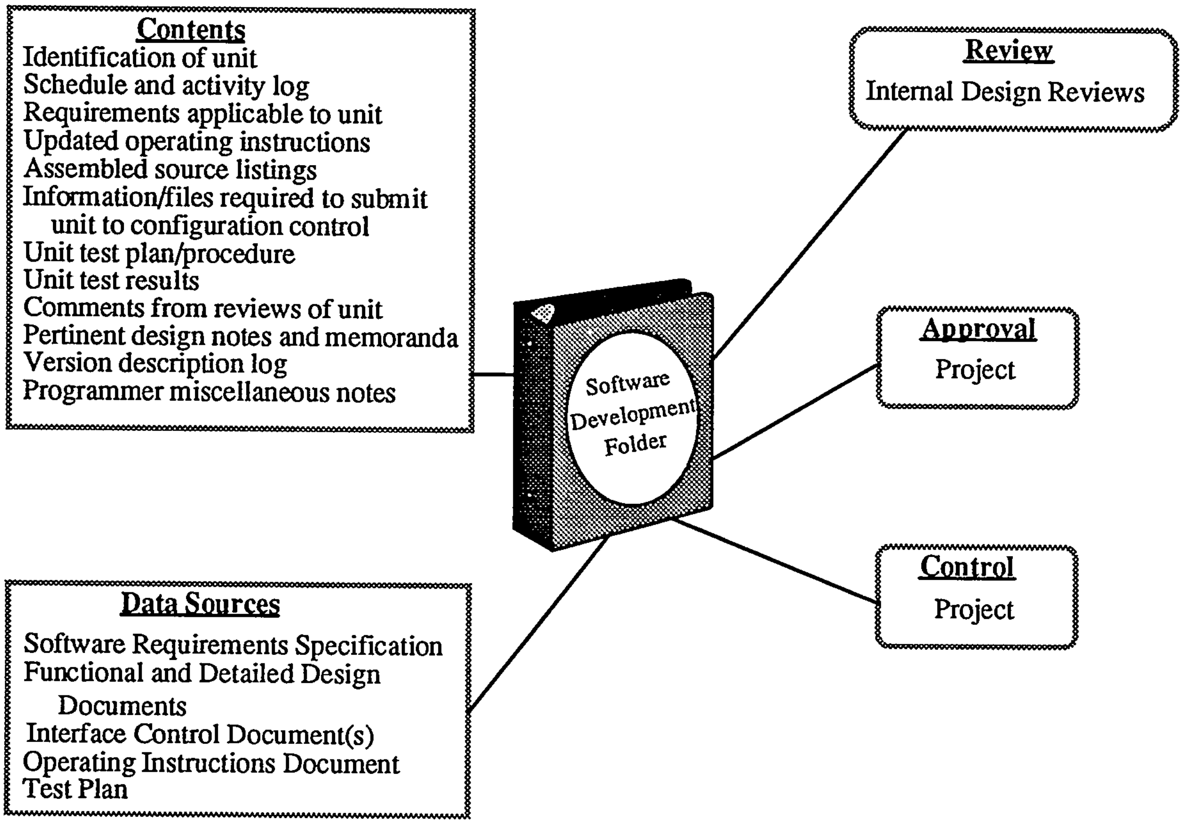

Contents
Identification of unit
Schedule and activity log
Requirements applicable to unit
Updated operating instructions
Assembled source listings
Information/files required to submit unit to configuration control
Unit test plan/procedure
Unit test results
Comments from reviews of unit
Pertinent design notes and memoranda
Version description log
Programmer miscellaneous notes
Software Development Folder
Review
Internal Design Reviews
Approval
Project
Control
Project
Data Sources
Software Requirements Specification
Functional and Detailed Design Documents
Interface Control Document(s)
Operating Instructions Document
Test Plan

Figure 5-2. Software Development Folder Content

The number and kind of SDFs will vary from project to project according to variations in the access rights and needs of the developers, which are directly related to the levels of control as outlined in the SCMP. The insertion of entities and changes to entities in a controlled SDF should produce an auditable authorization trail. The names of SDFs may vary, but fundamentally three kinds should be considered.

5.3.1 Programmer's SDF

This folder is used for holding newly created or modified software entities (units/modules or data files and associated documentation). This SDF is used by programmers in developing code. It is freely accessible to the programmers responsible for that unit at any time. It is the programmers' workspace and controlled by the programmers.

5.3.2 Controlled SDF

The Software Development Folder is used for managing the current baseline(s) and for controlling changes made to them. This folder documents units and components of a configuration item that have been promoted for integration. Entry is controlled, usually after verification. Copies may be freely made for use by programmers and others. Changes to units or components in this folder must be authorized by the responsible authority, which could be a configuration control board or other body with delegated authority.

5.3.3 Archived SDF

Sometimes called the software repository, this folder is used as an archive of various baselines released for general use. This folder contains the master copies plus authorized copies of computer program configuration items that have been released for operational use. Copies of these masters may be made available to requesting organizations.

Software libraries and SDFs are used as tools to manage the configuration evolution of the software development baselines such that changes are made in a systematic, controlled way and that the configuration is not inadvertently corrupted. This ensures that the status of the software is well-understood at all times and that changes have not been made without having gone through a process of review, approval, and authorization for implementation, producing an audit trail. Libraries and SDFs can also serve as a means to control access to software components, with standard procedures to be followed and checks made via passwords or authorization lists to be able to successfully modify them. Thus, a certain amount of safety and protection (against loss or deletion, accidental or intentional) is afforded in a convenient way to software components through effective use of SDFs and a library structure.

Blank Page

Appendix A: References and Bibliography

[ANS10.4] "Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry," American Nuclear Society, 1987.

[ARNOLD] Arnold, R., Software Reengineering, IEEE Computer Society, March 1993.

[BROCKMAN] Brockman, J., Writing Better Computer User Documentation, From Paper to Hypertext, John Wiley & Sons, Inc, 1990.

[DOD2167A] DOD-STD-2167A, "Defense System Software Development," Department of Defense, February 29, 1988.

[DOD2168] DOD-STD-2168, "Defense System Software Quality Program," Department of Defense, April 29, 1988. Documentation Data Item Description is:
DI-QCIC-80572 Software Quality Program Evaluation Plan

[DOE1330.1D] "Computer Software Management," DOE Order 1330.1D, May 18, 1992.

[DOE1360.1A] "Acquisition and Management of Computing Resources," DOE Order 1360.1A, May 30, 1986.

[DOE1360.2A] "Unclassified Computer Security Program," DOE Order 1360.2A, May 20, 1988.

[DOE1360.3A] "Automatic Data Processing Standards," DOE Order 1360.3A, July 11, 1983.

[DOE1360.4B] "Scientific and Technical Computer Software," DOE Order 1360.4B, December 31, 1991.

[DOE5637.1] "Classified Computer Security Program," DOE Order 5637.1, January 29, 1988.

[DOE5700.6C] "Quality Assurance," DOE Order 5700.6C, August 21, 1991.

[DPMM] Cook, C., and Visconti, M., "Documentation is Important," Software Technology Support Center, CrossTalk, November 1994.

[EP401040] "Drawing System," Interagency Engineering Procedure, January 15, 1992.

[EP401045] "Definition of Computer Software Configuration Items," Interagency Engineering Procedure, December 1, 1992.

[FAGAN] Fagan, M., "Advances in Software Inspections," IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, July 1986, pp 744-751.

[HUMPH90] Humphrey, W., Managing the Software Process, Addison-Wesley, Reading MA, 1990.

[IEEEstds] IEEE Software Engineering Standards Collection: 1994 Edition, Wiley-Interscience, New York, NY, 1994.

[IEEE610] IEEE Std 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology (ANSI)," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE730] IEEE Std 730.1-1989, "IEEE Standard for Software Quality Assurance Plans," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE828] IEEE Std 828-1990, "IEEE Standard for Software Configuration Management," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE829] IEEE Std 829-1983, "IEEE Standard for Software Test Documentation," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE830] IEEE Std 830-1993, "IEEE Guide for Software Requirements Specifications," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE990] IEEE Std 990-1987, "IEEE Recommended Practice for Ada As a Program Design Language," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1012] IEEE Std 1012-1986, "IEEE Standard for Software Verification and Validation Plans," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1016] IEEE Std 1016-1987, "IEEE Recommended Practice for Software Design Descriptions," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1042] IEEE Std 1042-1987, "IEEE Guide to Software Configuration Management," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1058] IEEE Std 1058.1-1987, "IEEE Standard for Software Project Management Plans," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1063] IEEE Std 1063-1987, "IEEE Standard for Software User Documentation," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1074] IEEE Std 1074-1991, "IEEE Standard for Software Life Cycle Processes," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1209] IEEE Std 1209-1992, "IEEE Recommended Practice for the Evaluation and Selection of CASE Tools," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1219] IEEE Std 1219-1992, "IEEE Standard for Software Maintenance," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1228] IEEE Std 1228-1994, "IEEE Standard for Software Safety Plans," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1348] IEEE Std 1348-1995, "IEEE Recommended Practice for the Adoption of CASE Tools," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[IEEE1498] IEEE Std 1498-1995 (Trial Use Standard-July 1995), "Standard for Information Technology Software Life Cycle Processes: Software Development Acquirer-Supplier Agreement," IEEE Software Engineering Standards, IEEE Service Center, Piscataway, NJ.

[ISO6592] "Information Processing - Guidelines for the Documentation of Computer-Base Application System," First Edition, International Standards Organization, 1985.

[ISO9000-3] ISO 9000-3:1991, "Quality Management and Quality Assurance Standards - Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software," International Standards Organization (ISO), 1991.

[ISO9127] "Information Processing Systems - User Documentation and Cover Information for Consumer Packages," First Edition, International Standards Organization, 1988.

[ISO9294] "Information Technology - Guidelines for the Management of Software Documentation," First Edition, TR9294, International Standards Organization, 1990.

[K&P] Kernighan, B., and Plauger, P., "Elements of Programming Style," 2nd ed., McGraw-Hill, 1978.

[MIL498] MIL-STD-498[1], "Software Development and Documentation," Department of Defense, December 5, 1994. Documentation Data Item Descriptions include:

DI-IPSC-81427	Software Development Plan
DI-IPSC-81428	Software Installation Plan
DI-IPSC-81429	Software Transition Plan
DI-IPSC-81430	Operational Concept Description
DI-IPSC-81431	System/Subsystem Specification
DI-IPSC-81432	System/Subsystem Design Description
DI-IPSC-81433	Software Requirements Specification
DI-IPSC-81434	Interface Requirements Specification
DI-IPSC-81435	Software Design Description
DI-IPSC-81436	Interface Design Description
DI-IPSC-81437	Database Design Description
DI-IPSC-81438	Software Test Plan
DI-IPSC-81439	Software Test Description
DI-IPSC-81440	Software Test Report
DI-IPSC-81441	Software Product Specification
DI-IPSC-81442	Software Version Description
DI-IPSC-81443	Software User Manual
DI-IPSC-81444	Software Center Operator Manual
DI-IPSC-81445	Software Input/Output Manual
DI-IPSC-81446	Computer Operation Manual
DI-IPSC-81447	Computer Programming Manual
DI-IPSC-81448	Firmware Support Manual

[MIL973] MIL-STD-973, "Configuration Management," Department of Defense, April 17, 1992.

[MUSA] Musa, J., Iannino, A., and Okumoto, K., Reliability: Measurement, Prediction Applications, McGraw-Hill Book Co., New York, 1987.

[NASA] "Manager's Handbook for Software Development," National Aeronautics and Space Administration, SEL-84-101(Revision 1), November 1990.

[NQA-1] "Quality Assurance Program Requirements for Nuclear Facilities," American National Standards Institute and American Society of Mechanical Engineers, March 31, 1990.

[1] This document supersedes DOD-STD-2167A and DOD-STD-7935A. This new standard defines a set of activities and documentation suitable for the development of both weapon systems and automated information systems.

[NQA-2 Pt 2.7] "Quality Assurance Requirements of Computer Software for Nuclear Facility Applications," American National Standards Institute and American Society of Mechanical Engineers, May 31, 1990.

[NUREG/CR4640] Bryant, J., and Wilburn, N., "Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry," August 1987.

[PARNAS] Parnas, D., Madey, J., and Iglewski, M., "Precise Documentation of Well-Structured Programs," IEEE Transactions on Software Engineering, December 1994.

[PGWR] "Process Guidelines for Sandia WR Software Development," SAND88-0024, July 1992.

[PPSD] "Sandia Preferred Processes for Software Development," Sandia National Laboratories, Issue 1, February 9, 1994.

[QC-1] "DOE/AL Quality Principles and Requirements for Nuclear Weapons Complex Research, Design, Development, Production, Dismantlement, Maintenance, Stockpile Evaluation, and Disassembly/Disposal," Revision 8, July 17, 1995.

[SDSM] "Software Development/Support Methodology," SAND88-2135, August 1988.

[SEI-CMM] Paulk, M., Curtis, B., Chrissis, M., and Weber, C., "Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, CMU/SEI-93-TR-24, February 1993.

[SEI-CORE] Carleton, A., Park, R., Goethert, W., Florac, W., Bailey, E., and Pfleeger, S., "Software Measurement for DoD Systems: Recommendations for Initial Core Measures," Software Engineering Institute, CMU/SEI-92-TR-19, September 1992.

[SEI-DEFECT] Florac, W., "Software Quality Measurement: A Framework for Counting Problems and Defects," Software Engineering Institute, CMU/SEI-92-TR-22, September 1992.

[SEI-EFFORT] Goethert, W., Bailey, E., and Busby, M., "Software Effort and Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information," Software Engineering Institute, CMU/SEI-92-TR-21, September 1992.

[SEI-SCE] Humphrey,W., "A Method for Assessing the Software Engineering Capability of Contractors," Software Engineering Institute, CMU/SEI-87-TR-23, September 1987.

[SEI-SEPG] Fowler, P., and Rifkin, S., "Software Engineering Process Group Guide," Software Engineering Institute, CMU/SEI-90-TR-24, September 1990.

[SEI-SIZE] Park, R., "Software Size Measurement: A Framework for Counting Source Statements," Software Engineering Institute, CMU/SEI-92-TR-20, September 1992.

[SLP1011] "Software Management," Sandia Laboratories Policy, April 1, 1993.

[SSG] Sandia Software Guidelines, Volumes 1-5, Sandia National Laboratories.
[SSGv1] Volume 1: Software Quality Planning
[SSGv2] Volume 2: Documentation
[SSGv3] Volume 3: Standards, Practices, and Conventions
[SSGv4] Volume 4: Configuration Management
[SSGv5] Volume 5: Tools, Techniques, and Methodologies

[STSC] Software Technology Support Center, Ogden Logistics Center
[STSCsm] Software Management Guide, April 1994.
[STSCsr] Requirements Analysis & Design Tools Report, April 1994.
[STSCsc] Source Code Static Analysis Tools Report, April 1994.
[STSCst] Test Preparation, Execution and Analysis Tools Report, April 1994.
[STSCre] Re-engineering Tools Report, April 1994.
[STSCdo] Documentation Technology Report, April 1994.
[STSCgd] Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapons Systems, Command and Control Systems, Management Information Systems, February 1995.

[SQAS-CASE] SQAS93-001, "CASE Tools: Culture and Implementation," Software Quality Assurance Subcommittee, January 1993.

[SQAS-GLOSS] SQAS90-001, "NWC Glossary of Preferred Software Engineering Terminology," Software Quality Assurance Subcommittee, October 1990.

[WCRE] "Proceedings of the 2nd Working Conference on Reverse Engineering (WCRE)," IEEE Computer Society, July 14-16, 1995.

[WFO] "WFO Development Standards Notebook," Sandia National Laboratories.

