

DEC 01 1993

Mr. Dwight E. Shelor, Associate Director
for Systems and Compliance
Office of Civilian Radioactive Waste Management
U. S. Department of Energy
1000 Independence Avenue, SW
Washington, DC 20585

Dear Mr. Shelor:

SUBJECT: SOFTWARE QUALITY ASSURANCE

In February 1993 the Nuclear Regulatory Commission issued NUREG/BR-0167, "Software Quality Assurance Program and Guidelines." The abstract of the NUREG states: "This document offers guidance to both NRC organizations and NRC contractors in the development and maintenance of software for use by the NRC staff." Therefore it is not required that DOE or its contractors follow the guidance provided in the NUREG in the current geologic repository activities.

However, we believe the guidance is appropriate for software that may be important to safety or important to licensing of a geologic repository. Therefore a copy of NUREG/BR-0167 is enclosed for the benefit of you and your staff.

A written response to this letter or the enclosed NUREG report is not required. If you have any questions, please call Jack Spraul of my staff on (301) 504-2446.

Sincerely,

15/

C. William Reamer, Acting Director
Repository Licensing and Quality Assurance
Project Directorate
Division of High-Level Waste Management
Office of Nuclear Material Safety
and Safeguards

Enclosure: As stated *at the shelf. 9303A0127*
cc: See next page *NUREG/BR-0167 12/1/93*

070003

*102:3
WM-11
NH16 /*

DEC 01 1993

Distribution for attached letter of _____, from the NRC (Holonich) to the DOE (Shelor), SUBJECT: SOFTWARE QUALITY ASSURANCE

- cc: R. Loux, State of Nevada
- T. J. Hickey, Nevada Legislative Committee
- J. Meder, Nevada Legislative Counsel Bureau
- C. Gertz, DOE/NV
- D. Weigel, GAO
- M. Murphy, Nye County, NV
- M. Baughman, Lincoln County, NV
- D. Bechtel, Clark County, NV
- P. Niedzielski-Eichner, Nye County, NV
- B. Mettam, Inyo County, CA
- V. Poe, Mineral County, NV
- F. Mariani, White Pine County, NV
- R. Williams, Lander County, NV
- L. Fiorenzi, Eureka County, NV
- J. Hoffman, Esmeralda County, NV
- C. Schank, Churchill County, NV
- L. Bradshaw, Nye County, NV

DISTRIBUTION

CNWRA	NMSS R/F	HLPD R/F	LSS
LPDR	ACNW	PDR	CENTRAL FILE
BJYoungblood, HLWM	JLinehan, HLWM	RBallard, HLGE	MFederline, HLHP
On-Site Reps	EShelburne, LSSA	QA Section, HLPD	BMabrito, CNWRA

OFC	HLPD	C	HLPD	C	HLPD	N		
NAME	JSpraul/dh		KHooks		CWReamer			
DATE	12/01/93		12/01/93		12/01/93			

C = COVER

E = COVER & ENCLOSURE

N = NO COPY

S:DOE-SQA.Ltr

OFFICIAL RECORD COPY

December 1, 1993

Rec'd with little dtd

NUREG/BR-0167

10/1/93

9312130050

SOFTWARE
DEVELOPMENT

SOFTWARE
MAINTENANCE

SOFTWARE
TESTING

Software Quality Assurance Program and Guidelines

SOFTWARE
DOCUMENTATION

SOFTWARE
CONVERSION

PROJECT
MANAGEMENT

SOFTWARE
DESIGN

SOFTWARE

QUALITY

ASSURANCE

U.S. Nuclear Regulatory Commission

1027

9303190127 LL R

SOFTWARE
DEVELOPMENT

SOFTWARE
MAINTENANCE

SOFTWARE
TESTING

Software Quality Assurance Program and Guidelines

SOFTWARE
DOCUMENTATION

SOFTWARE
CONVERSION

PROJECT
MANAGEMENT

SOFTWARE
DESIGN

SOFTWARE

February 1993

QUALITY

ASSURANCE

Division of Information Support Services
Office of Information Resources Management
U.S. Nuclear Regulatory Commission
Washington, DC 20555

ABSTRACT

This document offers guidance to both NRC organizations and NRC contractors in the development and maintenance of software for use by the NRC staff.

This document is based on various industry standards,

shown under references, and therefore meets the current industry standards for the operational levels of software described herein. This document may provide guidance but does not address the complex issue of software quality for software used in nuclear plants.

CONTENTS

	<i>Page</i>
Abstract	iii
Acknowledgements	ix
1 Introduction	1
1.1 Purpose	1
1.2 Scope and Applicability	1
1.3 The NRC Software Development/Sustaining Engineering Environment	1
1.4 Use of This Document	2
1.5 Organization of This Document	2
1.6 Maintenance of This Document	2
1.7 Style Used in This Document	2
2 The Software Life Cycle	5
2.1 Concepts and Definitions	5
2.2 Requirements Definition	5
2.3 Design	5
2.4 Implementation	5
2.5 Qualification Testing	5
2.6 Installation and Acceptance	6
2.7 Operations and Sustaining Engineering	6
2.8 Retirement and Archiving	6
3 Verification and Validation	7
3.1 Concepts and Definitions	7
3.2 Verification and Validation Activities	7
3.2.1 Verification and Validation Planning Activities	7
3.2.2 Formal Life Cycle Reviews and Audits	8
3.2.3 Formal Peer Inspections	10
3.2.4 Testing	10
3.3 Techniques and Tools	11
4 Documentation and Deliverables	13
4.1 Concepts and Definitions	13
4.2 Software Project Plan	13
4.3 Software Requirements Documentation	13
4.4 Software Design Documentation	13
4.5 Software Implementation Documentation	13
4.6 Software Verification and Validation Documentation	13
4.7 User Documentation	14
4.8 Other Documentation	14
4.9 Deliverables	14
4.9.1 Documentation Deliverables	14
4.9.2 Software Deliverables	14
4.10 Techniques and Tools	15

	<i>Page</i>	
5	Project Management	17
	5.1 Concepts and Definitions	17
	5.2 Project Planning and Organizing	17
	5.2.1 Required Inputs to the Contract	17
	5.2.2 Estimating	18
	5.2.3 Methodology, Standards, and Procedures	18
	5.2.4 The Software Project Plan	18
	5.3 Project Tracking and Oversight	19
	5.4 Supplier Control	20
	5.5 Metrics	20
	5.6 Security	20
	5.7 Training	20
	5.8 Risk Management	20
	5.9 Techniques and Tools	21
6	Configuration Management	23
	6.1 Concepts and Definitions	23
	6.2 Baselines	23
	6.3 Change Control	24
	6.4 Status of Baselines and Changes	24
	6.5 Software Development Library	24
	6.6 Software, Access, and Media Control	24
	6.7 Configuration Audits	24
	6.8 Techniques and Tools	25
7	Nonconformance Reporting and Corrective Action	27
	7.1 Concepts and Definitions	27
	7.2 Activities	27
	7.2.1 Nonconformance Detection and Reporting	27
	7.2.2 Impact Assessment and Corrective Action	28
	7.2.3 Tracking and Management Reports	28
	7.3 Interrelationships	28
	7.4 Techniques and Tools	28
8	Quality Assessment and Improvement	29
	8.1 Concepts and Definitions	29
	8.2 Responsibility for Quality Assessment and Improvement	29
	8.3 Documentation for Quality Assessment and Improvement	29
	8.4 Quality Assessments	29
	8.5 Quality Records Collection, Maintenance, and Retention	29
	8.6 Quality Improvement	29
	8.7 Techniques and Tools	29
9	Software Developed Before Issuance of This Document	33
	Appendix A Sample Software Project Management Plan	35
	Appendix B Glossary	55
	Appendix C Reference Documents	57

TABLES

	<i>Page</i>
1-1 Summary of Typical Life Cycle Activities and Documents	3
3-1 Verification and Validation Activities by Major Life Cycle Activity	8
3-2 Formal Life Cycle Reviews and Audits	9
8-1 Assessments of Products and Processes Used in Software Development	30

FIGURE

5-1 Table of Contents for a Software Project Plan	19
---	----

ACKNOWLEDGEMENTS

The Office of Information Resources Management wishes to acknowledge the contributions of NRC staff and contractors in developing this document. The final textual content was organized and written by Frank J. Douglas, SOFTRAN, Inc. NRC staff who participated in reviews and discussion briefings during the development

process were: Emily Robinson, Wil Madison and John Voglewede, IRM; Frank Coffman and Leo Beltracchi, RES; Jack Spraul and John Buckley, NMSS; Jim Stewart, Ralph Caruso, and Tony Mendiola, NRR; Steve Arndt, AEOD; and Mark Stella, ACRS.

1 INTRODUCTION

1.1 Purpose

It is the purpose and intent of this document to offer guidance to both NRC organizations and NRC contractors in the development and maintenance of software for use by the NRC staff.

1.2 Scope and Applicability

Software quality assurance is the planned and systematic pattern of all actions necessary to provide adequate confidence that a software product conforms to established technical requirements. Thus, the scope of software quality assurance includes both management and technical aspects of software development and maintenance. Therefore, this document provides guidelines for: the software life cycle; verification and validation activities; documentation and deliverables; project management; configuration management, nonconformance reporting and corrective action; and quality assessment and improvement.

Three levels of software are defined to make clear the wide variety of software used by the NRC. The three levels are:

1. Level 1 Software—Technical application software used in a safety decision by the NRC (an example would be RELAP5)
2. Level 2 Software—Technical or non-technical application software not used in a safety decision by the NRC (an example would be an agency financial software system)
3. Level 3 Software—Technical or non-technical application software not used in a safety decision and having local or limited use by the NRC (examples would include a macro for Lotus 1-2-3)

The guidelines in this document apply to Level 1 and Level 2 software only; they do not apply to Level 3 software or any other software.

The degree of applicability of these guidelines will depend on the level of software being developed, its purpose and use, and a managerial judgment of the cost-effectiveness of each software quality activity. Most projects should incorporate verification and validation, configuration management, and documentation control activities.

1.3 The NRC Software Development/Sustaining Engineering Environment

There are three types of organizations involved with NRC software: the regulated industry, NRC contractors, and NRC staff. These guidelines do not apply to the regulated industry. NRC contractors develop and maintain two general types of application software: 1) technical/scientific and 2) administrative/management information systems (MIS). Minimal software development and maintenance are done directly by the NRC staff.

The roles of NRC staff and NRC contractors in software development and maintenance can be divided into three categories: sponsors, developers, and users. A sponsor is the NRC organization that sponsors and manages the software development/maintenance effort. The sponsor acts as the acquirer or buyer for the user. A developer is the organization, usually a contractor, that develops or maintains the software. A user is the organization who utilizes the software product produced by the developer. The user is involved in defining requirements and should be made a partner during the development effort to help ensure that the product being built will meet the user's needs.

The authority for categorizing the software to be developed (either Level 1, Level 2, or Level 3) resides with the sponsor. The user's concurrence with the categorization should be sought. The Information Resources Management (IRM) organization is available for consultation during the categorization process.

The development and maintenance of software is a project, i.e., it has definitive start and end dates and a product(s) is delivered upon completion. Both the sponsor and the developer assign responsibility for the successful completion of the project to a project manager.

The IRM Office is responsible for the coordination of the NRC software quality assurance (SQA) initiative embodied in this document. IRM is responsible for maintaining this document.

The IRM organization is also responsible for the establishment and coordination of the NRC SQA Working Group. The objectives of this working group are to:

1. Facilitate communications (e.g., successes, lessons learned) about software development and maintenance among the NRC organizations involved with acquiring and using software

2. Facilitate technology transfer of the best practices in the management and technical aspects of software development and maintenance
3. Provide a focal point for improvements to the guidelines in this document

1.4 Use of This Document

The SQA Working Group is chaired by IRM/DISS and has members from each major office involved with software development and maintenance: ACRS, AEOD, IRM, NRR, NMSS, and RES.

This document will be used by sponsor project managers as a guide in developing inputs (e.g., the statement of work) to the request for proposal for software to be developed, and by developer project managers as a planning tool. It can also be used as a software quality assurance reference by sponsors and developers.

The guidelines in this document are not intended to be applied rigidly. They should be used within the context of NRC policy and Federal standards, as applicable to the project at hand, as well as with cost-effective management and engineering judgment based on past experience.

This document applies to all software currently in use, being developed, or planned for future development. Owners of software developed prior to the publication of this document should read Section 9 in particular. Future software plans should implement all major elements of this document, but the extent of implementation must be decided as part of the planning process by the sponsor and developer.

1.5 Organization of This Document

In addition to this introductory section,

- Section 2 defines the software life cycle.
- Section 3 discusses verification and validation activities.
- Section 4 identifies deliverables, including required documentation.
- Section 5 addresses project management.
- Section 6 addresses configuration management.
- Section 7 discusses nonconformance reporting and corrective action.

- Section 8 addresses quality assessment and improvement
- Section 9 discusses application of the guidelines in this document to software developed before issuance of the document

Table 1-1 shows an overview of a typical software life cycle. It is meant as a quick-look reference. It contains only major activities performed and documentation produced. It is not meant to be complete listing of all activities performed or documents produced.

1.6 Maintenance of This Document

The maintenance of this document is the responsibility of the IRM organization. Changes to the document will be issued as change pages as required. When the number of change pages is deemed to be excessive, a new version of the document will be published.

Suggestions for improvement are solicited from the entire NRC software community: sponsors, developers, and users.

1.7 Style Used in This Document

Most of this document is written using verbs in the indicative mood. The indicative mood is the standard mood of verbs, for example:

The software life cycle defined in this section *provides* the basis for planning and implementing a software development or maintenance project.

The life cycle *consists* of the following major activities: requirements definition, design, implementation, qualification testing, installation and acceptance, operations and sustaining engineering, and retirement and archiving.

When the intent is to communicate explicitly a suggestion or guideline to the sponsor or developer, we have chosen to use the imperative mood of the verb. Three examples follow:

Define the requirements so that they are correct, complete, verifiable, consistent, and technically feasible.

Perform planning activities for verification and validation in parallel with requirements definition activities.

Require the developer's approach to quality assessment and improvement to be documented.

Table 1-1. Summary of Typical Life-Cycle Activities and Documents

	Requirements Definition	Design	Implementation	Qualification Testing	Installation and Acceptance	Operations and Sustaining Engineering
Principal Technical Activities Performed	<ul style="list-style-type: none"> Analyze requirements 	<ul style="list-style-type: none"> Develop preliminary design Develop detailed design Develop preliminary user's documentation 	<ul style="list-style-type: none"> Develop unit designs and unit code 	<ul style="list-style-type: none"> Conduct qualification testing in accordance with the qualification test plan and qualification test procedures 	<ul style="list-style-type: none"> Install the software on the target computer Conduct acceptance testing in accordance with the acceptance test plan and acceptance test procedures 	<ul style="list-style-type: none"> Perform all the activities of development, as appropriate Perform sustaining engineering activities to ensure that the original capabilities and design remain intact
Verification and Validation Activities Performed	<ul style="list-style-type: none"> Conduct requirements inspections Conduct Software Requirements Review 	<ul style="list-style-type: none"> Conduct design inspections Plan qualification and acceptance tests Conduct Preliminary Design Review Conduct Critical Design Review 	<ul style="list-style-type: none"> Develop unit and integration test plans and procedures Conduct unit and integration testing Develop qualification and acceptance test procedures 	<ul style="list-style-type: none"> Witness qualification tests 	<ul style="list-style-type: none"> Witness acceptance tests 	<ul style="list-style-type: none"> Perform all verification and validation activities, as appropriate
Documentation and Deliverables Developed	<ul style="list-style-type: none"> Software requirements documentation Overall verification and validation plan Software Project Plan 	<ul style="list-style-type: none"> Software design documentation Qualification test plan Acceptance test plan Preliminary user's documentation 	<ul style="list-style-type: none"> Qualification test procedures Acceptance test procedures Unit and integration test results 	<ul style="list-style-type: none"> Qualification test report Nonconformance reports based on test results Final user's documentation 	<ul style="list-style-type: none"> Acceptance test report Nonconformance reports based on test results 	<ul style="list-style-type: none"> Update all documentation, as required Develop new documentation, as appropriate
Project Management Activities Performed	<ul style="list-style-type: none"> Develop Software Project Plan Ensure users participate in requirements definition Conduct tracking and oversight activities 	<ul style="list-style-type: none"> Conduct tracking and oversight activities Re-plan as required 	<ul style="list-style-type: none"> Conduct tracking and oversight activities Re-plan as required 	<ul style="list-style-type: none"> Conduct tracking and oversight activities Re-plan as required 	<ul style="list-style-type: none"> Conduct tracking and oversight activities Re-plan as required 	<ul style="list-style-type: none"> Conduct tracking and oversight activities Re-plan as required

Table 1-1. Summary of Typical Life-Cycle Activities and Documents
(continued)