Blank Page

Appendix B: Acronyms

ANSI	American National Standards Institute
ASME	American Society of Mechanical Engineers
CALS	Computer-aided Acquisition and Logistic Support
CALS	Continuous Acquisition and Life-Cycle Support
CASE	Computer-Aided Software Engineering
CD-ROM	Compact Disk-Read Only Memory
CGM	Computer Graphics Metafile
CM	Configuration Management
CMM	Capability Maturity Model
COBOL	Common Business Oriented Language
COTS	Commercial-Off-The-Shelf
CSC	Computer Software Component
CSU	Computer Software Unit
DID	Data Item Description
DoD	Department of Defense
DOE	Department of Energy
DPMM	Documentation Process Maturity Model
DTP	Desk Top Publisher
EP	Engineering Procedure
EPROM	Erasable Programmable Read Only Memory
ESTSC	Energy Science and Technology Software Center
GUI	Graphical User Interface
FORTRAN	Formula Translation Language
IDEA	Integrated Development Environment and Assistant
IEEE	Institute of Electrical and Electronics Engineers
IGES	Initial Graphics Exchange Specification
IRS	Interface Requirements Specification
ISO	International Standards Organization
IV&V	Independent Verification and Validation
KPA	Key Process Area
LAN	Local Area Network
MIL	Military
MS	Microsoft
NCLOC	Non-Commented Source Lines of Code
NIAM	Natural Language Information Analysis Method
NQA	Nuclear Quality Assurance
NWC	Nuclear Weapons Complex
OSTI	Office of Scientific and Technical Information
PC	Personal Computer
PDL	Program Design Language
PDR	Preliminary Design Review
PERT	Performance Evaluation Review Technique
PGWR	Process Guidelines for Sandia Weapons Related Software Development
PPSD	Preferred Process for Software Development
QER	Qualification Evaluation Release
QP	Qualification Plan
RAS	Requirements Allocation Sheet
S/W	Software
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan

SDD	Software Design Description
SDF	Software Development Folder (or File)
SDP	Software Development Plan
SDSM	Software Development/Support Methodology
SEI	Software Engineering Institute
SGML	Standard Generalized Markup Language
SLOC	Source Lines of Executable Code
SLP	Sandia Laboratory Policy
SMP	Software Management Plan
SNL	Sandia National Laboratories
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SQAS	Software Quality Assurance Subcommittee
SRR	Software Requirements Review
SRS	Software Requirements Specification
SSF	Software Support Folder
SSGvn	Sandia Software Guidelines, Volume n
SSMP	Software Support/Maintenance Plan
SSP	Software Safety Plan, Software Security Plan
SSTP	Software System Test Plan
STD	Standard
STSC	Software Technology Support Center
SUG	Software Users Guide
SVVP	Software Verification and Validation Plan
TIE-In	Technology Information Environment for Industry
TIM	Technical Interchange Meeting
WBS	Work Breakdown Structure
WFO	Work For Others
WR	War Reserve
WWW	World Wide Web

Appendix C: Tutorial on Life Cycle Documentation

This appendix reviews a comprehensive set of documentation that could be created as part of a software development project. Only a very limited number of projects would ever require the creation of a document set that contained all, or nearly all, of the documentation discussed. Software documentation provides a mechanism to efficiently communicate information about a software product or process. The application will dictate the format, required delivery time, level of detail, and storage media for the documentation. Documentation can serve as an interface definition between what a customer wants and what a supplier builds. Documentation can facilitate communication of schedule, resource, and engineering details among system and software team members during software development. Documentation can define as-built information for use by the support organization when software changes are needed. Documentation can capture key software characteristics of process and product that serve as a measure of software quality. Documentation can serve as an historical reference for use in lessons learned and future software process improvement. Accurate and timely documentation can reduce rework throughout the software's life cycle.

C.1 Life Cycle Relationships of Key Documents

The following paragraphs describe the software life cycle activities, life cycle processes, and integration of the software activities and processes in terms of their associated documentation. The software documentation provides snapshots of the overall state of the development and maintenance efforts over time. The management-planning documents and software-specific product documents addressed in this appendix include and supplement the many documents described in Chapter 4 of this volume.

The life cycle activities described in the following subsections are addressed as being carried out in sequence but are usually carried out in parallel and iteratively. The sequencing of documentation development as a function of the life cycle activity is depicted in Figure C-1.

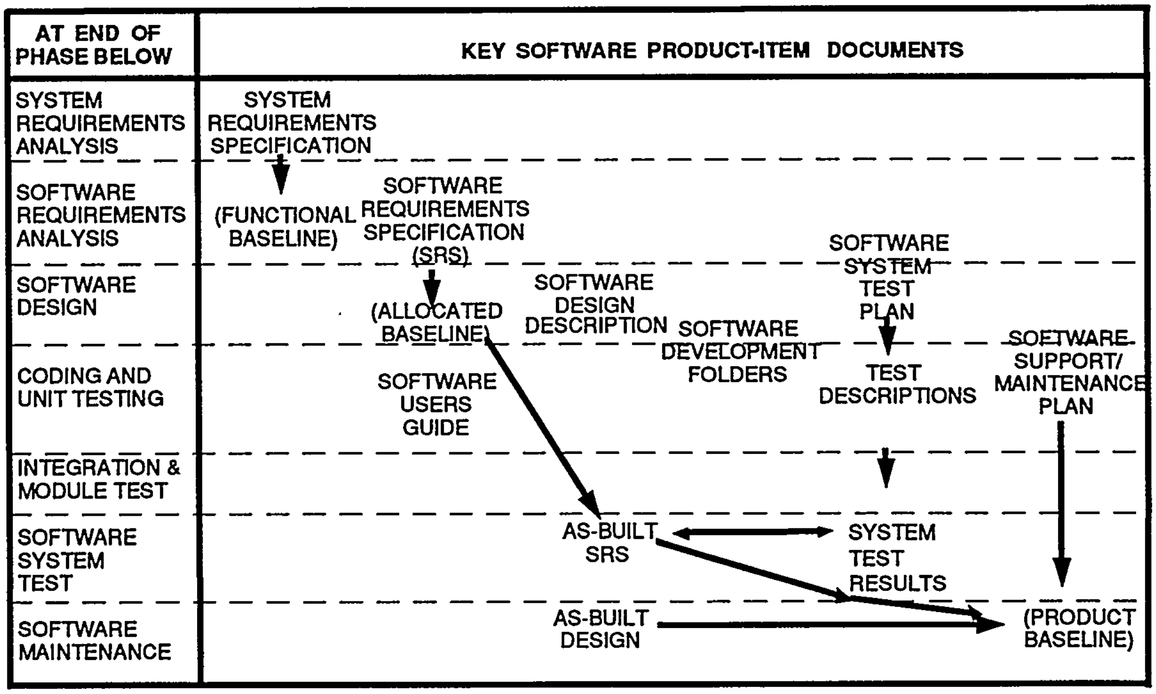

AT END OF PHASE BELOW
KEY SOFTWARE PRODUCT-ITEM DOCUMENTS
SYSTEM REQUIREMENTS ANALYSIS
SOFTWARE REQUIREMENTS ANALYSIS
SOFTWARE DESIGN
CODING AND UNIT TESTING
INTEGRATION & MODULE TEST
SOFTWARE SYSTEM TEST
SOFTWARE MAINTENANCE
SYSTEM REQUIREMENTS SPECIFICATION
(FUNCTIONAL BASELINE)
SOFTWARE REQUIREMENTS SPECIFICATION (SRS)
(ALLOCATED BASELINE)
SOFTWARE DESIGN DESCRIPTION
SOFTWARE SYSTEM TEST PLAN
SOFTWARE DEVELOPMENT FOLDERS
SOFTWARE SUPPORT/ MAINTENANCE PLAN
SOFTWARE USERS GUIDE
TEST DESCRIPTIONS
AS-BUILT SRS
SYSTEM TEST RESULTS
AS-BUILT DESIGN
(PRODUCT BASELINE)

Figure C-1. Relationship of Key Software Product-Item Documents to Standard Software Life Cycle Activities

C.1.1 Description of System Requirements Analysis Activities

The first life cycle activity, System Requirements Analysis, shown in Figure C-1 is not a primary software effort although software leaders may participate. The main objective of System Requirements Analysis is to identify, document, and gain customer approval for all the functional, external interface, performance, and miscellaneous requirements. In addition, design constraints, reliability measures, and availability measures are identified. When the System Requirements Specification exists, or is developed, it will be used as a starting point for software development. When the System Requirements Specification is created as part of the project, development of that document is normally the responsibility of the system project manager. At the end of this activity, the customer-approved System Requirements Specification should be brought under project configuration control as the Functional Baseline for the system.

Software-related processes and activities may be carried out to develop the plans and schedules that will be used to manage and control the software development activities. Existing software project management documentation can be tailored based upon the System Requirements Specification. The following documents are potential targets for modification or creation during System Requirements Analysis:

- Software Management Plan (SMP);
- Software Development Plan (SDP);
- Software Standards, Practices, and Conventions (may be tailored using guidance in reference [SSGv3]); could exist as a separate document or as an appendix in the SDP;
- Software Quality Assurance Plan (SQAP);
- Software Configuration Management Plan (SCMP);
- Software Verification and Validation Plan (SVVP); and
- Software Project Schedule.

C.1.2 Description of Software Requirements Analysis

The *Sandia Preferred Process for Software Development (PPSD)*, see reference [PPSD], lists and defines nine major tasks related to the process for development and maintenance of software requirements. Those tasks include:

- Task 1: Define and schedule Software Requirements Specification (SRS) activities;
- Task 2: Write SRS Overview and get customer feedback;
- Task 3: Prepare Draft SRS;
- Task 4: Schedule SRS Inspection and distribute SRS;
- Task 5: Inspect SRS;
- Task 6: Resolve inspection issues;
- Task 7: Get customer feedback;
- Task 8: Approve and distribute SRS; and
- Task 9: Maintain the approved SRS.

The first technical software effort in a software project is normally the Software Requirements Analysis activity. The objective is to identify the system requirements that are to be satisfied by application software and the interface requirements that have to be satisfied between the software being developed by the project and the remaining system components, which may consist of both hardware and software. These allocated requirements are documented in one or two specification documents. The Software Requirements Specification (SRS) may address both types of requirements. However, on large projects it may be useful to document the interface features in a separate Interface Requirements Specification (IRS). Prior to being submitted for approval, the requirements documents should be subjected to a formal software requirements inspection, and all defects found should be corrected. After approval, these documents provide the basis for design of the application software and the software components necessary to satisfy the interfaces to the rest of the system. After being approved, the requirements specification(s) should be brought under configuration management control as the Allocated Baseline for the software being developed.

When a Software Requirements Review (SRR) is required, the approved version of the requirements document(s) is the basis for the review. In addition, these documents will also form the technical basis against which the software architecture will be analyzed during a Preliminary Design Review (PDR). Depending on the size of the development project and the desires of the customer, a PDR may address the entire system, or only the software that is being developed for the system.

C.1.3 Description of Software Design

After the requirements documents have been baselined, they form the basis for deriving the preliminary design of the software being developed. Typically design activities consist of two design processes: preliminary design and detailed design. In many cases detailed design activities may begin before the preliminary design has been fully completed. The reference [PPSD] defines eleven major tasks related to the software design process. Those tasks include:

- Task 1: Gather, read, and understand information on which the design is to be dependent;
- Task 2: Identify consultants and interfaces;
- Task 3: Hold design brainstorming meeting to identify alternative design approaches;
- Task 4: Investigate and evaluate alternative design approaches;
- Task 5: Produce preliminary (top-level) and detailed design and define data dictionary;
- Task 6: Complete the draft of the Software Design Description;
- Task 7: Schedule the Software Design Description (SDD) inspection(s) and distribute the draft SDD to members of the Inspection Team (Note: for large systems there may be more than one SDD, hence, more than one SDD inspection; e.g., one for each major component);
- Task 8: Inspect the SDD(s);
- Task 9: Resolve inspection issues and report the inspection results to management;
- Task 10: Approve and distribute the inspected [and corrected] SDD; and
- Task 11: Maintain the approved SDD.

Preliminary Software Design

In a structured design approach, the preliminary software design process principally consists of decomposing the overall system into lower level elements, often referred to as Computer Software Components (CSCs). The top-level CSCs are then further decomposed into lower level CSCs or into units that are often referred to as Computer Software Units (CSUs). An example software component structure resulting from such a decomposition is depicted in Figure C-2. Decomposition is commonly carried down to lower levels until each CSU typically represents a code element that will carry out a single, unique function. Altogether, the lower level CSUs result in a set of code elements that, when implemented, will satisfy all the requirements allocated to software and identified in either the SRS or the IRS.

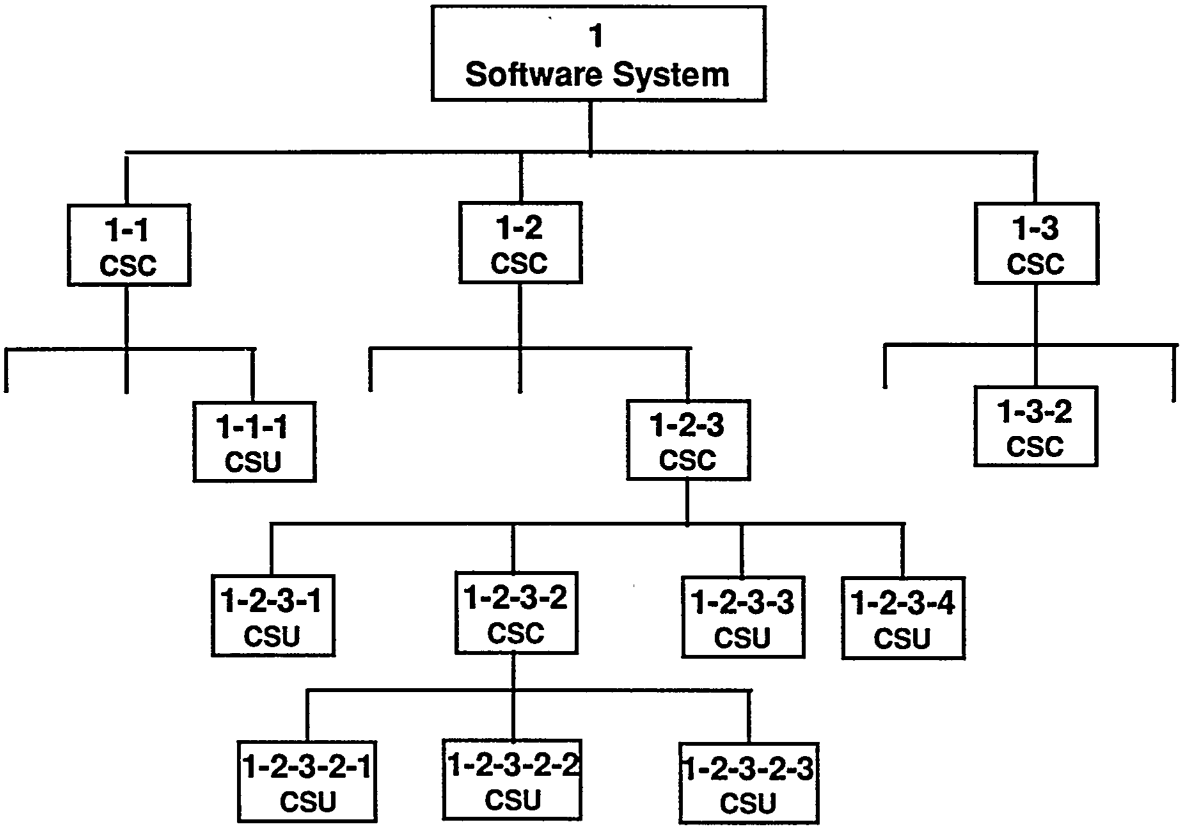


Figure C-2. Decomposition of Software Into Component Elements

The preliminary design is basically complete when the decomposition structure is fully defined and documented in a SDD. This design should then be subjected to one or more design inspections, depending on the size of the software system that is to be developed. Any defects noted during these inspections should be corrected before commencement of the detailed design process for the associated code elements. Following preliminary design inspections, the corrected preliminary design may be brought under informal control as the initial version of the Developmental Configuration.

The design inspections and later design, implementation, and testing activities will be facilitated by beginning development of a Traceability Matrix. This matrix should map every one of the software requirements onto the CSCs and CSUs under which it is to be satisfied.

If a Software Users Guide is to be developed, it should be drafted as a part of the design activities. The Software Users Guide is should be initiated prior to the completion of the preliminary design activity and completed during the detailed design activity. When such a document is produced, it also should be inspected. This inspection is useful in detecting defects in requirements and design.

Detailed Software Design

The objective of detailed design for software is to provide the basis whereby each individual computer software unit can be coded directly from its detailed design. Detailed designs may be documented in the form of flow charts, structure charts, state transition diagrams, schema and schemata (for databases when database managers are being used to support the implementation), database design (for systems that will not use a database manager), program design language, pseudo code, etc. A copy of all requirements and design features allocated to a given CSU should be placed in a Software Development Folder (SDF) for that unit. Each of these folders will be the initial basis for beginning coding, design, and execution of unit testing of the associated CSU.

C.1.4 Description of Coding and Unit Testing

The reference [PPSD] defines seven tasks related to the software implementation process. All of these tasks are applicable to each of the individual code units (CSUs). The seven tasks include:

- Task 1: Prepare for coding through familiarization with the development environment, the development language, the overall design being implemented, and the portions of the SDD that are applicable to the assigned unit to be coded;
- Task 2: Develop source code that implements the logic of the design for that code unit;
- Task 3: Schedule code inspection for the CSU and distribute the inspection package;
- Task 4: Inspect code unit;
- Task 5: Resolve code inspection issues and report the results of the software inspections to management;
- Task 6: Unit test each CSU and correct defects found; and
- Task 7: Maintain the code for each CSU.

C.1.5 Description of Software Integration and Testing

Software Integration and Testing is particularly applicable during the development of larger software systems but can be useful for medium and even small systems. These testing activities involve integrating the software units and testing the interfaces between the units after each new unit has been added to the integrated module. The integration strategy can be top-down (requiring stubs) or bottom-up (requiring drivers). Using Figure C-2 as an example, Level 1-2-3-2-1 (CSU) might be integrated with Level 1-2-3-2 (CSC) and the resulting module tested. Then Level 1-2-3-2-2 (CSU) could be added to the existing two-unit module and the combined three unit module tested together; followed by adding Level 1-2-3-2-3 (CSU) to the existing three unit module and the combined four unit module tested. The next step would be linking Level 1-2-3-2 (CSC) and its three subordinate units to Level 1-2-3 (CSC) for testing. The next step would be to link Level 1-2-3-1 (CSU) to Level 1-2-3 (CSC) for testing of the six unit module, and so forth until all CSUs in the branch under Level 1-2-3 (CSC) have been integrated with their parent Level 1-2 (CSC) and tested for connectivity and data passing. After that has been completed, the next step would be to link all three CSCs together in preparation for software system testing.

C.1.6 Description of Software System Testing

The reference [PPSD] defines ten major tasks related to the software system testing process. Those tasks include:

- Task 1: Identify system software tests that are driven by the requirements in the SRS;
- Task 2: Identify test environments driven by the SRS requirements;
- Task 3: Complete the Software System Test Plan (SSTP);
- Task 4: Schedule the SSTP inspection;
- Task 5: Inspect the SSTP;
- Task 6: Resolve SSTP inspection issues;
- Task 7: Approve and distribute the SSTP;
- Task 8: Execute tests and software to correct problems;
- Task 9: Rerun failed tests; and
- Task 10: Maintain approved SSTP.

The first two tasks for system software testing should be started a short time before the end of the Software Requirements Analysis activities. Starting these tasks early will help identify defects in the SRS related to the applicability of individual tests or potential for verification through demonstration, analysis, or inspection. The SSTP developed and corrected during tasks 3 through 7 should address how each requirement in the SRS will be verified. The necessary verification method(s) for each of the requirements can be described under the Test Descriptions section of the SSTP. Each test description addresses the following:

- brief description of the test;
- regression testing requirements relative to changes that have been made to the code;
- identification of the requirements from the SRS that will be verified by that test;
- cross references to any associated documents;
- step-by-step description of the test process; and
- identification of any dependencies between tests; i.e., which tests must have been completed before the test being described can be carried out.

C.1.7 Description of Software Maintenance

Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, adapt to a changed environment, or add new functionality. Maintenance can be divided into four basic subtasks and their underlying steps:

- Task 1: Change Analysis:
  ⇒ Change Identification and Classification
  ⇒ Change Feasibility Analysis
  ⇒ Change Detailed Analysis
- Task 2: Change Development:
  ⇒ Modify Software Requirements Specification
  ⇒ Modify Software Design Description
  ⇒ Modify Source Code
  ⇒ Test Software Units and Components
  ⇒ Review Unit and Component Test Results
- Task 3: Change Testing:
  ⇒ Update Software System Test Plan, Test Procedures, and Regression Test Cases
  ⇒ Conduct System Integration Test Cases
  ⇒ Analyze Test Failures
  ⇒ Review and Report Test Results
- Task 4: Change Release:
  ⇒ Develop Delivery Package
  ⇒ Verify Delivery Package
  ⇒ Release Delivery Package
  ⇒ Deliver and Install Delivery Package
  ⇒ Train Users on Delivery Package Features.

C.2 Project Management Documentation

The following paragraphs discuss project management documentation that addresses project plans, schedules, costing, resource requirements, software measurement and metrics documents, and recording of lessons learned information. Applicable documents that might include the project management documentation include: software project/development plan, software quality plan, software configuration management plan, software support plan, software verification and validation plan, software safety/security plan, software management notebook, and software development folder.

C.2.1 Project Plans, Schedules, and Resource Requirements Documentation

When developed, the Software Development Plan provides the approach to be followed in managing a software project. The ability to manage that project effectively often hinges on following the defined approach and is based on the schedule developed for execution of the project. Such a schedule becomes the baseline of completion dates for activities to be carried out and may define the expected resource requirements necessary to meet those dates. A commonly used method for development of such a schedule consists of the following sequence of activities:

Step 1: decompose the project activities identified in Figure C-1 into lower level work units through development of a work breakdown structure (WBS); Figure C-3 contains a detailed example of a partial WBS for a software project;
Step 2: define the precedence relationships between the work units in the WBS;
Step 3: create a network schedule based on the defined work units and precedence relationships using Critical Path Methodology; this may be facilitated using a software package such as Microsoft Project, TimeLine, or Super Project;
Step 4: allocate staff member resources to individual work units;
Step 5: allocate other resources to individual work units;
Step 6: reconcile schedule defects (e.g., excess durations, excess resource requirements during peak periods);
Step 7: print schedule in appropriate formats (e.g., network or PERT diagrams, Gantt charts, resource requirements versus time, cost versus time);
Step 8: coordinate approval of reconciled schedule by management and customers;
Step 9: baseline approved schedule; and
Step 10: copy baseline schedule to create initial actual schedule that will be updated on a periodic basis, and compared against the baseline schedule to support project tracking.

- 1\. Development of (Name of Software Project)
  - 1.1 Project Management
    - 1.1.1 General Project Management
      - 1.1.1.1 Project Budgeting
      - 1.1.1.2 Technical Interchange Meetings
      - 1.1.1.3 Technical Reviews
        - 1.1.1.3.1 Software Requirements Review(s)
        - 1.1.1.3.2 Software Preliminary Design Review(s)
        - 1.1.1.3.3 Software Detailed Design Review(s)
        - 1.1.1.3.4 Software Test Readiness Review(s)
      - 1.1.1.4 Project Close-out
    - 1.1.2 Baseline Scheduling
      - 1.1.2.1 Analysis of System Requirements
      - 1.1.2.2 Identification of Major Software Components
      - 1.1.2.3 Determination of Documentation Requirements
      - 1.1.2.4 Development of Work Breakdown Structure
      - 1.1.2.5 Definition of Precedence Relationships
      - 1.1.2.6 Development of Network Schedule
      - 1.1.2.7 Allocation of Resources to Project Activities
    - 1.1.3 Project Tracking
      - 1.1.3.1 Collection of Progress Data
      - 1.1.3.2 Collection of Effort Data
      - 1.1.3.3 Schedule Updating
      - 1.1.3.4 Schedule Adjustment
      - 1.1.3.5 Schedule Status Reporting

Figure C-3. Sample Software Work Breakdown Structure: Part 1 of 5

- 1.1.4 Software Staff Management
  - 1.1.4.1 Staff Initiation
    - 1.1.4.1.1 Reassignment of Current Staff
    - 1.1.4.1.2 Hiring of New Staff
  - 1.1.4.2 Staff Allocation
  - 1.1.4.3 Staff Training
- 1.1.5 Software Quality Assurance
  - 1.1.5.1 Definition of Quality Measures
  - 1.1.5.2 Collection of Quality Measures
  - 1.1.5.3 Analysis of Quality Data
  - 1.1.5.4 Reporting of Quality Status
- 1.1.6 Software Configuration Management
  - 1.1.6.1 Configuration Identification
  - 1.1.6.2 Configuration Management
    - 1.1.6.2.1 Software Allocated Baseline Management
    - 1.1.6.2.2 Software Developmental Configuration Management
      - 1.1.6.2.2.1 Software Development Folder Management
      - 1.1.6.2.2.2 Software Code Unit Management
      - 1.1.6.2.2.3 Software Development Library
    - 1.1.6.2.3 Software Product Baseline Management
      - 1.1.6.2.3.1 Software Product Version 1.0 Management
      - 1.1.6.2.3.2 Software Product Version 2.0 Management
      - 1.1.6.2.3.i Software Product Version i.0 Management
      - 1.1.6.2.3.n Software Product Version n.0 Management
  - 1.1.6.3 Software Configuration Control Board
    - 1.1.6.3.1 Software Problem Report Management
      - 1.1.6.3.1.1 Software Problem Report Recording and Tracking
      - 1.1.6.3.1.2 Software Problem Report Analysis
      - 1.1.6.3.1.3 Software Problem Report Close Out
    - 1.1.6.3.2 Software Change Management
      - 1.1.6.3.2.1 Software Change Analysis
      - 1.1.6.3.2.2 Software Change Review
  - 1.1.6.4 Configuration Status Accounting
    - 1.1.6.4.1 Configuration Status Analysis
    - 1.1.6.4.2 Configuration Status Reporting
  - 1.1.6.5 Configuration Audits
    - 1.1.6.5.1 Functional Configuration Audit
      - 1.1.6.5.1.1 Traceability of Requirements Into Test Reports
      - 1.1.6.5.1.2 Validation of Test Results For Each Requirement
    - 1.1.6.5.2 Physical Configuration Audit
      - 1.1.6.5.2.1 Verification Software Requirements Specification Reflects System Tested Codes
      - 1.1.6.5.2.2 Verification Software Design Description Reflects System Tested Codes
      - 1.1.6.5.2.3 Verification Source Code Listings Reflect System Tested Codes

- 1.2 Process Documentation
  - 1.2.1 Software Requirements Documents
    - 1.2.1.1 Software Requirements Specification
    - 1.2.1.2 Interface Requirements Specification

Figure C-3. Sample Software Work Breakdown Structure: Part 2 of 5

1.2.2 Design Documents
1.2.2.1 Documentation of Preliminary Design/Structure
1.2.2.2 Software Design Description
1.2.3 Coding Activities Documents
1.2.3.1 Unit Development Folders
1.2.3.1.1 Unit 1 Development Folder
1.2.3.1.2 Unit 2 Development Folder
1.2.3.1.i Unit i Development Folder
1.2.3.1.n Unit n Development Folder
1.2.3.2 Unit Tests
1.2.3.2.1 Unit 1 Tests
1.2.3.2.2 Unit 2 Tests
1.2.3.2.i Unit i Tests
1.2.3.2.n Unit n Tests
1.2.3.3 Module Tests
1.2.3.3.1 Module 1 Tests
1.2.3.3.2 Module 2 Tests
1.2.3.3.i Module i Tests
1.2.3.3.n Module n Tests
1.2.4 System Testing Documents
1.2.4.1 Software System Test Plan
1.2.4.2 Software Test Descriptions
1.2.4.2.1 Software Test 1 Description
1.2.4.2.2 Software Test 2 Description
1.2.4.2.i Software Test i Description
1.2.4.2.n Software Test n Description
1.2.4.3 Traceability of Requirements Into Test Descriptions
1.2.4.4 Software Test Procedures
1.2.4.4.1 Software Test 1 Procedures
1.2.4.4.2 Software Test 2 Procedures
1.2.4.4.i Software Test i Procedures
1.2.4.4.n Software Test n Procedures
1.2.4.5 Software Test Reports
1.2.4.5.1 Software Test 1 Report
1.2.4.5.2 Software Test 2 Report
1.2.4.5.i Software Test i Report
1.2.4.5.n Software Test n Report
1.3 Requirements Analysis
1.3.1 Analysis of Allocated Systems Requirements
1.3.2 Functional Requirements Analysis
1.3.3 Performance Requirements Analysis
1.3.4 Interface Requirements Analysis
1.3.4.1 Internal Interface Requirements Analysis
1.3.4.2 External Interface Requirements Analysis
1.3.5 Design Constraint Analysis
1.3.5.1 Compliance Standards Requirements Analysis
1.3.5.2 Hardware Limitations Analysis

Figure C-3. Sample Software Work Breakdown Structure: Part 3 of 5

- 1.3.6 Attribute Requirements Analysis
  - 1.3.6.1 Security/Safety/Integrity Requirements Analysis
  - 1.3.6.2 Reliability Requirements Analysis
  - 1.3.6.3 Availability Requirements Analysis
  - 1.3.6.4 Maintainability Requirements Analysis
  - 1.3.6.5 Operability Requirements Analysis
  - 1.3.6.6 Transportability Requirements Analysis
- 1.3.7 Miscellaneous Requirements Analysis
  - 1.3.7.1 Database Requirements Analysis
  - 1.3.7.2 Operations Requirements Analysis
  - 1.3.7.3 Site Requirements Analysis
- 1.3.8 Initiation of Requirements Traceability
- 1.3.9 Software Requirements Inspection(s)

1.4 Design

- 1.4.1 Preliminary Design
  - 1.4.1.1 Design Rationale
  - 1.4.1.2 Module Decomposition
    - 1.4.1.2.1 Module 1 Decomposition
    - 1.4.1.2.2 Module 2 Decomposition
    - 1.4.1.2.i Module i Decomposition
    - 1.4.1.2.n Module n Decomposition
  - 1.4.1.3 Data Decomposition
  - 1.4.1.4 Traceability of Requirements Into Preliminary Design
  - 1.4.1.5 Software Preliminary Design Inspection(s)
- 1.4.2 Detailed Design
  - 1.4.2.1 Module Detailed Design
    - 1.4.2.1.1 Module 1 Detailed Design .
    - 1.4.2.1.2 Module 2 Detailed Design
    - 1.4.2.1.i Module i Detailed Design
    - 1.4.2.1.n Module n Detailed Design
  - 1.4.2.2 Data Detail Design
    - 1.4.2.2.1 Data Entity 1 Detailed Design
    - 1.4.2.2.2 Data Entity 2 Detailed Design
    - 1.4.2.2.i Data Entity i Detailed Design
    - 1.4.2.2.n Data Entity n Detailed Design
  - 1.4.2.3 Traceability of Requirements Into Detailed Design
  - 1.4.2.4 Unit Testing Strategy
  - 1.4.2.5 Software Detailed Design Inspection(s)