	Requirements Definition	Design	Implementation	Qualification Testing	Installation and Acceptance	Operations and Sustaining Engineering
Configuration Management Activities Performed	<ul style="list-style-type: none"> Develop or update configuration management procedures Place software requirements documentation under configuration control (i.e., establish the requirements baseline) 	<ul style="list-style-type: none"> Place software design documentation under internal developer configuration control 	<ul style="list-style-type: none"> Place code and qualification test documentation under internal developer configuration control 	<ul style="list-style-type: none"> Place tested software and associated documentation under configuration control (i.e., establish the product baseline) 	<ul style="list-style-type: none"> Place tested software and associated documentation under configuration control (i.e., establish the operational baseline) 	<ul style="list-style-type: none"> Maintain the requirements baseline and developmental configuration Establish new product baseline and new operational baseline
Nonconformance Reporting and Corrective Action Activities Performed	<ul style="list-style-type: none"> Develop or update nonconformance reporting and corrective action procedures Document requirements documentation nonconformances 	<ul style="list-style-type: none"> Document design and requirements documentation nonconformances 	<ul style="list-style-type: none"> Document design and requirements documentation nonconformances 	<ul style="list-style-type: none"> Document code and qualification test nonconformances Document design and requirements documentation nonconformances 	<ul style="list-style-type: none"> Document code and acceptance test documentation nonconformances Document design and requirements documentation nonconformances 	<ul style="list-style-type: none"> Document all nonconformances, as applicable
Quality Assessment and Improvement Activities Performed	<ul style="list-style-type: none"> Assess software requirements documentation and software project plan Assess requirements definition process Initiate product and process improvement activities, as required 	<ul style="list-style-type: none"> Assess software design Assess the qualification test plan and acceptance test plan Assess design process Initiate product and process improvement activities, as required 	<ul style="list-style-type: none"> Assess unit designs, unit code, unit test plans, integration test plans, and integration test procedures Assess implementation process Assess qualification test procedures and acceptance test procedures Initiate product and process improvement activities, as required 	<ul style="list-style-type: none"> Assess qualification test results Assess qualification test process Initiate product and process improvement activities, as required 	<ul style="list-style-type: none"> Assess acceptance testing activities Assess acceptance test process Initiate product and process improvement activities, as required 	<ul style="list-style-type: none"> Assess all products and processes Initiate product and process improvement activities, as required

2 THE SOFTWARE LIFE CYCLE

2.1 Concepts and Definitions

The software life cycle defined in this section provides the basis for planning and implementing a software development or maintenance project. The life cycle consists of the following major activities:

1. Requirements definition
2. Design
3. Implementation
4. Qualification testing
5. Installation and acceptance
6. Operations and sustaining engineering
7. Retirement and archiving

Each major activity leads to specific products that can be measured, evaluated, approved, and controlled. No strict chronological constraints exist between major activities. The major activities may overlap in time and may be applied iteratively or recursively.

Each major activity is accompanied by verification actions that ensure that the products and processes of the major activity meet the requirements for those products and processes. Verification actions are discussed in Section 3, and the documentation and software deliverables of the software life cycle are discussed in Section 4.

The software life cycle presented here must be:

1. Tailored to fit the scope of each development/maintenance effort
2. Used within the context of NRC policy and Federal standards, as applicable to the project at hand, as well as with cost-effective management and engineering judgment based on past experience

Some projects will not encounter all major activities.

2.2 Requirements Definition

The requirements definition process is the set of activities that results in the specification, documentation, and review of the requirements that the software product must satisfy, including functionality, performance, design con-

straints, attributes, and external interfaces. The requirements form the basis for the software plans, products, and activities.

Ensure that the documented requirements define the response of the software to anticipated classes of input data (including erroneous data) and provide the information and detail necessary to design the software (e.g., mathematical models, equations, data requirements).

Define the requirements so that they are correct, complete, verifiable, consistent, and technically feasible. Perform planning activities for verification and validation in parallel with requirements definition activities.

Because requirements inevitably change as a project evolves, manage the requirements throughout the development and maintenance efforts in accordance with well-defined change control procedures (See Section 6).

2.3 Design

The design process is the set of activities that results in the development, documentation, and review of a software design that meets the requirements defined in the software requirements documentation.

As the design evolves, events (e.g., additional insight into problem areas) may necessitate the modification of the requirements documentation. Manage changes to requirements documentation in accordance with well-defined change control procedures.

2.4 Implementation

The implementation process is the set of activities that results in software that has:

1. Been constructed in accordance with the design documentation and coding standards
2. Undergone informal unit and integration testing

As the software is implemented, events (e.g., additional insight into data flow patterns) may necessitate the modification of the design, requirements, and/or verification and validation documentation. Manage changes to documentation in accordance with well-defined change control procedures.

2.5 Qualification Testing

The qualification testing process is the set of activities associated with:

1. Formally testing the implemented software, using test cases defined in the verification and validation documentation, against the baselined requirements defined in the software requirements documentation
2. Reviewing and analyzing the test results to ensure that the implemented software meets requirements and that the software produces correct results for all test cases executed

To evaluate technical adequacy, the software test results can be compared to results from alternative methods, such as:

1. Analysis without computer assistance
2. Other validated computer programs
3. Experiments and tests
4. Standard problems with known solutions
5. Confirmed published data and correlations

2.6 Installation and Acceptance

Section 3.2.4 discusses qualification testing in more detail.

Installation activities include one or more of the following:

1. Installing hardware
2. Installing the developed/maintained software
3. Integrating the developed/maintained software with other components (e.g., other software components, hardware, data)
4. Reformatting or creating data bases
5. Verifying that all components have been included

Acceptance activities include:

1. Execution of acceptance tests (which typically consist of some of the qualification test cases plus additional test cases)
2. Documentation of the acceptance of the software by the sponsor

This stage of the life cycle usually concludes with the user accepting the software for operational use. The responsibility for the sustaining engineering and maintenance of the software may be assigned to an organization different from the sponsor and/or the developer of the software.

2.7 Operations and Sustaining Engineering

Operation of the software is conducted by the user in accordance with the operation and usage instructions in the user's documentation. Sustaining engineering is set of software engineering and software maintenance activities needed to

1. Retain the software's initial functionality and design integrity (software engineering)
2. Remove latent errors (corrective maintenance)
3. Respond to new or revised requirements (perfective maintenance)
4. Adapt the software to changes in the operating environment (adaptive maintenance)

Perform sustaining engineering activities in a traceable, planned, and orderly manner based on:

1. The major life-cycle activities described in Sections 2.1 through 2.6
2. The verification and validation activities described in Section 3
3. Updating the required documentation and software deliverables as described in Section 4
4. The project management activities described in Section 5
5. The configuration management activities described in Section 6
6. The nonconformance reporting and corrective action activities described in Section 7
7. The software quality assessment and improvement activities described in Section 8

2.8 Retirement and Archiving

Retirement means the support for a software product is terminated, and the routine use of the software is prevented. The software and its documentation are archived.

3 VERIFICATION AND VALIDATION

3.1 Concepts and Definitions

Verification is the process of ensuring that the products and processes of each major activity of the life cycle meet the standards for the products and the objectives of that major activity. Validation is the process of demonstrating that the as-built software meets its requirements. Testing is the process of detecting errors and verifying performance. Testing typically includes unit, integration, qualification, and acceptance testing.

Independent verification and validation (IV&V) is verification and validation by an organization that is both technically and managerially separate from the organization responsible for developing the software. Sponsors and users of Level 1 software should together decide if the expense of a separate IV&V contractor is warranted for their project.

Examples of verification activities include:

1. Formal major life cycle reviews and audits (e.g., Preliminary Design Review)
2. Formal peer inspections (e.g., code inspections, documentation reviews)
3. Informal tests (e.g., unit and integration testing)

Testing is the primary method of software validation. Qualification and acceptance testing, which are formal tests, are validation activities. Validation is accomplished by review and demonstration in a live or simulated environment. The objectives of validation activities are to ensure that:

1. The as-built software correctly and adequately performs all intended functions
2. The software does not perform any unintended function that either by itself or in combination with other functions can degrade the entire system
3. All non-functional requirements (e.g., performance, design constraints, attributes, and external interfaces) are met

Software validation activities include the development of test plans, test procedures, and test reports.

Subject the validation of modifications to previously validated software to selective regression testing. The objectives of regression testing are to:

1. Detect possible errors introduced during the modification process
2. Ensure that the modifications have not caused unintended adverse effects
3. Validate that the modified software still meets specified requirements

3.2 Verification and Validation Activities

This section discusses verification and validation planning activities (Section 3.2.1); formal life cycle reviews and audits (Section 3.2.2); peer inspections (Section 3.2.3); and testing (Section 3.2.4). Table 3-1 shows verification and validation activities by major life-cycle activity. This table is intended to show the approximate time in the life cycle that these activities are performed. It is not intended to be applied rigidly. Like all of the guidelines in this document, management and engineering judgment, in conjunction with cost-effectiveness decisions, must be used in the application of these guidelines to the project at hand.

3.2.1 Verification and Validation Planning Activities

Planning for verification and validation takes place during the sponsor's initial planning for the project (e.g., the proposal stage) as well as during the requirements definition, design, and implementation major activities of the life cycle. Planning activities include:

1. Development or tailoring of procedures for conducting formal life cycle reviews
2. Development or tailoring of procedures for reviewing documentation and other deliverables
3. Development or tailoring of procedures for conducting inspections

Table 3-1. Verification and Validation Activities by Major Life Cycle Activity

Major Life Cycle Activity	Verification and Validation Activities
Requirements Definition	<ul style="list-style-type: none"> • Inspect requirements • Develop overall verification and validation plan • Conduct the Software Requirements Review
Design	<ul style="list-style-type: none"> • Inspect design • Develop qualification test plan • Develop acceptance test plan • Conduct the Preliminary Design Review • Conduct the Critical Design Review
Implementation	<ul style="list-style-type: none"> • Develop unit test plans • Inspect unit designs, unit code, and unit test plans • Perform unit testing • Inspect unit test results • Develop integration test plans • Inspect integration test plans • Perform integration testing • Inspect integration test results • Develop qualification test procedures
Qualification Testing	<ul style="list-style-type: none"> • Perform qualification testing • Write qualification test report • Develop acceptance test procedures
Installation and Acceptance	<ul style="list-style-type: none"> • Perform acceptance testing • Write acceptance test report
Sustaining Engineering and Operations	<ul style="list-style-type: none"> • Perform, as appropriate, the verification and validation activities defined above for requirements definition, design, implementation, qualification testing, and installation and acceptance • Perform regression testing as well as new tests for all levels of testing, as appropriate

4. Definition of a detailed test methodology, including standards for test documentation, specifically for test plans, test procedures, and test reports for both qualification and acceptance testing
5. Preparation of a validation matrix showing the relationship of software requirements to the software architecture down to the unit level and to the tests used to verify the requirements
6. Identifying the need for development and test tools, equipment, and data

3.2.2 Formal Life Cycle Reviews and Audits

A formal review, with sponsor and developer management and technical personnel participating, is held at or near the end of a major activity of the life cycle. The objective of the formal reviews is to evaluate the deliverable products, the progress, and to a lesser degree, the processes of the most recent life-cycle phase. Table 3-2 summarizes the formal major life cycle reviews and audits by major life-cycle activities.