1.5 Coding and Unit Testing

- 1.5.1 Code Unit 1 Development and Testing
  - 1.5.1.1 Unit 1 Coding
  - 1.5.1.2 Unit 1 Testing
  - 1.5.1.3 Unit 1 Code Inspection
- 1.5.2 Code Unit 2 Development and Testing
  - 1.5.2.1 Unit 2 Coding
  - 1.5.2.2 Unit 2 Testing
  - 1.5.2.3 Unit 2 Code Inspection

Figure C-3. Sample Software Work Breakdown Structure: Part 4 of 5

- 1.5.i Code Unit i Development and Testing
    - 1.5.i.1 Unit i Coding
    - 1.5.i.2 Unit i Testing
    - 1.5.i.3 Unit i Code Inspection
- 1.5.n Code Unit n Development and Testing
    - 1.5.n.1 Unit n Coding
    - 1.5.n.2 Unit n Testing
    - 1.5.n.3 Unit n Code Inspection

- 1.6 Integration Testing
    - 1.6.1 Module 1 Integration and Testing
        - 1.6.1.1 Module 1 Integration
        - 1.6.1.2 Module 1 Testing
    - 1.6.2 Module 2 Integration and Testing
        - 1.6.2.1 Module 2 Integration
        - 1.6.2.2 Module 2 Testing
    - 1.6.i Module i Integration and Testing
        - 1.6.i.1 Module i Integration
        - 1.6.i.2 Module i Testing
    - 1.6.n Module 1 Integration and Testing
        - 1.6.n.1 Module n Integration
        - 1.6.n.2 Module n Testing
- 1.7 Software System Testing
    - 1.7.1 Software System Test 1
        - 1.7.1.1 Software System Test 1 Preparations
        - 1.7.1.2 Software System Test 1 Execution
        - 1.7.1.3 Software System Test 1 Data Analysis
    - 1.7.2 Software System Test 2
        - 1.7.2.1 Software System Test 2 Preparations
        - 1.7.2.2 Software System Test 2 Execution
        - 1.7.2.3 Software System Test 2 Data Analysis
    - 1.7.i Software System Test i
        - 1.7.i.1 Software System Test i Preparations
        - 1.7.i.2 Software System Test i Execution
        - 1.7.i.3 Software System Test i Data Analysis
    - 1.7.n Software System Test n
        - 1.7.n.1 Software System Test n Preparations
        - 1.7.n.2 Software System Test n Execution
        - 1.7.n.3 Software System Test n Data Analysis
- 1.8 Software Installation Support
    - 1.8.1 Software-Hardware Integration
    - 1.8.2 System Installation
    - 1.8.3 System Acceptance Testing

Figure C-3. Sample Software Work Breakdown Structure: Part 5 of 5

C.2.2 Project Costing and Resource Analysis Documentation

The baseline schedule described in the preceding subsection documents the project cost and resource utilization. During the execution of the project it will be necessary to collect the data required to determine how well the project is progressing versus the assigned budget and forecast utilization of resources. These data can then be input into the actual schedule to provide documentation of actual costs and resource usage against the scheduled values. The actual data required to carry out this activity are described in the following subsection.

C.2.3 Software Measurement and Metric Documentation

A wide variety of potential software measurements and metrics have been developed. Generally, these can be divided into three categories: (1) Resource Measurements; (2) Process Measurements; and (3) Product Measurements. Resource metrics refer to the hardware, software tools, office space, and personnel needed to perform the development of software. Process metrics refer to task time and effort during requirements analysis, design, coding, and testing processes used to develop software. The associated metrics include elapsed time and elapsed effort used to complete a task, and efficiency in removing defects. Product metrics are quality measures of the documents, code listings, user manuals, test reports, and executable software product output from the processes.

For the purposes of this document, only basic data collection and analysis documentation is described. Basic measurements are normally used to support estimation and planning, project tracking, and defect measurement. These measurements include software size, effort required, schedule progress, and product defects. Software measurement goals and the use of metric information is normally documented in a Project Management Plan or is included in project tracking data reports. The core measures recommended for initial implementation are identified in Figure C-4. See reference [SEI-CORE] for more discussion on core measures.

Unit of Measure	Characteristic Addressed
Counts of physical source lines of code	Size, Progress, Reuse
Counts of staff hours expended	Effort, Cost, Resource Utilization
Calendar dates (major events/milestones)	Schedule, Progress
Counts of software problems and defects	Quality, Acceptability for Delivery, Improvement Trends
Mean Time Between Failures (due to software failures)	Operational Reliability

Figure C-4. Core Measures Recommended for Initial Implementation

C.2.3.1 Software Size

Some of the more popular and effective measures of software size are non-commented physical source lines of code (NCSLOC), logical source statements (instructions), function points (feature points), and counts of logical functions or computer software units. The measure recommended for physical source lines of code is generally stated as all lines of the source code listing except those that are comments or blank lines. However, such a definition is open to wide variations in interpretation. The Software Engineering Institute (SEI) has developed a guideline document, *Software Size Measurement: A Framework for Counting Source Statements* [SEI-SIZE], that assists an organization in determining which interpretation it will use. In addition, Appendix E of that document provides detailed checklist forms for defining counts of source statements.

C.2.3.2 Software Effort Required and Progress Against the Planned Schedule

Reliable measures of software effort are prerequisites for reliable measures of software cost. They are also important in a more direct way. The principal means for managing and controlling costs and schedules is through planning and tracking the human resources assigned to individual tasks and activities. The recommended unit of effort is the staff-hour. A staff hour can be defined as an hour of time actually expended by a member of the staff on activities directly related to software Work Breakdown Structure activities. The use of staff-hours is recommended rather than staff-weeks or labor-months because the number of hours comprising the latter two may not be the same across organizations or projects.

The SEI has developed a guideline document, *Software Effort and Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information* [SEI-EFFORT], that will assist an organization in defining which staff hours are included and which are excluded when counting actual staff effort expended in software development. In addition, that document provides detailed information relative to defining a framework for measurement of progress versus the planned schedule.

C.2.3.3 Software Defects

Defining what a customer or user views as true software quality can be elusive. Whatever the criteria, it is clear that the number of problems and defects associated with a software product varies inversely with perceived quality. The following are the SEI's definitions of the terms defects and problems:

Problem: A software problem is a human encounter with software that causes difficulty, doubt, or uncertainty in the use or examination of the software. Examples include: (1) a difficulty encountered with a software product or software work product resulting from an apparent failure, misuse, misunderstanding, or inadequacy; (2) a perception that the software product or software work product is not behaving or responding according to specification; (3) an observation that the software product or software work product is lacking function or capability needed to complete a task or work effort.

Defect: A software defect is: (1) any unintended characteristic that impairs the utility or worth of an item; (2) any kind of shortcoming, imperfection, or deficiency; (3) any flaw or imperfection in a software work product or software process. Examples include such things as mistakes, omissions and imperfections in software artifacts, or faults contained in software sufficiently mature for test or operation.

Counts of software problems and defects allow qualitative description of trends in detection and correction activities. They also allow the tracking of progress in identifying and fixing process and product imperfections. In addition, problem and defect measures are the basis for quantifying other software quality attributes such as reliability, correctness, completeness, efficiency, and usability.

Again, the SEI has developed a guideline document, *Software Quality Measurement: A Framework for Counting Problems and Defects* [SEI-DEFECT], that will assist an organization in defining the basis for identifying and counting problems and defects.

C.2.3.4 Software Failure Rate and Software Reliability

Two terms commonly associated with software problems are defined in reference [IEEE610] as:

- **Failure** - the inability of a system or component to perform its required functions within the specified performance requirements; and
- **Fault** - a defect in a hardware device or component, or an incorrect step in a process or data definition in a computer program.

From these definitions it is apparent that a fault may exist in software whether or not it is being executed. Conversely, a failure occurs when a fault is encountered during program execution. Given these definitions, failures and defects can both be counted but only failures can be used to derive a rate of occurrence.

Software reliability is defined in statistical terms as the probability of failure free operation of the software in a specified environment for a specified period of time. For this definition to make sense the following terms must be specifically defined:

- **Specified Environment** - refers to the defined conditions within which the software must operate; includes the given machine configuration or physical facilities that are in place during the period of time; hardware failures cannot be counted as software failures;
- **Period of Time** - refers to an interval with a definite starting point and ending point; and
- **Failure** - as defined above; a term that is typically taken to mean nonconformance to requirements and therefore is a relative term; gradations exist within this definition so that failures can be catastrophic or merely annoying.

Based on these definitions, software failure rates can be calculated as the number of failures experienced during a specified period of time. Software reliability depends on the criticality of failures considered, the failure rate, and other considerations related to the failure probability distribution and operational use profiles (see reference [MUSA]).

C.3 Process Documentation

The following paragraphs provide guidelines for documentation of software development and maintenance processes that address:

- software requirements management;
- software design activities;
- software unit coding and testing activities; and
- formal and informal in-process software evaluations such as inspections, walk throughs, reviews, and configuration audits.

These discussions relate the process documentation to the process outputs and typical product documentation, and describe how integration of activities and processes in the software life cycle is supported through development, review, approval, and maintenance of product documentation.

C.3.1 Documenting Software Requirements Management and Analysis Activities

C.3.1.1 Documenting Software Requirements Management Activities

The purpose of software requirements management is to establish a documented, common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project. Requirements management involves establishing and maintaining an agreement with the customer on the requirements for the software project. Software requirements management includes the following activities:

- requirements gathering from customer interactions and system allocations;
- requirements analysis;
- requirements documentation;
- requirements change management; and
- requirements satisfaction survey from customer interactions.

The general approach to documenting software requirements management activities should follow that of change/problem reporting and tracking. Each software requirement specification (or possible requirement) gathered from customer interactions and system allocations should be put into an information base such as used for change/problem reporting and tracking. This requirements gathering and documentation activity provides a basis for the subsequent analysis activity that discriminates which statements and allocations actually become a software requirement. The information to be recorded relative to definition of requirements should include the decisions made during the process and the reasons behind why particular alternatives have been taken versus why the others were rejected.

Those software requirements that are retained from the analysis activity are documented in a Software Requirements Specification and perhaps an Interface Requirements Specification. After inspection and approval, the requirements documents are controlled in accordance with the organization and/or project configuration management procedures. Any subsequent change that would add a new requirement or would impact one or more of the requirements contained in the customer-approved requirements documents is written up on a change proposal form by the person(s) initiating such a request. This action may take place during the software development process or during fielded operational support. The only difference might be in the level of formality followed for the requirements change process. The requirements change management described in the following paragraphs is reasonably formal.

Any change proposals should be processed by the appropriate component of the software development team, preferably the software configuration management activity. The change proposal form should exist throughout the rest of the software's applicable software life cycle. The software configuration management activity should record the request and assign an identification by which the change can be tracked to closure. The change proposal should be presented to a software configuration control board, or appropriate approval authority, that will assign analysis responsibility (if required). Following such an assignment, the responsible group should then arrange for and schedule analyses of :

- potential approaches for its implementation;
- benefit (need for and value) of such a change;
- cost (effort and time) to make the change; and
- potential risks (technical and schedule).

Since the customer-approved requirements documents have been baselined, the results of these analyses are then evaluated by the software configuration control board. If the change request is approved by both that body and the customer, then it should be documented and distributed as an official change to the appropriate requirements document. If the proposal is not approved, then its status should be closed and the originator advised of that action and the reason(s). The next decision for an approved change is determination of how and when the change will be incorporated. In general, the possible alternatives are either immediate implementation or assignment to a later version in which it will be implemented. When the decision is for immediate implementation, the results of the analyses should be used to adjust the baseline and actual schedules, milestones, resource allocations, and the project budget. A customer-initiated change received during the software development effort that necessitates a change in the approved software specifications document is normally considered a valid basis for changing the schedule, milestones, and cost of the project. Requirement changes initiated for other reasons may not be allowed as a basis for such changes in project management considerations. For systems having both hardware and software requirements, software changes that might affect an interface may need to be approved by higher level project configuration management boards that examine hardware/software interface impacts.

Perhaps the most important and most often forgotten part of the requirements management activity is to include periodic surveys or some other mechanism for determining whether the customer is satisfied that the software implements the customer's requirements -- stated and implied. This activity allows for closure of the requirements management activity by documenting lessons learned and providing continual input to a root cause analysis activity that may lead to requirements management process improvement.

C.3.1.2 Documenting Software Requirements Analysis Activities

Documentation of software requirements analysis activities provides a record of the sources and/or reasons for individual requirements that will:

- be useful in gaining customer agreement on the final requirements document;
- support design and trade-off analyses; and
- assist in analyses related to reported problems and proposed changes.

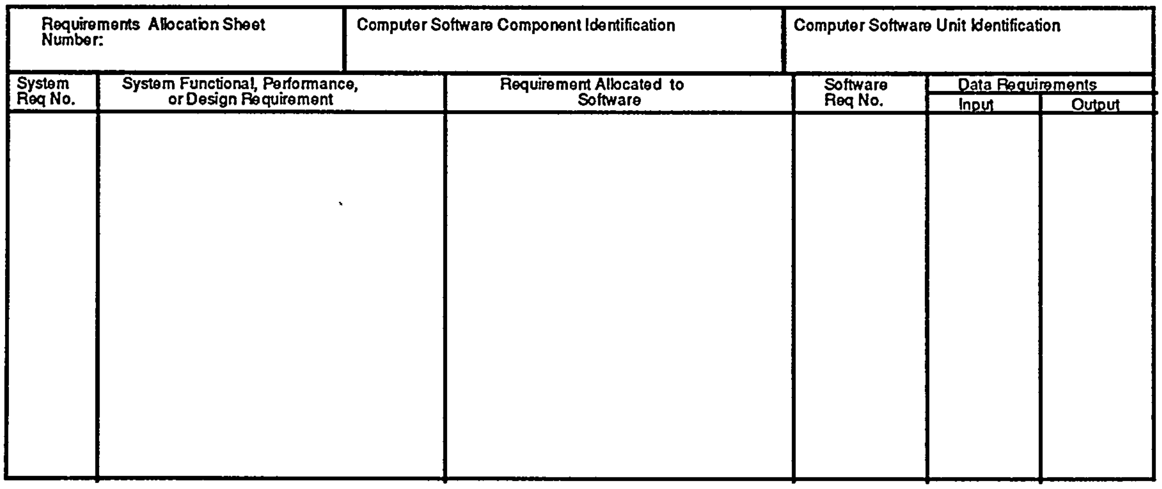
System requirements are analyzed to identify those functions that must be performed to satisfy the objectives of each functional area. Each function is described in terms of inputs, processing, outputs, and interface requirements; subfunctions are identified and recognized as part of larger functional areas. To support such analysis the functions need to be arranged in a logical sequence so that specified operation of the system can be traced in an end-to-end path. In addition to other methodologies supported by Computer-Aided Software Engineering tools, seven approaches commonly used to perform such analyses include:

- functional flow block diagrams;
- data flow diagrams;
- N^2 diagrams;
- time line analyses;
- process decomposition;
- data modeling; and
- information modeling.

Requirements allocation is the further decomposition of system level requirements until a level is reached at which a specific software routine or set of routines can fulfill each of the needed functional and performance requirements. Some straightforward allocation of functional requirements can be made but the procedure may involve the use of supporting analyses, simulations, and prototypes to allocate system level requirements. Examples where additional analysis may be required include the allocation of memory usage or performance timing limits to individual software components.

Another aspect of the documentation of the requirements analysis process is the need to provide traceability analysis for each system requirement allocated to software to one or more lower level software requirements. Such traceability analysis and decision rationale will ensure that the impact of changes to requirements at any time during the software life cycle can be analyzed to determine its impact on the total system. The decision rationale can be reviewed without the need to reconstruct analyses.

An example of a documentation form that can be used to capture requirements identification and allocation is the Requirements Allocation Sheet (RAS) depicted in Figure C-5. The physical format of a RAS is optional and should be tailored to meet the needs of the organization or project involved. In its most usable format, it will have no physical form but will be implemented through an automated support tool. Such a tool allows requirements to be identified and grouped as desired. In addition, such a tool allows individual requirement descriptions to be expanded to sufficient detail to provide criteria for synthesizing and evaluating alternative concepts and approaches. The use of an automated support tool also facilitates problem/change analysis and requirements traceability modifications.