Table 3-2. Formal Life Cycle Reviews and Audits.

Major Life Cycle Activity	Formal Reviews and Audits
Requirements Definition	<ul style="list-style-type: none"> • Software Requirements Review
Design	<ul style="list-style-type: none"> • Preliminary Design Review • Critical Design Review
Implementation	<ul style="list-style-type: none"> • Qualification Test Readiness Review
Qualification Testing	<ul style="list-style-type: none"> • Software Configuration Audit
Installation and Acceptance	<ul style="list-style-type: none"> • Software Configuration Audit • Post Mortem Review
Operations and Sustaining Engineering	<ul style="list-style-type: none"> • The formal reviews and audits above, as applicable

The products associated with each formal review are:

1. The documents to be reviewed (e.g., the software requirements documentation for the Software Requirements Review)
2. The agenda for the review
3. The hardcopy presentation materials
4. The minutes that document the activities and results of the review
5. The updated documents that were reviewed

Allow sufficient time for sponsor review participants to review the documents prior to the review (2 to 3 weeks, say). Identify in the agenda the review participants and their specific responsibilities during the review. Assign a person to capture key discussion items and actions items, especially those related to specific assignments for updating the documentation that is the object of the review. Document in the review minutes all proposed revisions to the reviewed documents and all actual changes to the reviewed documents, and place the updated documents under configuration control after approval by the sponsor.

The paragraphs below discuss each formal life cycle review and audit.

3.2.2.1 Software Requirements Review

Conduct the Software Requirements Review at the end of requirements definition. The primary objective of this review is to assure that the sponsor and the developer understand and agree on the intent, completeness, verifiability (through testing or other means), consistency, and technical feasibility of the requirements. Secondary objectives are to review other documentation products available at this time, including, for example, the Software Project Plan and the overall verification and validation plan.

3.2.2.2 Preliminary Design Review

Conduct the Preliminary Design Review when the preliminary design (software architecture) has been designed. The primary objective of this review is to assure that the preliminary design is complete (meets all the requirements), verifiable (through testing or other means), consistent, and technically feasible.

3.2.2.3 Critical Design Review

Conduct the Critical Design Review when the design is complete. Design completion criteria should be defined clearly in the Software Project Plan. Suggested design completion criteria are:

1. All software units have been identified and all interfaces between and among the units have been defined
2. All elements of the database have been defined down to the data item level.

The primary objective of this review is to assure that the design is complete (meets all the requirements and meets design completion criteria), verifiable (through testing or other means), consistent, and technically feasible.

3.2.2.4 Qualification Test Readiness Review

Conduct the Qualification Test Readiness Review when integration testing and the qualification test procedures are complete. The primary objective of this review is to assure that the as-built software; the software documentation; and qualification test data, test tools, test configuration, and test team are ready for formal qualification testing.

3.2.2.5 Software Configuration Audit

Conduct the Software Configuration Audit twice, first at the completion of qualification testing and second at the completion of acceptance testing. The primary objective of this audit is to ensure that the as-built software:

1. Meets its requirements as documented in the software requirements documentation
2. Conforms to its technical documentation
3. Does not contain any unauthorized changes

3.2.2.6 Post Mortem Review

Conduct the Post Mortem Review after the software has been accepted. The objective of this audit is to capture the lessons learned from the project for use by future projects.

3.2.3 Formal Peer Inspections

A formal peer inspection is a detailed examination of a product on a step-by-step or line-by-line basis. The purpose of conducting formal peer inspections is to find errors. The group that performs a peer inspection is composed of peers of the person who developed the product to be inspected. Peer inspections are objective approaches that have been proved very effective in verifying that products meet requirements.

For Level 1 software, require the developer to

1. Subject each intermediate product and final product of development and maintenance (i.e., all documentation, all code) to an internal peer inspection
2. Make available to the sponsor the written procedure and the product standards that govern peer inspections

3. Make available, if requested by the sponsor, records that document the results of all peer inspections

For Level 2 software, encourage the developer to work toward subjecting each intermediate product and final product of development and maintenance to an internal peer inspection.

See Section 3.3 for more discussion of formal peer inspections.

3.2.4 Testing

Testing is the process of exercising or evaluating a software product or part of a software product by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results. Testing approaches depend on the number of levels of testing. For most cases, four levels of testing are sufficient:

1. Unit testing
2. Integration testing
3. Qualification testing
4. Acceptance testing

A unit of software is an element of the software design that can be compiled or assembled and is relatively small (e.g., 100 lines of high-order language code). Require that each software unit be separately tested.

Integration testing focuses on a collection of related units that performs an identifiable functional requirement. Require that integration testing be carried out. Both unit testing and integration testing are classified as informal testing because a formal test plan is not required.

Qualification testing is the process that allows the sponsor to determine whether a software product complies with its requirements. Acceptance testing is the process that allows the sponsor to determine whether a software product complies with its requirements after it has been installed in its operational environment.

In many cases, acceptance tests will, to a large degree, coincide with qualification tests. In some cases, qualification tests and acceptance tests are the same in all respects, in which case the test hierarchy telescopes down to three levels of testing. Both qualification testing and acceptance testing are classified as formal testing because a formal test plan is required.

Testing may be either requirements-driven or design-driven. Informal testing may be either requirements-driven or design-driven.

3.2.4.1 Design-Driven and Requirements-Driven Testing

Design-driven or white-box testing is the process where the tester examines the internal workings of the code. Design-driven testing is accomplished by selecting input data and other parameters based on the internal logic paths to be checked. The goals of design-driven testing include ascertaining correctness of:

1. All paths through the code. For most software products, this can be feasibly done only at the unit test level
2. Interfaces between units
3. Size and timing of critical elements of code

Requirements-driven or black-box testing is done by selecting input data and other parameters based on the software requirements and observing the outputs and reaction of the software. In addition to testing for satisfaction of requirements, some of the objectives of requirements-driven testing are to ascertain:

1. Computational correctness
2. Proper handling of boundary conditions, including extreme inputs and conditions that cause extreme outputs
3. State transitioning as expected
4. Proper behavior under stress or high load
5. Adequate error detection, handling, and recovery

Sometimes the term "operational testing" is used. Operational testing is either the random or statistically-controlled application of the software in its actual environment or in a simulated version of the operational environment. An example of such testing is the so-called beta test use of an applications software package by individuals typical of the intended user population. In the terminology used above, such operational testing and beta testing would be qualification testing and are requirements-driven.

3.2.4.2 Informal Testing

Require the developer to perform informal tests to:

1. Ensure software units and combinations of software units are correct

2. Measure progress

"Informal" in this case does not mean the tests are conducted in a casual manner, just that no deliverable test plan is required, the sponsor is not formally involved, the witnessing of the testing is not required, and that the prime purpose of the test is to find errors.

3.2.4.3 Formal Testing

For Level 1 and Level 2 software defined in this document, formal testing is always requirements-driven and its purpose is to demonstrate that the software meets its requirements. The reader is cautioned not to confuse formal testing with formal proof-of-correctness methods, which are formal techniques used to prove mathematically that a computer program satisfies its specifications.

Require that formal tests include:

1. A sponsor-approved test plan and procedures
2. Test witnesses
3. A record of all nonconformance
4. A test report

If the software is to be developed and delivered in increments or builds, there may be incremental qualification and acceptance tests. As a practical matter, any contractually required test is usually considered a formal test; others are "informal."

After acceptance of a software product, all changes to the product should be accepted only as a result of a formal test. Include regression testing in all post-acceptance testing. Regression testing involves rerunning previously used acceptance test cases to ensure that the change did not introduce error into previously accepted software.

3.3 Techniques and Tools

Perhaps more tools have been developed to aid verification and validation of software (especially testing) than any other software activity. The tools available include code tracers, special-purpose memory dumpers and formatters, data generators, simulations, and emulations. Some tools are essential for testing any significant set of software, and, if they have to be developed, may turn out to be a significant cost and schedule driver. Ensure that the need for test tools is examined during software design.

An especially useful technique for finding errors is the formal peer inspection. The formal peer inspection is

performed by a team, each member of which has a well-defined role. The team is led by a moderator, who is formally trained in the inspection process. The team includes a reader, who leads the team through the item to be inspected; one or more reviewers, who look for errors in the product; a recorder, who notes the faults; and the author, who helps explain the product.

This formal, highly structured inspection process has been extremely effective in finding and eliminating errors. It can be applied to any product of the software development process, including documents, design, and code. One of its important side benefits is the direct feedback to the developer/author, often resulting in significant improvement in product quality.

4 DOCUMENTATION AND DELIVERABLES

4.1 Concepts and Definitions

This section identifies the documentation and software deliverables essential to a successful software development project. This section should be used as a starting point to help determine a realistic set of documentation requirements and deliverables for the project at hand. A realistic set of documentation requirements will result from tailoring the information in this section in light of past experience with similar projects, the size of the software, and sponsor requirements. Small and short-duration projects will normally produce fewer documents or combine related documents.

4.2 Software Project Plan

A result of the developer's planning process, a Software Project Plan is written by the developer and details how the developer will manage the software project. The Software Project Plan is discussed in detail in Section 5.2.4.

4.3 Software Requirements Documentation

Software requirements documentation specifies the requirements that the software to be developed/maintained must meet. Include in this documentation the following, as applicable:

1. **Functionality**—the functions that the software is to perform
2. **Performance**—the time-related requirements of software operation such as speed, response time, etc.
3. **Design constraints imposed on implementation activities**—any elements that will restrict design options (e.g., specifying the hardware platform or the programming language)
4. **Attributes**—characteristics of the software, its acceptance, or use (e.g., portability, acceptance criteria, access control, availability, maintainability, etc.)
5. **External interfaces**—interactions with people, hardware, and other software

An item can be called a software requirement only if its achievement can be verified and validated. It is important that each software requirement be traceable throughout the stages of the software life cycle.

4.4 Software Design Documentation

In software design documentation, specify the overall structure of the software so that it can be translated into code. Include in this documentation:

1. A description of the major elements of the software as they relate to the requirements
2. A description of the theoretical basis, physical model, mathematical model, control flow, data flow, control logic, and data structure
3. An identification and detailed definition of the software units and data elements of the software architecture.

4.5 Software Implementation Documentation

Software implementation documentation includes unit designs (usually presented as a commentary prologue to the unit's source code) and the unit code itself.

4.6 Software Verification and Validation Documentation

Software verification and validation documentation includes:

1. An overall verification and validation plan that includes a description of:
 - a. The objectives and processes for each review and inspection
 - b. The test methodology including the objectives of each level of testing (e.g., unit, integration, qualification, acceptance)
 - c. Contents of each level of formal test documentation (test plans, procedures, reports)
 - d. How verification and validation documentation will be organized so that traceability of reviews, inspections, and tests to requirements and design will be apparent
2. Agenda, presentation materials, and minutes for formal life-cycle reviews and audits
3. Results of formal peer inspections
4. Informal test plans for unit and integration testing

5. Informal test procedures for unit and integration testing
6. Informal test reports for unit and integration testing
7. Formal test plans for qualification and acceptance testing
8. Formal test procedures for qualification and acceptance testing
9. Formal test reports for qualification and acceptance testing

4.7 User Documentation

In user documentation, include:

1. A description of the user's interaction with the software, and a description of any required training necessary to use the software
2. Input and output specifications and formats, including sample cases
3. A description of the limitations of the software
4. A description of anticipated errors and how the user can respond
5. For each error message, provide the message, an explanation of the message, how the message may have come about, and actions that may or should be taken
6. Information about obtaining user and sustaining engineering support

4.8 Other Documentation

Other documentation may include the following:

1. Software Operations Concept
2. Standards and Procedures Manual
3. Software Maintenance Manual
4. Software Engineering Notebooks

4.9 Deliverables

Documentation deliverables are discussed in Section 4.9.1, and software deliverables are discussed in Section 4.9.2.

4.9.1 Documentation Deliverables

The sponsor must decide what the contract deliverables should be. For large projects, the following documentation deliverables are suggested:

1. Software Project Plan
2. Requirements documentation
3. Overall verification and validation plan
4. Design documentation (delivered three times: initially at the Preliminary Design Review; updated at the Critical Design Review; and updated after acceptance testing)
5. Qualification test plan
6. Qualification test procedures
7. Qualification test report
8. Acceptance test plan
9. Acceptance test procedures
10. Acceptance report
11. User documentation

For smaller projects, documents can be combined. For example:

1. The requirements documentation can be combined with the design documentation
2. The overall verification and validation plan can be combined with the Software Project Plan
3. Test plans and test procedure documents can be combined

4.9.2 Software Deliverables

The decision about what software deliverables to require depends on numerous considerations, including

1. Whether the software will be implemented and delivered in segments or builds (for large software products, builds have been proved to be a very effective risk-reduction technique)
2. What organization will perform maintenance and the environment needed to perform maintenance.

If the maintainer is different from the developer, a maximal subset of the following list of possible software deliverables should be chosen:

1. Source code
2. Object code
3. Executable code
4. Test cases for formal testing, including machine-readable test procedures
5. Required job control language, e.g., to compile, link, load, and execute the software

6. Software and job control language necessary to establish and maintain the software development library
7. Software engineering environment
8. Software test environment
9. Non-developmental software

4.10 Techniques and Tools

Numerous tools exist for generating documentation: word processing programs, desktop publishing programs, graphics programs, spelling checkers, grammar checkers, etc.

There are numerous standards for documentation that should be consulted before deciding on the documentation requirements to be levied on the developer.

Consult an experienced project manager for his/her experiences when deciding what deliverables to choose.

5 PROJECT MANAGEMENT

5.1 Concepts and Definitions

Assign the responsibility for each software development or maintenance effort within an NRC sponsor organization to a project manager. This sponsor project manager should be an experienced NRC employee trained in managing the technical and personnel aspects of the project. He or she is assigned by sponsor management the responsibility for the successful completion of the project, i.e., for meeting technical objectives within cost and schedule constraints. Delegate to the sponsor project manager the authority to negotiate, via the Government's Contracting Officer, commitments with the developer.

Similarly, the developer is expected to assign a project manager who will be responsible for meeting the developer's contractual commitments.

The two basic project management activities, discussed in Sections 5.2 and 5.3, respectively, are: 1) project planning and organizing and 2) project tracking and oversight.

5.2 Project Planning and Organizing

Project planning and organizing involves:

1. Development, by the sponsor project manager, of required inputs to the contract, e.g., the statement of work, schedule, list of deliverables, identification of applicable standards, and software specification
2. The definition, by the sponsor project manager, of work elements necessary to develop or maintain the required software. The work elements are defined in a statement of work that will be a part of the contract with the developer
3. The establishment, by the developer and approval by the sponsor, of budgets and schedules for each defined work element
4. The establishment, by the developer, of a project organization for implementing the project; and assignment of work elements, budgets, and schedules to each organizational entity
5. Documentation of the overall plan for approval by the sponsor project manager

Consider requiring that the developer's organization assigned to plan for and perform formal testing be different from and independent of the organization(s) that designed and implemented the software.

5.2.1 Required Inputs to the Contract

Hold the sponsor project manager responsible for developing the following inputs to the contract: the statement of work, top-level schedule, list of deliverables, identification of applicable standards, and software specification.

The statement of work defines the activities required of the developer. The statement of work should:

1. Define what the developer must do, not what the software must do (the software specification defines what the software must do)
2. Contain explicit tasks modeled after the life cycle activities defined in Section 2 and the verification and validation activities defined in Section 3
3. Identify the deliverable documentation and software required of the developer (See Section 4)
4. Require the developer to perform project planning and organization activities resulting in the Software Project Plan (See Section 5.2.4)
5. Require the developer to perform project tracking and oversight activities and deliver periodic progress reports as indicated in Section 5.3
6. Require the developer to perform configuration management activities as indicated in Section 6
7. Require the developer to establish and maintain a nonconformance reporting and corrective action system as indicated in Section 7
8. Require the developer to establish and maintain a quality assessment and improvement program as indicated in Section 8

Develop the top-level schedule around the:

1. Formal life-cycle reviews and audits discussed in Section 3.2.2 (e.g., Software Requirements Review, Preliminary Design Review, etc.)
2. Deliverables

The list of deliverables should contain:

1. The software end products

2. Required documentation
3. Agenda, presentation materials, and minutes for formal reviews
4. Progress reports

The identification of applicable standards may include:

1. Programming language standards (e.g., FORTRAN 77)
2. Coding standards
3. Documentation standards
4. De facto standards embedded in software and documentation to be maintained

The software specification documents the requirements the software is to satisfy. The software specification is often preliminary and subject to analysis and expansion by the developer during the requirements definition process. The software specification should contain:

1. Technical goals and objectives
2. Identification of users and their interaction with, and use of the software
3. The characteristics presented in Section 4.3

5.2.2 Estimating

Both the sponsor and developer should derive estimates for the size of the software products and documentation, software development resources and costs, and critical target computer resources. These estimates should be derived from in-house experience-based data using documented procedures. Discuss overall projected software size (estimated combined with actuals) at each formal review.