Requirements Allocation Sheet Number:		Computer Software Component Identification		Computer Software Unit Identification	
System Req No.	System Functional, Performance, or Design Requirement	Requirement Allocated to Software	Software Req No.	Data Requirements	
				Input	Output

Figure C-5. Example Form for Allocation of Software Requirements

Allocated requirements are normally stated in terms such as:

- purpose of the function;
- performance parameters;
- interface requirements/constraints;
- design constraints;
- operating environment limits and concepts;
- user considerations; and
- specific design characteristics created by the function such as input, output, control and synchronization, performance values, allowable tolerances, and so forth.

Both qualitative and quantitative requirements resulting from an analysis can be identified on the Requirements Allocation Sheet. Detail supplied on the RAS should be sufficient for direct use as design trade-off criteria. Technical detail should be included to allow portions of one or more RASs to be extracted and, in conjunction with data flows, assembled in the design document as an integrated design element.

C.3.2 Documenting Software Design Activities

In addition to the contents of the Software Design Description, the principal documentation that should be recorded during design activities includes the decisions made during the process and the reasons behind why particular alternatives have been taken and why the others were rejected. During prototype development the principal reasons for decision making includes activities such as comparisons of:

- new approaches versus proven concepts (e.g., use of graphic user interface (GUI) versus menu trees, hardware versus software analog-digital conversions);
- reuse of known software components versus new development;
- commercial-off-the-shelf software versus new development; and
- potential languages or different compilers/assemblers for a given language.

During the design phase the principal types of alternatives that are analyzed are selections such as:

- design approaches/algorithms for individual software components;
- test methods for inclusion in the test documentation; and
- database methods.

In cases where trade studies are conducted as part of the decision-making process, those efforts should be fully documented. The types of information that should be recorded include the following:

- block and data flow diagrams developed for alternatives;
- value estimates for timing (e.g., retrieval, execution, and storage times; external communications times);
- estimates for memory usage (e.g., random access, read only, direct access storage devices, sequential access storage devices);
- estimates of accuracy and tolerance capabilities versus performance differences;
- accuracy/tolerances, timing, formats, and sources for potential external data input devices; and
- trade study element descriptions for each option considered:
  - ⇒ objectives and requirements;
  - ⇒ alternatives considered;
  - ⇒ selection criteria used;
  - ⇒ weights assigned to each of the selection criteria;
  - ⇒ utility functions developed and used; and
  - ⇒ results of sensitivity analyses.

C.3.3 Documenting Software Unit Coding, Unit Codes and Unit Testing Activities

Documentation of software unit coding, unit codes, and unit testing is somewhat informal and is normally contained in the source code listing and/or the Software Development Folder. In the following discussions, the term code unit (also known as a module) is intended to mean the smallest piece of separately compilable code in a software system.

C.3.3.1 Documenting Software Unit Coding Activities

Documentation of software unit coding activities should be sufficient to satisfy the measurements required as part of a software measurements program. This documentation should be kept in the unit's Software Development Folder and might include:

- labor hours expended through initial coding, compiling, and linking the associated code unit;
- labor hours expended in unit testing the code;
- description of:
  - ⇒ the results of, and/or a mark-up of the code listing used for, each attempt to compile or link the code that resulted in detection of an error or defect;
  - ⇒ the root cause of the problem (e.g., programming error, defect in the design, ambiguous software requirement, incorrect interface requirement);
  - ⇒ action taken to correct problem; and
  - ⇒ labor hours required to complete rework and redo the compile or linkage.
- calendar periods during which the above labor hours were expended and what type of activity was performed during each of the periods;
- size of the code unit (e.g., number of non-commented lines of source code) in the final debugged and tested version;
- description of each review performed on the code unit, including at least the following:
  - ⇒ type of review (e.g., peer review, walk through, software inspection);
  - ⇒ date of the review;
  - ⇒ names of persons involved in the review;
  - ⇒ copies of the reports generated prior to the review (e.g., inspection profile), during the review (e.g., inspection defect list, marked-up listing), or after the review (e.g., inspection management report, inspection metrics); and
  - ⇒ summary of the results of the review (e.g., inspection summary).

C.3.3.2 Documenting Software Unit Codes

Documentation of unit codes is normally in the form of "Header Blocks" at the beginning of each code unit listing and the inclusion of comments/remarks internal to the code listing. Header blocks should contain at least the following information relevant to the code unit:

- name of software project under which the code unit is being reused or developed;
- identification of all of the software requirements contained in the requirements documents that are to be fully or partially satisfied within the code unit; and, if any are only being partly satisfied, which parts are to be satisfied within the code unit;
- name of the programmer(s) responsible for development and unit testing of the code unit;
- a brief description of the processing or special algorithms used in the code unit;
- version number of the code unit itself;
- if the same code unit is used more than once within the software system, identification of the other code units that are duplicates;
- a description of any special limitations or error handling characteristics of the code unit; and
- revision history, which contains date of latest revision to the code unit and a brief description and date for each revision's set of changes.

Comment lines within the code unit should be extensive enough to provide an understanding of design considerations and implementation methods for the lines of source code that follow each comment. One good method often used is to:

- describe the detailed design for the code unit in a program design language (PDL) or a high level pseudo code; reference [IEEE990] describes a recommended practice for using Ada as a program design language;
- include the PDL version of the design in the code listing by converting each line of PDL to a comment line and leaving it in the source listing immediately preceding the actual source code that will implement the intent of the line of PDL code;
- include additional comments as necessary to fully describe the function of each code segment;
- if more than one of the requirements from the requirements documents is being completely or partially implemented within a single code unit, then the comments within the code should identify the segments of code that are designed to satisfy each of the requirements and what part of each requirement that segment is to satisfy; this information will provide the traceability linkage between requirements and code elements.

C.3.3.3 Documenting Software Unit Testing Activities

Two standards, references [IEEE829] and [IEEE1008], relate to the subject of software unit testing. The reference [IEEE829] is most pertinent to the subject of documentation for unit testing. It identifies seven test documents and defines their contents. The following two of those seven test documents are identified by reference [IEEE1008] as the absolute minimum for process visibility into unit/module testing:

Test Design Specification which is made up of sections with the following titles:

- Test-Design-Specification Identifier;
- Features to be Tested;
  - ⇒ individual features
  - ⇒ combined features
- Approach Refinements;
  - ⇒ requirements coverage
  - ⇒ design coverage
  - ⇒ domain coverage
  - ⇒ branch coverage
  - ⇒ statement coverage
- Test Identification; and
  - ⇒ test cases
  - ⇒ test procedures
- Feature Pass/Fail Criteria.

Test Summary Report which is made up of sections with the following titles:

- Test Summary Report Identifier;
- Summary;
  - ⇒ number of faults found and corrected
  - ⇒ statement of the final results of testing after the faults had been corrected
  - ⇒ how the unit was exercised for the testing
  - ⇒ identification of all test documents associated with that unit
- Variances;
  - ⇒ special conditions identified during the testing that were valid but were, in fact, enhancements beyond the software requirements associated with that unit
  - ⇒ any additional test cases that were found to be necessary to fully verify satisfaction of the requirements
  - ⇒ notation of the changes to requirements, design, coding, and/or testing documents that are necessitated by the results of the testing activities and the fault corrections that were necessary to attain satisfaction of the requirements
- Comprehensiveness Assessment;
  - ⇒ checklists that were followed and annotated during the testing

  - ⇒ execution trace reports
- Summary of Results;
- Evaluation;
- Summary of Activities; and
  - ⇒ total staffing level for each major testing activity
  - ⇒ total machine time for each major testing activity
  - ⇒ total lapsed time for each major testing activity
- Approvals.

C.3.4 Documenting Software Inspections and Walk Throughs

Software inspections and walk throughs should be documented. Defect data should be documented since it can be analyzed to understand how the current software development processes lead to insertion defects. Cost and benefits of the associated review process should be documented to justify use of the process.

C.3.4.1 Documentation of Software Inspections

Software Inspections are documented using the following four inspection documentation forms:

Inspection Profile Cover: This document is completed by the author of the software product that is to be inspected. It is used as the cover sheet for the inspection package that is distributed to members of the inspection team. The profile includes: project name, date, type of inspection, and size of the package. It also contains a summary of any remaining open issues and indicates whether the inspection is a re-inspection.

Inspection Defect List: During the inspection meeting the recorder completes the inspection defect list. For each defect identified, the recorder enters a location, description, defect severity as major or minor, and defect type (e.g., incomplete, ambiguous, logic, data). The recorder also lists questions that cannot be answered during the inspection meeting. The author will use this form as the basis for correcting the software product and the moderator will use it as a checklist for completing the inspection summary. The author will complete the inspection defect list by noting the root cause (e.g., development activity that was the source for the defect) and rework time for all defects and open issues identified during the inspection meeting.

Inspection Metric Summary: After the inspection meeting, the recorder completes the inspection metric summary. This summary contains a count for each of the defect types, severity, and root cause. This information is most frequently used to evaluate the inspection process. This summary is updated to be consistent with the rework and follow-up for the inspection defect list.

Inspection Summary: After the inspection meeting, the moderator completes this report. The Inspection Management Report, along with the Inspection Summary, is used to provide management with information regarding the inspection. The Inspection Management Report includes information regarding the total amount of time spent in the inspection meeting and in preparation for the meeting. It also summarizes the amount of effort required for the rework phase and the disposition of the inspection meeting.

Documentation from each software inspection should be maintained in the project library for possible analysis across projects to improve the software inspection process.

C.3.4.2 Documentation of Software Walk Throughs

The documentation of software walk throughs is normally less formal than that of software inspections. At a minimum, walk through documentation should contain the following information:

- record of the time spent conducting the walk through and names of the participants;
- copy of the marked-up version of the software product that was reviewed;
- record of the contents of participant inputs; and
- disposition of defects found during the walk through.

Documentation from each software walk through should be maintained in the project library for possible analysis across projects to improve the software walk through process.

C.4 Other Potential Product Documentation

The types of other potential product documentation depend upon the software being developed, who will be using it, and how it will be used. Potential product documentation that could supplement the standard management, requirements, design, and test documentation products are described in the following subsections.

C.4.1 Traceability Documentation for Software and Interface Requirements

The objectives of traceability documentation are to:

- ensure that each of the individual requirements identified in the System Requirements Specification that are allocated to software are incorporated into the Software Requirements Specification;
- ensure that each of the individual requirements in the Software Requirements Specification are incorporated into the Software Design Description;
- ensure that each of the requirements in the Software Design Description are incorporated into the appropriate source code listing;
- ensure that each of the individual requirements identified in the Software Requirements Specification are addressed in the Software Test Descriptions and Test Procedures;
- provide a bi-directional trace of requirements between the software specification, software design, code listings, and software test documents; and
- facilitate evaluation of the impact of a proposed change on all levels of documentation other than that document for which the need for change was first recognized.

Traceability can be documented through either manual or automated methods. Commercial-off-the-shelf automated traceability tools are available and may be required for large-scale software development efforts. However, for medium-sized or smaller software efforts a personal-computer-based database management package may be used. Manual methods may be applicable for comparatively small efforts.

The more complex the system being developed or the larger the number of organizations or personnel involved in the development, the greater the need for early implementation of accurate and complete documentation that traces requirements from their source to their final tested, integrated, and installed forms.

C.4.2 Software User, Installation, and Maintenance Documents

Reference [IEEE1063] addresses the subject of user manuals and also suggests a style manual might be useful to complement the guidance provided in this standard. This IEEE standard does not break out user documents into specific types such as user, installation, computer operator, and maintenance manuals. Instead, it addresses issues such as: document sets (number of volumes) and usage modes (e.g., instructional or reference). This standard is a good reference for determining the subject matter for user documents but does not provide a template that can be followed to create a document for a specific audience. The following subsections provide further guidelines for three audiences: users, installers and operators, and maintainers.

C.4.2.1 Software User Manual

The Software User Manual should provide user personnel with instructions sufficient to execute the software. It may be supplemented with on-line tutorials or on-line help capabilities. This document should contain the following information:

- an overview of the system and the role of the software within that system;
- an overview of the user manual;

- a listing of reference documents that may be of value to the user
- the body of the document describes execution procedures and addresses the following:
  - ⇒ Initialization - initial loading of the software on the computer on which it will be executed;
  - ⇒ User Inputs - description of the user interface and input of user commands and data;
  - ⇒ System Inputs - description of the inputs from the system to the software that may occur while the software is in use and that may affect the software's interface with the user (e.g., inputs from a remote sensor); formats, frequencies of input, allowable ranges, and units of measure should be included where applicable.;
  - ⇒ Termination - description of how to terminate the software operation and how the user can determine whether normal termination has occurred;
  - ⇒ Restart, Backup, and Recovery - description of the procedures to be used to restart, backup, and recover the software;
  - ⇒ Outputs - Description of expected outputs of the software, including error messages
- a list of error messages: all error messages that can be output by the software, describe the meaning of each error message, and define the action to be taken when each message appears;
- notes and explanations: general information that aids in understanding of the users manual (e.g., background information) and a listing of all acronyms and abbreviations with their meanings within the context of the user manual; and
- appendices: used to provide information that is separated for convenience in document maintenance (e.g., charts, classified data) and for coverage of details not appropriate for the body of the document.

C.4.2.2 Software Installation and Transition Documentation

The Department of Defense separates such documentation into several documents such as a Software Installation Plan, Software Transition Plan, and Software Product Specification. See references [MIL498] and [IEEE1498]. The following is a list of potential information contained in such documentation:

- installation instructions for the software and interfacing/integrating the software to other software components, hardware, and communications links;
- software support resources - identifies and describes the components of the software engineering and test environments required to support the deliverable software; identifies the relationships of the components and discusses the items required to modify the software, perform testing, and copy the software for further distribution; further information may include the following:
  - ⇒ software - identifies and describes all of the deliverable software and associated documentation;
  - ⇒ support software - identifies and describes the automated tools needed to support and maintain the deliverable software;
  - ⇒ hardware - describes the operating environment necessary to support the delivered software;
  - ⇒ facilities - describes the facilities required to support the deliverable software and identifies the purpose of each;
  - ⇒ personnel - identifies the personnel required to support the deliverable software, including the types of skills, number of personnel, security clearances, and skill levels; and
  - ⇒ other resources - identifies any other resources that may be required to support the deliverable software.
- operations - describes the operations necessary to support the deliverable software; further information may include the following:
  - ⇒ software modification - describes the procedures necessary to modify deliverable operational and support software and the procedures for accommodating revisions to commercially available and reusable computer resources;
  - ⇒ software integration and testing - describes the procedures necessary to integrate and fully test all software modifications; includes procedures to identify portions of changes that need further testing in the operational environment and to establish guidelines for determining, developing, and verifying the amount of testing required;

- ⇒ software generation - provides the information necessary to facilitate compilations or assemblies of the deliverable software and instructions for loading, executing, or recording the results of compilations or assemblies;
- ⇒ simulation - details the hardware, software, and procedures necessary for any required simulation; and
- ⇒ emulation - details the hardware, software, and procedures necessary for any required emulation.

- training - describes the developing organization's plans for training of personnel to manage, implement, use, and support the deliverable software; the schedule, duration, and location for all training is included;
- transition planning - describes the developing organization's plans for transitioning the deliverable software to the user's operating environment; further information might include:
  - ⇒ resources for both the developer and the user that will be necessary to carry out the transition;
  - ⇒ schedules and milestones for the transition efforts; and
  - ⇒ procedures for installation and checkout of the deliverable software.