5.2.3 Methodology, Standards, and Procedures

Developers should work toward basing software planning (and monitoring) activities on documented methodologies, standards, and procedures.

5.2.4 The Software Project Plan

Require the developer to submit, for sponsor project manager approval, a Software Project Plan that appropriately and realistically documents the required software activities and contractual commitments. When approved by the sponsor project manager, the Software Project Plan becomes the baseline management plan. Figure 5-1 shows a suggested table of contents for the Software Project Plan.

Section 1 of the Software Project Plan should be kept brief. Because the plan should be kept up to date, consider requiring the developer to submit any changes to the plan with the monthly progress reports.

Section 2 documents the developer's management approach. The following paragraphs provide guidance to the developer for the contents of each subsection:

1. 2.1—Planning Approach. Briefly describe the approach used to plan the project.
2. 2.2—Tracking and Oversight Approach. Briefly describe the approach used to track: technical progress, conformance to the planned schedule, and costs as related to actual work performed. Include approach to: supplier control; metrics; security; training; and risk management
3. 2.3—Organization, Tasks, and Responsibilities. Describe the project organization. Show how the tasks of the statement of work are assigned to responsible elements of the project organization.
4. 2.4—Scheduling. Provide the initial, top-level project schedule and the rationale for arriving at this schedule.
5. 2.5—Resources. Identify project resources, including staffing, software engineering facilities and environment, and support tools. Identify Government Furnished Equipment (GFE) and Government Furnished Information (GFI) required by the developer.
6. 2.6—Configuration Management. Identify and define project baselines. Include or reference procedures for: change control; determining status of baselines, proposed changes, and implemented changes; release control; the software development library; and code, access, and media control.

SECTION 1 — INTRODUCTION	
1.1	Project Background and Objectives
1.2	Plan Scope and Organization
1.3	Plan Maintenance
SECTION 2 — MANAGEMENT APPROACH	
2.1	Planning Approach
2.2	Tracking and Oversight Approach
2.3	Organization, Tasks, and Responsibilities
2.4	Scheduling
2.5	Resources
2.6	Configuration Management
SECTION 3 — TECHNICAL APPROACH	
3.1	Implementing the Life Cycle Tasks of the Statement of Work
3.2	Verification And Validation Approach
3.3	Nonconformance Reporting and Corrective Action
3.4	Quality Assessment and Improvement Approach
3.5	Deliverables

Figure 5-1
Table of Contents for a Software Project Plan

Section 3 documents the developer's technical approach. The following paragraphs provide guidance to the developer for the contents of each subsection:

1. **3.1—Implementing the Life Cycle Tasks of the Statement of Work.** Describe briefly how each major life-cycle task of the statement of work will be implemented.
2. **3.2—Verification and Validation Approach.** Describe the verification and validation approach. Include the elements addressed in Section 3.

3. **3.3—Nonconformance Reporting and Corrective Action Approach.** Describe the nonconformance reporting and corrective action process, including nonconformance detection and reporting, impact assessment and corrective action, and tracking and management reports. Identify the interrelationships, if applicable, with the status accounting function of configuration management. Identify any techniques and tools used (e.g., use of a data base management system).
4. **3.4—Quality Assessment and Improvement Approach.** Describe the quality assessment and improvement approach. Include the elements addressed in Section 8.
5. **3.3—Deliverables.** Identify all deliverables and the dates they are due.
6. **3.4—Standards, Procedures, Conventions, and Metrics.** Identify all standards, procedures, conventions, and metrics to be used. Identify both product standards (e.g., documentation standards, coding standards) and process procedures (e.g., inspection and review procedures).

5.3 Project Tracking and Oversight

Project tracking and oversight involves:

1. Monitoring, assessing, and reporting technical progress
2. Determining and reporting schedule and cost status
3. Developing and implementing corrective action plans as required

Monitoring, assessing, and reporting technical progress requires tracking actual results and performance of the software project against the Software Project Plan. Implementation of planned verification and validation, configuration management, and quality assessment and improvement activities are part of the ordinary tracking and oversight functions. The key to monitoring progress on an ongoing basis is to maintain communications at all levels of the developer and sponsor organizations. Use formal mechanisms such as reviews and reports and informal mechanisms such as meetings and brainstorming sessions to keep project members and the project manager informed. Track technical progress, costs, critical target computer resources, the schedule, estimates for lines of code, and risks in as quantitative way as possible.

Hold the developer project manager responsible for determining and reporting schedule and cost status in terms

of variances from the baseline plan. Require the developer to report the reasons for schedule and cost variances, e.g., unexpected problem complexity and changes in requirements.

Take corrective actions when the actual results and performance of the software project deviate significantly from the Software Project Plan and current schedule. Basic corrective actions may include adding staff, extending the work week, and/or upgrading (or downgrading) the skill mix.

5.4 Supplier Control

If the developer plans to use subcontractors or vendors, the sponsor and developer project managers should ensure that:

1. The developer selects qualified subcontractors and vendors
2. The software standards, procedures, and product requirements for the subcontract comply with the prime contractor's contractual commitments
3. A Software Project Plan as outlined in Section 5.2.4 is required of the major subcontractors
3. Commitments between the prime contractor and subcontractor are understood and agreed to by both parties
4. The prime contractor tracks the subcontractor's actual results and performance against the commitments
5. Potential technical and business risks are identified and managed

5.5 Metrics

Plan to measure both the products being developed (the software and its documentation) and the processes being used. Process-related metrics (e.g., number of errors found in the requirements or design) are often useful in evaluating the programmatic risks involved in software development.

Establish a technical performance measurement program, using the following steps:

1. Develop a comprehensive list of key technical performance parameters, associated with both products and processes, that can be predicted and estimated

(e.g., accuracy, data access times, response time, number of errors found in requirements).

2. For each parameter, specify the requirement and develop a time-phased profile with tolerance bands that depict the acceptable range of performance as the project progresses.
3. Plan for periodic analysis and predictions of these parameters, especially in conjunction with formal life-cycle reviews
4. Keep the sponsor project manager informed of all unfavorable trends and the corrective action plans being initiated to resolve them

5.6 Security

The Computer Security Act of 1987 requires Federal agencies to identify each computer system that contains sensitive information and to prepare and implement a plan for the security and privacy of these systems. OMB Bulletin No. 90-08 provides guidance for preparing such plans, but does not address the unique security requirements of local area networks. NUREG/BR-0166, *Instructions for Preparing Security Plans for Local Area Networks in Compliance With OMB Bulletin No 90-08*, provides guidelines for preparing security plans for local area networks and contains OMB Bulletin No. 90-08 as an appendix. If the proposed project will be a sensitive application, IRM/DISS should be notified on the sensitive system survey form.

5.7 Training

The right people properly trained are necessary for a successful project. Required training includes both management and technical training in the knowledge and skills of a variety of disciplines. Sponsor and developer project managers need to evaluate training needs for:

1. Sponsor management and technical personnel
2. Developer management and technical personnel
3. Maintainer management and technical personnel
4. Operations management and technical personnel

When the evaluation is complete, invest in the required training.

5.8 Risk Management

Initiate the risk management program while the technical, schedule, and budget planning efforts are under way.

The following activities are typical of a risk management program:

1. Identify, assess, document, and rank technical, cost, resource, and schedule risks
2. Develop a risk mitigation plan
3. Formalize the risk management program
4. Review the risk management program regularly

5.9 Techniques and Tools

Numerous commercially available tools exist for:

1. Project management support
2. Estimating software costs and schedules
3. Earned-value reporting

Many organizations have made the commitment to provide in-depth training to software project managers.

6 CONFIGURATION MANAGEMENT

6.1 Concepts and Definitions

For a project to be successful, the developer and sponsor must establish and maintain integrity of the software and its documentation as they evolve throughout the life cycle. Because requirements, the design, the code, and the test environment can change significantly and often, it is essential that change be managed successfully. Briefly stated, configuration management is change management.

Fundamental to configuration management are the concepts of a baseline and change control. A baseline is a document or software that has been formally reviewed and agreed upon by the developer and sponsor, that thereafter serves as the basis for further development and that can be changed only through formal change control procedures. Change control is the process by which a change to a baseline is proposed, evaluated, approved or rejected, scheduled, and tracked.

There are four major functions of configuration management:

1. The identification and establishment of baselines
2. Controlling both changes to baselines and the release of baselines
3. Recording and reporting the status of baselines, change requests, and implemented changes
4. Verifying, through auditing, the correctness and completeness of baselines prior to release

For a software configuration management program to be successful, experience has shown that most of the following conditions exist:

1. The content and status of the software and documentation baselines are known at all times
2. The developer follows a written configuration management policy that has the following characteristics:
 - a. Responsibility for configuration management for each project is explicitly assigned
 - b. Configuration management is implemented on products throughout the product's life cycle

- c. Configuration management is implemented for externally-deliverable products and for appropriate products used inside the organization
 - d. All projects have a repository for storing key software engineering elements and associated configuration management records
 - e. The software baselines and configuration management activities are audited on a regular basis
3. A group that is responsible for coordinating and implementing configuration management for the project exists or is established
 4. Adequate resources and budget for performing configuration management activities are provided
 5. Members of the configuration management group are trained in the objectives, procedures, and methods for performing their assigned activities
 6. The configuration management activities are reviewed with the project manager on a regular basis

6.2 Baselines

Establish controlled and stable baselines for planning, managing, and building the system. Explicitly identify as project baselines software products (e.g., source code, object code, test cases) and software process specifications (e.g., standards and procedures) that are needed to establish and maintain stability of the software activities.