C.4.2.3 Software Maintenance Document

The Software Maintenance Document provides information on how to support the software during its operational life. Sometimes this information is contained in a Software Programmer's Manual. This document should contain the following information as applicable:

- system overview - states the purpose of the software system and the computer systems to which this software is applicable;
- software programming environment - describes the operating system(s) of the host and target computer(s) and other software involved in loading, compiling, and executing the deliverable software; if the information described below is provided in a commercially available document, that document should be referenced by title, number, and applicable paragraphs:
  - ⇒ equipment configuration - description of the components and configuration of the host and target computer systems, and communications linkages and networks;
  - ⇒ operational information - description of the operating characteristics, capabilities, and limitations of the host and target computer systems in terms such as:
    - machine cycle times;
    - word lengths;
    - memory capacity and characteristics;
    - instruction set characteristics;
    - interrupt capabilities;
    - modes of operation (e.g., batch, interactive, real-time, privileged, non-privileged);
    - operational registers;
    - error indicators;
    - input/output characteristics; and
    - special features.
  - ⇒ compilations, assemblies, and linkages:
    - description of the equipment (e.g., tapes, disks diskettes, other peripherals) necessary to perform compilations and assemblies on the computer system;
    - identify, as applicable, by name and version number the editor, linker, link-editor, compiler, assembler, cross-compilers, and other utilities used, and reference appropriate manuals describing their use; and
    - highlight any special flags or instructions necessary for loading, executing, or recording the results of compilations and assemblies.
- programming information - description of the programming information relative to the host and target computers; if the information described below is provided in a commercially available document, that document should be referenced by title, number, and applicable paragraphs:

⇒ programming features - descriptions of the computer's instruction set architecture in terms such as:
  - data representation (e.g., byte, word, integer, floating-point)
  - instruction formats and addressing modes;
  - special registers and words (e.g., stack pointer, program counter, processor status word);
  - control instructions (e.g., branch, jump, subroutine and call instructions, privileged information, and the operating modes);
  - subroutines and procedures (e.g., non-reentrant, reentrant, macro code routines, argument lists, parameter passing conventions);
  - interrupt processing;
  - timers and clocks; and
  - memory protection features.

⇒ programming instructions - description, as necessary, of each instruction in the computer's instruction set architecture in terms such as:
  - use;
  - syntax;
  - condition codes set;
  - execution time;
  - machine-code format; and
  - mnemonic conventions.

⇒ input and output control programming - description of the input and output control programming of the computer system, including, as applicable, descriptions of:
  - initial loading and verification of computer memory;
  - serial and parallel data channels;
  - discrete inputs and outputs;
  - interface components; and
  - device numbers, operational codes, and memory locations for peripheral equipment.

⇒ additional or special techniques - descriptions of additional or special programming techniques with the computer system (e.g., concise description of the micro program control section showing how the user instruction set is implemented via microcode);

⇒ programming examples - provide examples that demonstrate the programming features described above for all categories of instructions on the specific computer system; and

⇒ error detection and diagnostic features - description of effort detection and diagnostic features associated with the computer system, including conditions codes, overflow, and addressing exception interrupts, and input and output errors status indicators.

C.4.3 Software Version Description Document

A Software Version Description Document identifies and describes a version of deliverable software. It can be used to document an initial release version or an updated release version. It can also be used by the developer to track and control versions of software released to different operational environments and users. The version description information contained in the body of this document includes the following:

- inventory of materials released - listing of all physical media (e.g., source code listings, tapes, diskettes) and associated documentation that make up the new version; in addition, an identification of all operational and support documents that are not part of the delivered package but that are required to operate, load, or regenerate the delivered software;
- inventory of the software components released - identification of all computer software elements that are part of the delivered software, normally in the same sequence as is used to organize the source code listing for delivery;
- descriptions of the changes from previous release -list of each of the changes from the previous version; include with each change description a cross reference to the affected software specifications;

- adaptation data - for the initial release of the software identify or reference all unique-to-site data contained in the items being delivered; for subsequent versions, provide the information necessary to identify changes to the adaptation data;
- interface compatibilities - indicate other system and configuration items affected by the changes incorporated in this version; not applicable to the initial version release;
- bibliography of reference documents - for the initial version, list all documents pertinent to the deliverable software; for subsequent versions, identify all changes to the listed documents;
- installation instructions - provide the instructions, either directly or by reference, necessary to install the version delivered;
- possible problems and known defects - identify all possible or known defects in the delivered version and any steps being taken to resolve those problems.

C.4.4 Firmware Support Manual

The Firmware Support Manual provides information necessary to load software or data into firmware components of a system. It is equally applicable to read only memory, programmable read only memory, erasable programmable read only memory, and other firmware devices. The principal components of this document include:

- firmware device information - describes the firmware devices and references commercially available documents for the following types of information:
  - ⇒ device description(s);
  - ⇒ installation and repair procedures;
  - ⇒ security; and
  - ⇒ limitations.
- programming equipment and procedures - descriptions of the programming hardware, programming software, and loading procedures necessary for programming the firmware devices; and
- vendor information - information supplied by the vendors for firmware device, programming hardware, and programming software or references to the appropriate commercially available documents related to those products.

C.5 Supporting Documentation

The following paragraphs discuss various other supporting documentation that may be useful for a software project. This supporting documentation includes meeting minutes, lessons learned, briefings, project reviews.

C.5.1 Technical Interchange Meeting Minutes

Technical interchange meetings take place with various combinations of the end-use customer, system development team, and different elements of the development team. Minutes of those meetings need to be recorded to document the decisions that were reached and agreed-to by the participant parties and to document the future actions that are to be taken as a result of the meeting. The contents of meeting minutes should include at least the following items:

- date of the meeting;
- names of persons attending the meeting;
- description of business from previous meeting that was discussed;
- description of each of the new business subjects discussed;
- description of the outcome of each old and new business item discussed; and
- list of action items that includes at least the following:
  - ⇒ description of the action required and the results expected;
  - ⇒ name(s) of the person(s)/organization(s) responsible for completing the required action;
  - ⇒ specification of the date/time by which the action needs to be completed; and

⇒ identification of the recipients to whom the completed action results will be distributed.

C.5.2 Lessons Learned

The objectives of recording lessons learned are to:

- document the successes and failures in such a way as to make the results of past efforts available for use in improving the processes used in software development and maintenance; and
- provide a record of the overall improvements in software development activities.

In general, the documentation of lessons learned has been in a free-form format. This approach results in a collection of information that is difficult to use later as reference material. Therefore, a standardized format that will facilitate cross referencing within the project and across multiple projects should be developed. One approach might be to develop a form that can be used to record individual lessons. Another might be to use a personal computer- or workstation-based database management system. The following is a list of potential paragraph headings for use in recording lessons learned:

- success/problem - the entry here must be a single word "Success" or "Problem";
- key words - one or more key words must be included to facilitate grouping lessons learned; examples of key words might include: personnel, software inspection, root cause, defect, failure;
- lesson learned - a free-form paragraph on the lesson learned that should address/include one or more of the words from the key words list;
- stage - the applicable development stage or phase in which the lesson was learned; should be filled in by selecting from a pre-defined list of stages/phases;
- description/action - a free-form paragraph with a detailed description of the action taken as a result of the lesson learned that should address/include one or more of the words from the key words list;
- recommendation - a free-form paragraph on recommendations (such as improvements) that, if accomplished, would decrease similar problems or make similar successes more likely in future projects; and
- degree of importance - typically a single word such as: high, medium, low, major, minor.

C.5.3 Briefing Documents

Briefings will often be required as part of a software development project. Individuals presenting such briefings will normally develop briefing materials to support their presentations. The software project manager should collect a copy of such support materials for inclusion in the archives for the project. Such an archival document will be of more use when it has been annotated with all pertinent comments that were brought up by either the presenter or others attending the meeting. In addition, the briefing documents should include the names and organizations of those who attended the briefing and any action items that resulted. Action items should be documented as follows:

- description of the action required and the results expected;
- name(s) of the person(s)/organization(s) responsible for completing the required action;
- specification of the date/time by which the action needs to be completed; and
- identification of the recipients to whom the completed action results will be distributed.

C.5.4 Project Review Documents

Formal project reviews such as Preliminary Design Reviews, Critical Design Reviews, and Software Test Readiness Reviews may also take place during a software development effort. Typically, such formal reviews will include one or more briefing-like presentations and a number of discussions. As a result, documentation of such formal reviews will normally be a combination of the information described above for Technical Interchange Meetings and Briefing Documents.

Appendix D: Document Set Selection for Example Projects

This appendix provides examples of how to select a project document set using the risk scoring and selection guidelines outlined in Chapter 3, Figures 3-2 and 3-3. These examples represent a broad spectrum of software development projects likely to occur within Sandia National Laboratories. A summary of the example projects is provided in the matrix below.

PROJECT	DESCRIPTION
Project 1	A War Reserve (WR) software development project that provides a secure recode capability to all nuclear weapons
Project 2	A non-WR project related to an internal Sandia information system software development project
Project 3	A small two-person research project
Project 4	A software support project requiring on-going support for a large software program developed over the last 20 years
Project 5	A project requiring purchase of commercial software to support an information system architecture
Project 6	A customer-specific Work For Others (WFO) program with software developed in-house until technology can be supported through the technology transfer program
Project 7	A WFO program developed for an external customer involving a command and control system that will be a totally new development

D.1 Project 1: WR Development

Description: I have a WR software development project. Use control software is being developed for a state of the art system that will provide a secure recode capability to all nuclear weapons (US and Allies). Sandia experience in this area is the best possible. Previous project software will be reused but at least 30K new non-commented source lines of code are expected to be developed. What documentation set is appropriate for this project?

Risk Factor	Type	Score	Comments
1. Process Risk			DOE project; ultimate customer is DoD; Engineering Procedure EP401045 applies
acquisition			
development	development	100	
use			
support			
2. Size Risk			Approximately 40K NCSLOC can be reused, but it is estimated that 30K NCSLOC of newly developed 'C' language code will be required.
low: < 1K NCSLOC			
medium: 1K-20K NCSLOC			
high: 20K-100K NCSLOC	high	100	
xhigh: > 100K NCSLOC			
3. Cost Risk			Estimate is for 155 person months labor over a period of 17 calendar months. Total cost will be approximately $2.2M. Due to the level of security required, an additional factor of 1.5 pushes the estimated cost to $3.3M.
low: < $25K			
medium: $25K - $50K			
high: $50K - $100K			
xhigh: $100K - $1,000K			
xxhigh: > $1,000K	xxhigh	500	
4. Customer Risk			This WR project is for an external customer and is key to Sandia's future business base in the area of nuclear weapons.
none			
internal			
WR or non-WR			
Research or non-Research			
external	external	100	
WR or non-WR	WR	100	
WFO or non-WFO			
Research or non-Research			
5. Life Cycle Risk			This is a production development effort with products that are expected to have a fairly long lifetime (certainly more than five years and up to twenty years). All life cycle activities will be performed, including any future software support for the products.
Prototype or Production	production	100	
Expected Lifetime	long	100	
Activities of Concern	all	170	
6. Support Environment Risk			The development and support environment will be a LAN of SUN workstations using a UNIX-based system. The target environment will be a combination of COTS PCs and special-purpose hardware for weapons interface designed by Sandia. A Motorola processor will be in the special-purpose hardware.
Platform	LAN workstation	50	
Automated Tool Support	intermediate	50	
Personnel Skill Mix	medium experience	20	

7. Historical Experience Risk Experience Level Personnel Stability	 high high	 10 10	Sandia is the only experienced software developer in this area. All personnel assigned to the effort are expected to be available during the majority of the software's life cycle.
8. SNL Risk Operational Reliability Customer Satisfaction	 high high	 500 200	This project is an external WR project that is key to Sandia's future business base in the area of nuclear weapons. Operational Reliability of the software is expected to be very high. Customer satisfaction is critical to Sandia's future in this area.
9. Other Risk Areas Security Safety Use Control	 high high high	 100 100 100	Need Software Security/Safety Plan. High level of security, safety, and use control is required. Refer to project Classification Guide and Use Control Theme documentation for more information.
Totals Risk Range: xLow < 500: Minimal Set Low 500 - 999: Basic Set Medium 999 - 1499: Small Set High 1500 - 1999: Medium Set xHigh Risk > = 2000: Large Set	xhigh risk	2410	Select Large Document Set: SDP, SQAP, SCMP, SRS, SDD, CODE, SSTP, SSMP, SVVP, SDF Plus: SSP since safety/security is an issue SUG since user interface is critical QP/QER since project is WR Use Control Theme (reference) Classification Guide (reference)

D.2 Project 2: Non-WR Information System Development

Description: I have an internal Sandia information system software development project. The customer(s) are all internal Sandia personnel who use the personnel information system. The software to be developed will enable the financial and project management information databases to be linked with the personnel information system so that project managers across all Sandia organizations can maintain and update on-line project data for personnel assigned to their projects. A project manager interface for data entry and reporting will be part of the project. It is expected that only a small amount of software will need to be developed using modern fourth generation database query and user interface language capabilities. The adjusted function points (inputs, outputs, inquiries, files, interfaces plus adjustment factor) to be developed have been estimated at 816. Documentation exists for the current systems and their existing software (the main system software has been commercially purchased, but a significant amount of user software has been developed internally). What documentation set is appropriate for my project?

Risk Factor	Type	Score	Comments
1. Process Risk acquisition development use support	development	100	Internal SNL project; direct customers are the personnel systems group, but indirect customers are all SNL project managers.
2. Size Risk low: < 1K NCSLOC medium: 1K-20K NCSLOC high: 20K-100K NCSLOC xhigh: > 100K NCSLOC	medium	50	Approximately 816 function points of newly developed Fourth Generation Language code will be required. (Falls into the medium size risk range.)
3. Cost Risk low: < $25K medium: $25K - $50K high: $50K - $100K xhigh: $100K - $1,000K xxhigh: > $1,000K	xhigh	200	Estimate is for 35 person months labor over a period of 10 calendar months. Total cost will be approximately $300K.
4. Customer Risk none internal WR or non-WR Research or non-Research external WR or non-WR WFO or non-WFO Research or non-Research	internal non-WR	50 10	This non-WR project is for an internal customer and is key to Sandia's information architecture and capability to support the management of future projects in accordance with government and industry standards.
5. Life Cycle Risk Prototype or Production Expected Lifetime Activities of Concern	production long all	100 100 170	This is a production development effort with products that are expected to have a fairly long lifetime (certainly more than five years and up to ten years). All life cycle activities will be performed, including any future software support for the products.

6. Support Environment Risk			The development and support environment will be a LAN of DEC workstations linked to the production VAX machine containing the operational system and networked to Sandia's internal secure network. The target environment will be a combination of COTS PCs linked to the internal secure network and layered with appropriate system support software and the project management application support software.
Platform	LAN workstation	50	
Automated Tool Support	integrated CASE	10	
Personnel Skill Mix	experienced	10	
7. Historical Experience Risk			Sandia is very experienced in developing applications software in this area. Some junior level personnel have been leaving the organization to broaden career opportunities. This is expected to continue over the life of the effort.
Experience Level	high	10	
Personnel Stability	medium	20	
8. SNL Risk			Operational Reliability of the software and Customer Satisfaction are expected to be good, but are not critical to human life or expensive property. Workarounds typically exist for any defects.
Operational Reliability	low	10	
Customer Satisfaction	medium	100	
9. Other Risk Areas None			
Totals Risk Range: xLow < 500: Minimal Set Low 500 - 999: Basic Set Medium 999 - 1499: Small Set High 1500 - 1999: Medium Set xHigh Risk > = 2000: Large Set	low risk Note: since the risk score is so close to the medium risk boundary, it may be useful to evaluate in more detail whether some of the Medium Document Set might be appropriate.	990	Select Small Document Set: SDP. SRS, CODE, SSTP, SDF Plus: SUG since user interface is important; and any applicable vendor documents for the project manager stations Note: it is expected that existing software configuration management will be used so current plans and tools will be sufficient for the new S/W.