Establish a naming or labeling system that:

1. Uniquely identifies all project entities (e.g., documents, software elements, test cases)
2. Identifies changes by revision or version
3. Uniquely identifies each configuration/version of revised software for use

Establish the following baselines that will be controlled by the sponsor's configuration control board (CCB) (See Section 6.3):

1. The project management baseline consisting of the Software Project Plan, documented standards and procedures, and up-to-date budgets and schedules

2. The requirements baseline consisting of the software requirements documentation plus approved changes
3. The product baseline consisting of software and documentation resulting from the qualification testing activity
4. The operational baseline consisting of software and documentation resulting from the installation and acceptance activity that is placed into operation

The developmental configuration is the developer's software and associated technical documentation that defines the evolving software products during development. It contains the software design and implementation products (software design documentation, code, test cases, and related information). Require the developer to apply internal configuration control procedures to the developmental configuration as it evolves. See Sections 6.3 and 6.6.

6.3 Change Control

Once a baseline has been established, changes to the baseline can be made only in accordance with formal change control procedures. To manage changes to baselines:

1. Establish a board (i.e., a configuration control board (CCB)) controlled by the sponsor project manager that has the authority for managing the software baselines and approving or rejecting proposed changes to them
2. Establish and follow a documented procedure for initiating, recording, reviewing, approving or rejecting, and tracking change requests for baselines
3. Establish and follow a documented procedure for ensuring that all changes, especially those to the requirements and design, are appropriately reviewed for "ripple" effects and incorporated into all related activities
4. Establish and follow a documented procedure to create and control the release of software baseline products

6.4 Status of Baselines and Changes

Track accurately the current status of baselines and changes throughout development and maintenance. To track status accurately:

1. Establish and follow a documented procedure to record the status of baselines and change requests
2. Create and distribute to affected groups and individuals standard reports documenting the configuration management activities

6.5 Software Development Library

Require the developer to establish and maintain a software development library (SDL). An SDL is a controlled collection of software, documentation, and associated tools and procedures used to facilitate the orderly development and subsequent maintenance of software. The SDL contains the developmental configuration as part of its contents. An SDL provides storage of and controlled access to software and documentation in human-readable form, machine readable form, or both. The SDL may also contain management data pertinent to the software development project. The SDL becomes the repository for the software baselines when the product baseline and the operational baseline are established.

6.6 Software, Access, and Media Control

Require the developer to establish and maintain the facilities and procedures used to

1. Maintain, store, secure, and document controlled versions of the software throughout the life cycle. This may be implemented with the SDL (See Section 6.5)
2. Permit authorized and prevent unauthorized access to the software and documentation
3. Identify the media for each software product and the documentation required to store the media, including the copy and restore process
4. Protect software physical media from unauthorized access on inadvertent damage or degradation throughout the life cycle.

6.7 Configuration Audits

Require the developer to plan and execute documentation audits, software configuration audits, and in-process audits. Require the developer to establish and follow a documented procedure to prepare for, conduct, report results from, and track action from configuration audits.

A documentation audit is a line-by-line comparison of revised documentation against the previous version of the documentation to ensure that only approved changes

have been incorporated. A documentation audit is typically performed after a formal life-cycle review (e.g., after the Critical Design Review to ensure only CCB-approved changes to the software requirements documentation and software design documentation have been incorporated).

As indicated in Section 3.2.2.6, the Software Configuration Audit is executed twice, first at the completion of qualification testing and second at the completion of acceptance testing.

Periodic in-process audits are performed to assess how well the configuration management standards and pro-

cedures are being followed and how effective they are in managing the software baselines.

6.8 Techniques and Tools

Use a data base management system as a tool in tracking and reporting on proposed and actual changes to baselines. Often the data base of proposed and actual changes is integrated with the data base used to track and report on nonconformances and associated corrective action (see Section 8).

In addition, choose a software tool, often a part of the operating system utilities, to help manage the SDL.

7 NONCONFORMANCE REPORTING AND CORRECTIVE ACTION

7.1 Concepts and Definitions

A nonconformance, often called a problem, discrepancy, fault, or error, is any failure of any document, code, data structure, or process to meet its requirements or standards. Corrective action is a general name for the process by which nonconformances are corrected, verified, and controlled.

Require the developer to establish and maintain a nonconformance reporting and corrective action system and associated procedures. The purpose of a nonconformance reporting and corrective action system is to report, analyze, correct, and verify nonconformances and collect information from which reports on the overall status of nonconformances can be made.

The need for a nonconformance reporting and corrective action system arises early in the software life cycle, as soon as the first documents and other products are developed. A nonconformance reporting and corrective action system should:

1. Define a nonconformance report form
2. Identify the organization(s) and procedures for:
 - a. Analyzing the nonconformance
 - b. Assigning priorities
 - c. Communicating with the person who reported the nonconformance
 - d. Correcting the nonconformance
 - e. Verifying the correction and/or the corrective action
3. Track the status of the nonconformance and corrective action
4. Produce management reports

7.2 Activities

There are three basic activities associated with a nonconformance and corrective action system:

1. Nonconformance detection and reporting (Section 7.2.1)

2. Impact assessment and corrective action (Section 7.2.2)
3. Tracking and management reports (Section 7.2.3)

7.2.1 Nonconformance Detection and Reporting

Allow nonconformance reports to be filed against any product in any part of the software life cycle by anyone associated with the project. Normally a nonconformance reporting and corrective action system is used after a product is first approved or baselined by its developer and released for wider use. For example, while a developer is unit testing and integration testing the code, errors found may be tracked only locally and not in the nonconformance reporting and corrective action system. After the code is declared correct and released for qualification testing by the implementation group, the nonconformance reporting and corrective action system is used to inform the users of the code and the designer/programmer about nonconformances and to assure that the nonconformances are corrected, verified, and not overlooked.

Examples of the information that a nonconformance report form might contain are:

1. Date and time of the detection of the nonconformance
2. Nonconformance identification (report number)
3. Reporting individual and organization
4. Reporting individual's determination of the criticality of the nonconformance
5. Statement of the nonconformance
6. Organization responsible for analysis of the nonconformance
7. Result of the analysis of the nonconformance
8. The project's determination of the criticality of the nonconformance
9. Organization(s) responsible for designing, implementing, and verifying the corrective action or "fix"

10. Identification of the unit(s) of code, data, or documentation in which corrective action must be taken
11. Summary of the test case results (or other verification activity) indicating that the corrective action was successfully implemented
12. Identification of the date or version of the product(s) or baseline in which the correction will be included
13. Date on which the nonconformance is closed

7.2.2 Impact Assessment and Corrective Action

Evaluate all nonconformances for criticality and level of importance. Consider the following factors:

1. The impact of not correcting the nonconformance
2. The resources required for correcting the nonconformance
3. The impact on other baselined items if the nonconformance is corrected

Ensure that a written procedure exists to control the corrective action process. Include in this procedure a follow-up activity to ensure the nonconformance has been documented and corrected in the appropriate version of software and to assure that adequate testing, including regression testing, has been done.

7.2.3 Tracking and Management Reports

After the nonconformance report is prepared by the individual who found the nonconformance, enter the report data into a controlling system. Data base management systems are often used to increase productivity in the

tracking of nonconformances and providing management reports. Entering the nonconformance report electronically can increase productivity further.

Provide in the nonconformance tracking and reporting system management reports containing such information as nonconformance and correction status, the number of nonconformance found per product, and the criticality of open problems. The data enable the impact of the nonconformance to be evaluated so that the use of resources may be prioritized.

7.3 Interrelationships

The nonconformance reporting and corrective action system is a basic and fundamental tool for project management and for assuring quality products. As such it impacts and interacts with many software management, verification and validation, and quality assessment and improvement activities. For example, it interacts with:

1. Configuration management activities that deal with product changes and versions that result from correcting nonconformances
2. Requirements management activities because some nonconformance reports will contain requirements changes disguised as nonconformances. These reports should result in the opening of a change request
3. Quality improvement activities that study trends in nonconformances in specific products or processes

7.4 Techniques and Tools

Consider using an automated tracking system for nonconformance reports and an automatic capability to identify and record changes that occur as a result of the resolution of the nonconformances.

8 QUALITY ASSESSMENT AND IMPROVEMENT

8.1 Concepts and Definitions

Require the developer to institute a quality assessment and improvement program. The objective of this program is assess and improve the quality of:

1. Deliverable software and documentation
2. The processes used to produce deliverable software
3. Non-deliverable software (software not required to be delivered by the contract)

8.2 Responsibility For Quality Assessment and Improvement

Allow the developer the freedom in assigning responsibility for meeting the objectives of the quality assessment and improvement program. However, for Level 1 software development and maintenance efforts, require that the persons responsible for the assessments of a product or activity be independent of the persons who developed the product, performed the activity, or are responsible for the product or activity. This restriction does not preclude members of the development team from participating in these assessments.