D.3 Project 3: Small Research Development

Description: I have a small two-person research project that requires the development of a small amount of software to support some of the research. The software is primarily to support analysis and is not intended to be used by anyone else, internally or externally. The software will be primarily applied mathematics code. The customer(s) are just the two principal investigators, although the research results are expected to be of interest to our entire department that specializes in robotics research. If the research results are as positive as we expect, then future software will need to be developed to support a larger production version of the products that will be prototyped. What documentation set is appropriate for my project?

Risk Factor	Type	Score	Comments
1. Process Risk acquisition development use support	development	100	Internal SNL project; only direct customers are the two principle investigators.
2. Size Risk low: < 1K NCSLOC medium: 1K-20K NCSLOC high: 20K-100K NCSLOC xhigh: > 100K NCSLOC	low	10	Less than 1,000 NCSLOC of Small Talk software source code is expected to be developed.
3. Cost Risk low: < $25K medium: $25K - $50K high: $50K - $100K xhigh: $100K - $1,000K xxhigh: > $1,000K	low	10	Estimate is for 3 person months labor over a period of 12 calendar months. Total cost will be approximately $20K.
4. Customer Risk none internal WR or non-WR Research or non-Research external WR or non-WR WFO or non-WFO Research or non-Research	none	10	This project is an internal SNL project with essentially no customers.
5. Life Cycle Risk Prototype or Production Expected Lifetime Activities of Concern	prototype short concept definition, implementation, installation, operation	10 10 50	This is a prototype development effort with products that are expected to have a very short lifetime (certainly < 1 year). Requirements are fairly well understood, but emphasis will be on accuracy of implementation. Some concept is needed, but very little design or formal test is needed since the algorithms are all understood. Test will be mostly at the system level.

6. Support Environment Risk Platform Automated Tool Support Personnel Skill Mix	PC workstation integrated CASE experienced	10 10 10	The development and support environment will be a PC workstation that will be networked to the laboratory robotics system being developed. Personnel are experienced with the current Small Talk environment and have developed interfaces and prototypes before in this environment.
7. Historical Experience Risk Experience Level Personnel Stability	medium high	20 10	Principal investigators are very experienced in developing applications in this area, although they do not consider themselves to be software developers. No personnel turnover is anticipated.
8. SNL Risk Operational Reliability Customer Satisfaction	low low	10 10	Operational Reliability of the software and Customer Satisfaction are expected to be very low. Workarounds will exist for any defects.
9. Other Risk Areas None			
Totals Risk Range: xLow < 500: Minimal Set Low 500 - 999: Basic Set Medium 999 - 1499: Small Set High 1500 - 1999: Medium Set xHigh Risk > = 2000: Large Set	xlow risk	280	Select Minimal Document Set: CODE, SDF Note: The SDF should contain a brief description of the project plan, and a one-two page statement of the requirements for the software to be developed. The algorithms to be implemented should be included in a separate section of the SDF. A lessons learned section in the SDF would be appropriate.

D.4 Project 4: Support of Existing Simulation Code Software

Description: I have a software support project that requires the on-going support of approximately 300K non-commented source lines (total of 400K with comments) of FORTRAN simulation and user command language code. The code simulates the flow of various gases and fluids through different geological strata and is being used by several environmental support projects to make critical decisions that will affect future Sandia and other industry support projects. The code has been developed over a period of over 20 years, and support activities include correcting any defects found in the code, enhancing the code to provide new capabilities to support newly defined gases and/or fluids & geological strata, operating and using the code in a project analysis support role, and writing professional papers to interface with other researchers in this area. The code algorithms are very well documented in a mathematical code design specification, which matches fairly well the software implementation, but there is no software design documentation. Although there is no formal software requirements specification, there is a concept document that describes in detail the intended applications for the code, and this document is kept reasonably current with lessons learned on any existing projects. The interface to the code for user parameter inputs and analysis outputs is documented in a user interface document that is constantly being updated to keep it current with changes to the code. Although there are several persons who use the code in the department, there are only two of us that really know the algorithms and their implementation in the software, and we are the only two who provide actual software change support. I've had some training in modern software engineering techniques but consider myself primarily a hydrology physics scientist who develops software simulations to support my primary area of interest. What documentation set is appropriate for my project?

Risk Factor	Type	Score	Comments
1. Process Risk			Internal SNL project; direct external customers on both WR and non-WR work for others projects
acquisition			
development			
use			
support	support	50	
2. Size Risk			Less than 1,000 NCSLOC of FORTRAN developed/changed per year, but software source code to be supported is very large for only a one/two person staff.
low: < 1K NCSLOC			
medium: 1K-20K NCSLOC	medium	50	
high: 20K-100K NCSLOC			
xhigh: > 100K NCSLOC			
3. Cost Risk			Estimate is for 1.5 persons labor each year. Total cost will be approximately $200K.
low: < $25K			
medium: $25K - $50K			
high: $50K - $100K			
xhigh: $100K - $1,000K	xhigh	200	
xxhigh: > $1,000K			
4. Customer Risk			This research project is for both internal SNL and external customers. Probably the more important aspect to consider here is the external customer that is primarily WR related.
none			
internal	internal		
WR or non-WR			
Research or non-Research	research	10	
external	external	100	
WR or non-WR	WR	100	
WFO or non-WFO	WFO	100	
Research or non-Research			

5. Life Cycle Risk			This is a production support effort with products that are expected to have a very long lifetime (certainly > 5 years). Concept is well understood, requirements are partially documented, very little software design exists, a common regression test suite is used for testing, but no test case/results document exists - just previous executions. Code has some comments that have been added as changes were made to the code.
Prototype or Production	production	100	
Expected Lifetime	long	100	
Activities of Concern	installation, operation, support	60	
6. Support Environment Risk			The support environment is a SUN workstation linked into the Sandia supercomputer network (code operates on CRAY level machines). Personnel are experienced with the current environment but only a few CASE tools are used, such as source code management, compilers, some graphic support, etc.
Platform	LAN workstation	50	
Automated Tool Support	intermediate set	50	
Personnel Skill Mix	experienced	10	
7. Historical Experience Risk			Principal investigators are very experienced in developing applications in this area, although they do not consider themselves to be software developers. Unfortunately, the two individuals who know the most about the code are planning to retire within the year. They will be replaced by much less experienced personnel.
Experience Level	medium	20	
Personnel Stability	low	50	
8. SNL Risk			Operational Reliability of the software and Customer Satisfaction are expected to be very good. Workarounds generally exist for any defects, but the accuracy of results is very important for the customer decision-making process.
Operational Reliability	low	10	
Customer Satisfaction	medium	100	
9. Other Risk Areas			
Lack of Documentation	medium	50	Support documentation is minimal.
Lack of Backup Personnel	medium	50	Key personnel can not be easily replaced.
Sandia Proprietary Software	medium	50	Software requires proprietary controls.

Totals Risk Range: xLow < 500: Minimal Set Low 500 - 999: Basic Set Medium 999 - 1499: Small Set High 1500 - 1999: Medium Set xHigh Risk > = 2000: Large Set	medium risk	1310	Select Small Document Set: SDP, SRS, CODE, SSTP, SDF Plus: SUG: since user interface is critical SDF should contain a brief description of each project plan and a one-two page statement of the requirements the software is supporting. The implemented algorithms documentation should be included in a separate section of the SDF. The regression test should be documented with a results section in the SDF. A lessons learned section in the SDF would be appropriate for each project. Note: To reduce some of the risks, it would be advisable to reverse engineer the software code through use of some modern tools to create data dictionary and flow diagram design information. It would also be advisable to re-organize the current algorithm specification documentation into a good SRS.

D.5 Project 5: Purchased Information System Software

Description: I have a project that requires the purchase of commercial software that will be used to support the applications of my department with involvement of other internal Sandia departments. The software is primarily to support the new information system architecture that will be developed within Sandia over the next decade. The commercial vendor will be the support agent. It is expected that this software will cost in excess of $1M. What documentation set is appropriate for my project?

Risk Factor	Type	Score	Comments
1. Process Risk			Internal SNL project; direct customers are internal SNL departments that use the primary SNL information systems.
acquisition	acquisition	10	
development			
use	use	10	
support			
2. Size Risk			Although the software is very large, Sandia will have no development or support responsibility, so the size is not really a direct factor.
low: < 1K NCSLOC	low	10	
medium: 1K-20K NCSLOC			
high: 20K-100K NCSLOC			
xhigh: > 100K NCSLOC			
3. Cost Risk			The cost of commercial software of this magnitude requires DOE notification and certain internal Sandia processing based on DOE Orders and internal Sandia policies.
low: < $25K			
medium: $25K - $50K			
high: $50K - $100K			
xhigh: $100K - $1,000K			
xxhigh: > $1,000K	xxhigh	500	
4. Customer Risk			This non-WR project is for internal SNL customers. It could be argued that the use of the information systems for WR work could imply some WR impact. And, since the results of information systems supports external customer use, it could also be argued that some customers are external.
none			
internal	internal	50	
WR or non-WR	non-WR	10	
Research or non-Research			
external			
WR or non-WR			
WFO or non-WFO			
Research or non-Research			
5. Life Cycle Risk			This is a production acquisition effort with products that are expected to have a very long lifetime (certainly > 5 years). It will be necessary to work on the operational concept analysis with the proposed vendor to ensure the vendor's product will support Sandia's intended applications (either directly or through product modifications). It will also be necessary to enforce certain vendor documentation delivery to support Sandia's installation and operational use of the software.
Prototype or Production	production	100	
Expected Lifetime	long	100	
Activities of Concern	concept, installation, operation	30	

6. Support Environment Risk			The vendor development and support environment will be a LAN workstation that will be networked to the laboratory so that updates and direct vendor support can be provided to the Sandia on-site staff. Sandia personnel are somewhat experienced with the vendor's product and will receive extensive training on the installation/update and operation procedures.
Platform	LAN workstation	10	
Automated Tool Support	integrated CASE	10	
Personnel Skill Mix	medium experience	20	
7. Historical Experience Risk			Vendor is well-established and capable of providing extended support to the commercial product. Sandia has good experience with information systems and is committed to providing a state-of-the-art information architecture (communications, hardware platform, and software) for the laboratories and its interface to external customers. There is not expected to be any unusual change is personnel stability over the expected lifetime of this project.
Experience Level	medium	20	
Personnel Stability	medium	20	
8. SNL Risk			Operational Reliability of the software and Customer Satisfaction are expected to be high. Workarounds will exist for any defects, but the software is key to future evolution of the information architecture and must provide high reliability and internal Sandia customer satisfaction.
Operational Reliability	medium	100	
Customer Satisfaction	high	200	
9. Other Risk Areas			
Vendor Support	medium	50	Vendor capabilities should be documented through a vendor survey, to provide some assurance that the risks in these two areas are not too high.
Vendor S/W Processes	high	100	

Totals Risk Range: xLow < 500: Minimal Set Low 500 - 999: Basic Set Medium 999 - 1499: Small Set High 1500 - 1999: Medium Set xHigh Risk > = 2000: Large Set	medium risk	1350	Select (Special) Small Document Set: Software Products: CODE, SUG, SDF variation (see below) Plus: Software Acquisition Plan System Specification Requirements Software Vendor Survey Installation/Operation Manuals Software Support Folder (SSF) (variation on SDF) Note: This project requires that Sandia develop an acquisition plan as the customer of the software supplier. This plan should account for the development of a system/software specification requirements document that can be used to determine whether vendor's software will satisfy Sandia's operational concept for its use. A software vendor survey should be used to verify that the vendor's software processes and products satisfy the medium to high risk category. The SSF should contain: a brief description of the project plan; a concept of operation description for the software; and references to the system specification requirements document, software vendor survey document, and vendor survey results.

D.6 Project 6: Customer Specific WFO - SEMATECH

Description: I have a Work For Others contract with SEMATECH, the semiconductor equipment consortium, that has a specific task to develop software for a Reliability Analysis Program (RAP). This program will be used by SEMATECH consortium members to build reliability block diagrams and conduct limited reliability and fault tree analyses for member fabrication laboratories and for member semiconductor equipment suppliers. The software will be developed in-house and supported until the technology can be transferred to an outside support industry through a technology transfer program. It is estimated that most of the software will be new although the algorithms and application technology is well understood and used by Sandia. The development and target platform will be a standalone PC. It is estimated that approximately 15K non-commented source lines of code will be developed in the "C" programming language. What documentation set is appropriate for my project?

Risk Factor	Type	Score	Comments
1. Process Risk acquisition development use support	 development use support	 100 10 50	Internal SNL WFO project; direct customer is SEMATECH; indirect customers are the semiconductor companies who will use the product as well as SNL developers who will support the use of the product.
2. Size Risk low: < 1K NCSLOC medium: 1K-20K NCSLOC high: 20K-100K NCSLOC xhigh: > 100K NCSLOC	 medium	 50	Less than 20,000 NCSLOC of "C" language code will be required.
3. Cost Risk low: < $25K medium: $25K - $50K high: $50K - $100K xhigh: $100K - $1,000K xxhigh: > $1,000K	 xhigh	 200	Estimate is for 60 person months labor over a period of 12 calendar months for the development, and approximately 0.5 persons per year for a two year support period. Total cost will be approximately $700K.
4. Customer Risk none internal WR or non-WR Research or non-Research external WR or non-WR WFO or non-WFO Research or non-Research	 external WFO	 100 100	This WFO project is primarily for external customers.
5. Life Cycle Risk Prototype or Production Expected Lifetime Activities of Concern	 production medium all	 100 50 170	This is a production development effort with products that are expected to have a medium lifetime. Full life cycle of activities is required, including support.
6. Support Environment Risk Platform Automated Tool Support Personnel Skill Mix	 PC workstation intermediate set medium	 10 50 20	The development and support environment will be a standalone PC workstation. Some CASE tools will be available. Personnel are experienced with reliability applications and have some software development experience.
7. Historical Experience Risk Experience Level Personnel Stability	 low high	 50 10	Principal investigators are experienced in reliability applications, but not in developing software for this area.

8. SNL Risk			Operational Reliability of the software is not critical, workarounds will exist for any defects. Customer Satisfaction should be reasonably good or the tool will not be used.
Operational Reliability	low	10	
Customer Satisfaction	medium	100	
9. Other Risk Areas			
Sandia Proprietary Software	medium	50	Software requires proprietary controls.
SEMATECH Proprietary Data	medium	50	SEMATECH company data may require proprietary controls.
Totals Risk Range: xLow < 500: Minimal Set Low 500 - 999: Basic Set Medium 999 - 1499: Small Set High 1500 - 1999: Medium Set xHigh Risk > = 2000: Large Set	medium risk	1280	Select Small Document Set: SDP, SRS, CODE, SSTP, SDF Plus: SUG: since user interface is critical SSMP: since continued support is required Note: The algorithms to be implemented should be included in a separate section of the SDF. A lessons learned section in the SDF would be appropriate.

D.7 Project 7: Internally Developed for External Customer WFO - Ada

Description: I have a software development project for an external DoD customer on a Work For Others contract. The software is a command and control application that will be totally new development. The estimate is for approximately 250K non-commented source lines of code developed in the Ada programming language using an Object-Oriented development methodology with a fully integrated CASE tool set. Personnel are experienced with command and control applications, but only recently became trained in the use of the Ada language and Object-Oriented methodologies. What documentation set is appropriate for this project?

Risk Factor	Type	Score	Comments
1. Process Risk			DoD project; Engineering Procedure EP401045 might also apply. MIL-STD-498 applies. Customer Data Item Descriptions apply per contract.
acquisition			
development	development	100	
use			
support			
2. Size Risk			About 250,000 NCSLOC of newly developed Ada language code will be required.
low: < 1K NCSLOC			
medium: 1K-20K NCSLOC			
high: 20K-100K NCSLOC			
xhigh: > 100K NCSLOC	xhigh	200	
3. Cost Risk			Estimate is for 2000 person months labor over a period of 72 calendar months. Total cost will be approximately $22.2M.
low: < $25K			
medium: $25K - $50K			
high: $50K - $100K			
xhigh: $100K - $1,000K			
xxhigh: > $1,000K	xxhigh	500	
4. Customer Risk			This WFO project is for an external customer and is key to Sandia's future business base in the area of non-nuclear DoD weapons business.
none			
internal			
WR or non-WR			
Research or non-Research			
external	external	100	
WR or non-WR			
WFO or non-WFO	WFO	100	
Research or non-Research			
5. Life Cycle Risk			This is a production development effort with products that are expected to have a fairly long lifetime (certainly more than five years and up to twenty years). All life cycle activities will be performed, including any future software support for the products.
Prototype or Production	production	100	
Expected Lifetime	long	100	
Activities of Concern	all	170	
6. Support Environment Risk			The development and support environment will be a LAN of SUN workstations using a UNIX-based system. The target environment will be a combination of LAN Workstations and existing special-purpose hardware for weapons communications interface.
Platform	LAN workstation	50	
Automated Tool Support	integrated CASE	10	
Personnel Skill Mix	medium experience	20	

7. Historical Experience Risk			Sandia has wide experience with the command and control applications, but only minimal experience with Ada and DoD software development efforts.
Experience Level	medium	20	
Personnel Stability	medium	20	
8. SNL Risk			This project is an external WR project that is key to Sandia's future business base in the area of non-nuclear weapons. Operational Reliability of the software is expected to be very high. Customer satisfaction is critical to Sandia's future in this area.
Operational Reliability	high	500	
Customer Satisfaction	high	200	
9. Other Risk Areas			
Security	high	100	Need Software Security/Safety Plan.
Safety	high	100	High level of security and safety is required.
Customer Proprietary Data	medium	50	Protection of customer data is required.
Totals Risk Range: xLow < 500: Minimal Set Low 500 - 999: Basic Set Medium 999 - 1499: Small Set High 1500 - 1999: Medium Set xHigh Risk > = 2000: Large Set	xhigh risk	2440	Select Large Document Set: SDP, SQAP, SCMP, SRS, SDD, CODE, SSTP, SSMP, SVVP, SDF Plus: SSP since safety/security is an issue SUG since user interface is critical WR qualification documentation, since project is so large; any DoD contractual documentation not already covered

Blank Page

Appendix E: Documentation Standards and Contacts

The following paragraphs contain brief descriptions of documentation references and standards whose content provided a basis for this volume of the Sandia Software Guidelines. Sources for these standards and document template outlines include the International Standards Organization (ISO), Institute of Electrical and Electronic Engineers (IEEE), and Department of Defense (DoD). The following list of contacts, addresses, and phone numbers will probably change but should provide a starting place for obtaining information on the latest status of documentation standards.

ISO standards:

ISO Central Secretariat
Case postale 56, CH-1211
Geneve 20, Switzerland
International Telephone Number: (022) 749-01-11
International Facsimile Number: (022) 41-22-733-34-30

IEEE standards:

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08854-1331
US Telephone Number (General Information): 800-678-4333

DoD standards:

Space & Naval Warfare Systems Command
SPAWAR 10-12
2451 Crystal Drive (CPK-5)
Arlington, VA 22245-5200
US Telephone Number: 703-602-4491

Defense Quality and Standardization Office
5203 Leesburg Pike, Suite 1403
Falls Church, VA 22041-2466
US Telephone Number: 703-756-2340

Other possible contacts for software documentation standards:

American National Standards Institute (ANSI)
1430 Broadway
New York, NY 10018

American Society of Mechanical Engineers (ASME)
Order Department
22 Law Drive
P.O. Box 2300
Fairfield, NJ 07007-2300

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
US Telephone Number: 801-777-7989

E.1 International Standards Related to Documentation

The following list identifies some of the international standards related to software documentation:

- Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry, ANS 10.4, a 43-page document published by the American Nuclear Society in 1987 [ANS10.4];
- Information Processing - Guidelines for the Documentation of Computer-Based Application Systems, ISO 6592, First Edition, a 20-page standard published by the International Organization for Standardization (ISO) in 1985 [ISO6592];
- Information Processing Systems - User Documentation and Cover Information for Consumer Packages, ISO 9127, First Edition, an 11-page document published by the ISO in 1988 [ISO9127]; and
- Information Technology - Guidelines for the Management of Software Documentation, ISO TR9294, First Edition, a 12-page document published by the ISO in 1990 [ISO9294].

E.2 IEEE Standards Related to Software Documentation

The following paragraphs identify and briefly describe the IEEE standards and guides related to software documentation. The synopsis provided for each of the listed references are extracted from the "IEEE Standards Collection, Software Engineering," 1994 Edition [IEEEstds], or the standard itself if not published in this reference.

E.2.1 IEEE Standard for Software Quality Assurance Plans [IEEE730]

This standard has legal liability as its basic rationale. It is directed toward the development and maintenance of critical software, that is, where failure could impact safety or cause large financial or social losses. The orientation is toward delineating all of the planned and systematic actions on a particular project that would provide adequate confidence that the software product conforms to establish technical requirement. This standard establishes a required format and a set of minimum contents for software quality assurance plans. It should be noted that the "Example Software Quality Assurance Plan" contained in Appendix A reference [SSGv1] is based on this standard, reference [IEEE730].

E.2.2 IEEE Standard for Software Configuration Management Plans [IEEE828]

The standard identifies requirements for configuration identification, configuration control, configuration status accounting and reporting, and configuration audits and reviews. The implementation of those requirements provides a means by which the evolution of the software product items are recorded, communicated, and controlled. This provides assurance of the integrity and continuity of the software product items as they evolve through the software development and maintenance life cycle.

The last paragraph under section 1.1 of reference [SSGv4], states " ... this volume provides information that should supplement the content of related IEEE standards ... and assist the user in the application of these nationally recognized standards to their particular projects."

E.2.3 IEEE Standard for Software Test Documentation [IEEE829]

This standard defines the content and format of eight documents that cover the entire testing process. The Test Plan prescribes the scope, approach, resources, and schedule of the testing activities. It identifies the items to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the Plan. Test specification is covered by three document types (Test-Design Specification, Test-Case Specification, and Test-Procedure Specification), while test reporting is covered by four document types (Test-Item Transmittal Report, Test Log, Test-Incident Report, and Test-Summary Report). The standard shows the relationships of these documents to one another and to the test process they document.

E.2.4 IEEE Guide for Software Requirements Specifications [IEEE830]

This guide describes alternate approaches to good practices in the specification of software requirements. To enable the reader to make knowledgeable choices, extensive tutorial material is provided. The guide covers attributes of a good software requirements specification, as well as specification methodologies and associated formats.

E.2.5 IEEE Standard for Software Verification and Validation Plans [IEEE1012]

This standard has a threefold purpose:

- to provide, for both critical and non critical software, uniform and minimum requirements for the format and content of Software Verification and Validation Plans (SVVPs);
- to define, for critical software, specific minimum Verification and Validation tasks and their required inputs and outputs that shall be included in SVVPs; and
- to suggest optional verification and validation tasks to be used to tailor SVVPs as appropriate for the particular VERIFICATION AND VALIDATION effort.

E.2.6 IEEE Recommended Practice for Software Design Descriptions [IEEE1016]

A software design description is a representation of a software system. It is used as a medium for communicating software design information. This recommended practice describes that documentation of software designs. It specifies the necessary information content and the recommended organization for a software design description.

E.2.7 IEEE Standard for Software Project Management Plans [IEEE1058]

This standard specifies the format and content of software project management plans. It does not specify the procedures or techniques to be used in the development of project management plans, nor does it provide examples of project management plans. Instead, the standard sets a foundation for an organization to build its own set of practices and procedures for developing project management plans.

E.2.8 IEEE Standard for Software User Documentation [IEEE1063]

This standard provides minimum requirements on the structure and information content of user documentation. It applies primarily to technical substance rather than to style. Users of this standard may develop their own style manual for use within their organizations to implement the requirements of this standard.

E.2.9 IEEE Standard for Software Maintenance [IEEE1219]

This standard defines the process for performing the maintenance of software. It prescribes requirements for process, control and management of planning, execution, and documentation of software maintenance activities.

E.2.10 IEEE Standard for Software Safety Plans [IEEE1228]

This standard describes the minimum acceptable requirements for the content of a Software Safety Plan. It applies to the development, procurement, maintenance, and retirement of safety-critical software; for example, software products whose failure could cause loss of life, serious harm, or have widespread negative social impact.

E.3 Department of Defense Software Documentation References

The following paragraphs identify Department of Defense standards, handbooks and documentation templates for software for reference [MIL498]. A commercial IEEE standard version of this documentation will be available in the near future; a draft is contained in reference [IEEE1498]. Documents in reference [MIL498] for which there are no current IEEE equivalent documents include the following:

- DoD DID (Data Item Description) DI-IPSC-81428 - Software Installation Plan
- DoD DID DI-IPSC-81429 - Software Transition Plan
- DoD DID DI-IPSC-81431 - System/Subsystem Specification
- DoD DID DI-IPSC-81441 - Software Product Specification
- DoD DID DI-IPSC-81442 - Version Description Document
- DoD DID DI-IPSC-81447 - Computer Programming Manual
- DoD DID DI-IPSC-81448 - Firmware Support Manual

Documents in reference [MIL498] for which some IEEE equivalent documents exist include the following:

- DoD DID DI-IPSC-81427 and IEEE STD 1058.1-1987 address documentation of software project management plans;
- DoD DIDs DI-IPSC-81433/81434 and IEEE STD 830-1993 address Software Requirements Specifications;
- DoD DIDs DI-IPSC-81435/81436/81437 and IEEE STD 1016-1987 address documentation of software designs; and
- DoD DIDs DI-IPSC-81438/81439/81440 and IEEE STD 829-1983 address documentation related to software test planning, specification (description), and reporting;
- DoD DIDs DI-IPSC-81443/81444/81445/81446 and IEEE STD 1063-1987 address software user documentation.

In addition, the following closely-related DoD documents from reference [DOD2168] and [DOD973] have IEEE equivalent documents:

- DOD DID DI-QCIC-80572 and IEEE STD 730-1989 address Software Quality Assurance Plans; and
- DOD DID DI-CMAN-80858A and IEEE STD 828-1990 address Software Configuration Management Plans.

E.3.1 Department of Defense Software Standards and Handbooks

The following paragraphs briefly describe the Department of Defense standards and guides that identify the documentation related to software projects. The synopsis provided for each of the listed references is extracted from their associated Foreword or Purpose paragraphs.

E.3.1.1 MIL Standard 498, Software Development and Documentation [MIL498]

This standard can be applied to contractors, subcontractors, or Government in-house agencies performing software development. The term "acquirer" is used for the organization requiring the technical effort, "developer" for the organization performing the technical effort, "contract" for the agreement between those parties, "Statement of Work" for the list of tasks to be performed by the developer, and "subcontractor" for any organization tasked by the developer to perform part of the required effort. Software Development is used as an inclusive term encompassing new development, modification, reuse, reengineering, maintenance, and all other activities resulting in software products. This standard and its Data Item Descriptions (DIDs) are meant to be tailored for each software project to which they are applied.

E.3.1.2 DoD Standard 2168, Defense System Software Quality Program [DOD2168]

This standard contains requirements for the development, documentation, and implementation of a software quality program. The software quality program includes: planning for and conducting evaluations of the quality of software, associated documentation, and related activities; and planning for and conducting the follow-up activities necessary to assure timely and effective resolution of problems. The applicable documentation is identified within the standard.

E.3.1.3 MIL Standard 973, Configuration Management [MIL973]

This standard defines the configuration management requirements that are to be selectively applied, as required, throughout the life cycle of any configuration item. This standard applies to hardware and software components as well as the system as a whole, and can be applied to contractors, subcontractors, or Government in-house agencies performing system development or support. The configuration management system is for the control of all configuration documentation, physical media, and physical parts representing or comprising the product. For software, the system must address the evolving developmental configuration and support environments used to generate and test the product. The system configuration management plan can directly include software configuration management or may reference a separate software configuration management plan. A general format for a configuration management plan is described in Appendix A of this standard and various documentation forms for engineering product release, change proposal, change notice, and waiver are contained in other appendices.

E.3.2 Department of Defense Data Item Descriptions for Software

The Data Item Descriptions identified in the following lists have been grouped together into functional areas. The presentation order for these groups is project planning, requirements, design, test, description of final (as-built) products, and product support.

- Data Item Descriptions for Software Project Plans
  - ⇒ Software Development Plan, DI-IPSC-81427
  - ⇒ Software Quality Program Plan, DI-QCIC-80572
  - ⇒ Software Configuration Management Plan, DI-CMAN-80858A
- Data Item Descriptions for System and Software Specification Documents
  - ⇒ Operational Concept Description, DI-IPSC-81430
  - ⇒ System/Subsystem Specification, DI-IPSC-81431
  - ⇒ Software Requirements Specification, DI-IPSC-81433
  - ⇒ Interface Requirements Specification, DI-IPSC-81434
- Data Item Descriptions for Software Design Description
  - ⇒ System/Subsystem Design Description, DI-IPSC-81432
  - ⇒ Software Design Description, DI-IPSC-81435
  - ⇒ Interface Design Document, DI-IPSC-81436
  - ⇒ Database Design Document, DI-IPSC-81437
- Data Item Descriptions for Software Test Documents
  - ⇒ Software Test Plan, DI-IPSC-81438
  - ⇒ Software Test Description, DI-IPSC-81439
  - ⇒ Software Test Report, DI-IPSC-81440
- Data Item Descriptions for Software As-Built Product Documents
  - ⇒ Software Product Specification, DI-IPSC-81441
  - ⇒ Software Version Description, DI-IPSC-81442
- Data Item Descriptions for Software Support and User Documents
  - ⇒ Software Installation Plan, DI-IPSC-81428
  - ⇒ Software Transition Plan, DI-IPSC-81429
  - ⇒ Software User Manual, DI-IPSC-81443
  - ⇒ Software Center Operator Manual, DI-IPSC-81444
  - ⇒ Software Input/Output Manual, DI-IPSC-81445
  - ⇒ Computer Operation Manual, DI-IPSC-81446
  - ⇒ Computer Programming Manual, DI-IPSC-81447
  - ⇒ Firmware Support Manual, DI-IPSC-81448

Blank Page

Distribution:

MS0660	Margaret Olson	(02122)
MS1007	Charleene Lennox	(02172)
MS0535	Lorraine Baca	(02615)
MS0535	Marianna Eisenhour	(02615)
MS0535	Laney Kidd	(02615)
MS0535	Patty Trellue	(02615)
MS0977	Ann Chipman	(05501)
MS0746	Eric Tomlin	(06613)
MS0944	Nancy Freshour	(07901)
MS0944	Paula McAllister	(07901)
MS0974	Walt Huebner	(09421)
MS0974	Paul Attermeier	(09426)
MS0977	Ann Hodges	(09432)
MS0775	Pam Harris	(09617)
MS0638	Michael Blackledge	(12326) - 5 copies
MS0638	Dwayne Knirk	(12326)
MS0638	Gerald McDonald	(12326)
MS0638	Dave Peercy, Editor	(12326) - 5 copies
MS0627	Fred Trussell	(12334)
MS1375	Drayton Boozer	(12400)
MS0661	Joe Schofield	(13316)
	Sandia Software Guidelines Library	
MS0809	Betty Straba	(13411) - 150 copies
	Technical Reports Distribution List	
MS9018	Central Technical Files	(08523-2)
MS0899	Technical Library	(13414) - 5 copies
MS0619	Print Media	(12615)
MS0100	Document Processing For DOE/OSTI	(07613-2) - 2 copies

Blank Page

Org.	Bldg.	Name	Rec'd by	Org.	Bldg.	Name	Rec'd by

Sandia National Laboratories