8.3 Documentation For Quality Assessment and Improvement

The developer's approach to quality assessment and improvement will be documented in the Software Project Plan. This approach will be implemented throughout the development or maintenance effort. Developers should work toward defining in detail their methodology for quality assessment and improvement in written procedures.

8.4 Quality Assessments

Require the developer to assess:

1. Software
2. Software documentation
3. Processes used in software development

A prerequisite to any assessment is a yardstick or standard against which a product or process can be measured or assessed. A key yardstick is the developer's "software

plans". This is a collective term used to describe the developer's plans, methodologies, standards, and procedures for software management, software engineering, software verification and validation, software documentation, software product evaluation, and software configuration management.

Table 8-1 identifies the products and processes to be assessed and the objectives of the assessments.

8.5 Quality Records Collection, Maintenance, and Retention

Require the developer to prepare and maintain records of the quality assessment and improvement activities.

Require the developer to prepare a software quality assessment record for each assessment required by the contract. Require these records to contain as a minimum:

1. Assessment date
2. Assessment participants
3. Assessment criteria
4. Assessment results including detected problems, with reference to the appropriate nonconformance reports, as applicable
5. Recommended corrective action

Include in these required records the nonconformance reports that are the basis of the nonconformance reporting and corrective action system outlined in Section 7. Require the developer to make quality records available for sponsor review and to maintain them for the life of the contract.

8.6 Quality Improvement

Encourage the developer to integrate quality assessment activities with quality improvement activities (which may be part of the developer's approach to total quality management).

8.7 Techniques and Tools

Checklists for quality audits and inspections and automated code standards analyzers are examples of tools used in quality assessment activities.

Table 8.1
Assessments of Products and Processes Used in Software Development

Product or Process To Be Assessed	Assurance Objectives
Software	Compliance with the contract and adherence to the software plans
Software plans	<ul style="list-style-type: none"> • All software plans required by the contract have been documented • The software plans comply with the contract • Each software plan is consistent with other software plans
Deliverable software	<ul style="list-style-type: none"> • Each document adheres to the required format documentation • Each document complies to the contract
Software management	Compliance with the contract and adherence to the software plans
Software engineering	Compliance with the contract and adherence to the software plans
Software qualification	<ul style="list-style-type: none"> • The qualification plans and procedures include provisions for all requirements • Software qualification is conducted as required by the contract and as specified in the software plans • The version number of each item being qualified and each item used in qualification is documented • The results of required qualifications are accurately recorded and analyzed to determine whether the software meets its specified requirements • All required facilities for qualification are available
Software configuration	Compliance with the contract and adherence to the software plans management
Software corrective actions	<p>Compliance with the contract and adherence to the software plans and:</p> <ul style="list-style-type: none"> • All nonconformances detected in processes and in products that are under developer or sponsor control are promptly reported and entered into the software corrective action process • Each nonconformance is classified, as required by the contract, and analysis is performed to identify trends in the nonconformances reported • Action is initiated on the nonconformances and adverse trends, resolution is achieved, status is tracked and reported, and records are maintained for the life of the contract • Corrective actions are evaluated to: 1) verify that problems have been resolved, 2) verify that adverse trends have been reversed, 3) verify that changes have been correctly implemented in the appropriate processes and products, and 4) determine whether additional problems have been introduced

Table 8.1 (continued)

Product or Process To Be Assessed	Assurance Objectives
Documentation and media distribution	<ul style="list-style-type: none"> • Compliance with the contract and adherence to the software plans • Evaluation of the controls exercised on the internal distribution of deliverable media and documentation
Storage, handling, and delivery	<ul style="list-style-type: none"> • Compliance with the contract and adherence to the software plans • Evaluation of the storage, handling, packaging, shipping, and external distribution of deliverable software and associated documentation
Software development library	<ul style="list-style-type: none"> • The library and its operation comply with the contract and adhere to the software plans • The most recent authorized version of the materials under configuration control are clearly identified and are the ones routinely available from the library • Previous versions of materials under configuration control are clearly identified and controlled to provide an audit trail that permits reconstruction of all changes made to each baseline
Non-developmental software	<p>Evaluate each item of non-developmental software to be incorporated into deliverable software to assure that:</p> <ul style="list-style-type: none"> • Objective evidence exists, prior to its incorporation, that it performs required functions • It was placed under configuration control prior to its incorporation • The data rights provisions are consistent with the contract
Non-deliverable software	<p>Evaluate each item of non-deliverable software used in the qualification or acceptance of deliverable software to assure that:</p> <ul style="list-style-type: none"> • Objective evidence exists, prior to its intended use, that it performs required functions • It was placed under configuration control prior to its use
Deliverable elements of the software engineering and test environments	<p>Evaluate each deliverable element of the software engineering and test environments to assure that:</p> <ul style="list-style-type: none"> • It complies with the contract and adheres to the software plans • Objective evidence exists, prior to its use, that it performs required functions • It was placed under configuration control prior to its use • The data rights provisions are consistent with the contract

Table 8.1 (continued)

Product or Process To Be Assessed	Assurance Objectives
Subcontractor management	<p>Evaluate all subcontractor activity to assure that:</p> <ul style="list-style-type: none"> • All subcontractor-developed software and related documentation deliverable to the sponsor satisfies the requirements of the prime contract • A set of baselined requirements is established and maintained for the software to be developed by the subcontractor • Applicable software quality assessment and improvement requirements are included or referenced in the subcontract or purchase documents • Access is available for developer reviews at subcontractor and vendor facilities • The sponsor has the right to review all software products and activities required by the subcontract, at subcontractor facilities, to determine compliance with the contract. Sponsor review does not constitute acceptance, nor does it in any way replace assessment by the developer or otherwise relieve the developer of his responsibility to furnish acceptable software and associated documentation
Acceptance inspection and preparation for delivery	<ul style="list-style-type: none"> • Compliance with the contract and adherence to the software plans • Evaluation of the controls exercised on the internal distribution of deliverable media and documentation
Participation in formal reviews and audits	<ul style="list-style-type: none"> • Prior to each formal review and audit, assure that 1) all required products will be available and ready in sufficient time for sponsor review before the review meeting and 2) all required preparations have been made • At each formal review and audit, require the developer to present an assessment of the status and quality of each of the development products reviewed • Following each formal review and audit, require the developer to assure that all software-related action items assigned to the developer have been performed

9 SOFTWARE DEVELOPED BEFORE ISSUANCE OF THIS DOCUMENT

All the detailed guidelines in this document obviously cannot be applied in a cost-effective manner to software developed before this document was issued.

However, Level 1 software developed before this document was issued should receive as-is verification and validation or certification based on its length of service and error profile. In addition, the software should be placed under configuration control in accordance with the guidance in ASME Standard NQA 2, Part 2.7, Section 10.2.

Level 1 and Level 2 software developed before this document was issued can benefit from selected application of the software quality assurance principles presented in this document. The following list provides, in relative priority order, suggested actions that can be taken on a step-by-step basis to enhance existing software cost-effectively:

1. Establish and maintain a software development library
2. Institute a nonconformance and corrective action system
3. Develop a set of acceptance test cases
4. Institute a clearly defined test program
5. Institute a well-defined configuration management system, especially a software release system
6. Begin code inspections

Ultimately, however, the extent to which new techniques can or should be introduced into ongoing maintenance efforts is a matter of managerial and engineering judgment.

APPENDIX A

Sample Software Project Plan

Appendix A consists of a sample Software Project Plan. Although the project and the sample plan are fictitious, the sample plan is provided to indicate an acceptable level of detail.

The statement of work for the project is divided into ten tasks as follows:

- 1. Task 1—Requirements Definition**
- 2. Task 2—Design**
- 3. Task 3—Implementation**
- 4. Task 4—Qualification Testing**
- 5. Task 5—Installation and Acceptance**
- 6. Task 6—Verification and Validation**
- 7. Task 7—Project Management**
- 8. Task 8—Configuration Management**
- 9. Task 9—Nonconformance Reporting and Corrective Action**
- 10. Task 10—Quality Assessment and Improvement**

The sample software project plan follows.

SAMPLE SOFTWARE PROJECT PLAN

SECTION 1 --INTRODUCTION

1.1 Project Objectives

The ABC Corporation has been developing, enhancing, and maintaining the XYZ analysis tool for NRC in-house use for the past 8 years. The current contract requires the ABC Corporation to update Version 3.4 of XYZ by:

1. Adding two new major capabilities, C1 and C2
2. Analyzing and implementing corrective action for 30 known nonconformances in Version 3.4 of XYZ
3. Analyzing and implementing a corrective action for as many as 20 yet-to-be-determined nonconformances in Version 3.4 of XYZ.

The contract period of performance is 2 years from the contract start date.

1.2 Plan Scope and Organization

This software project plan defines ABC's management and technical approach to meet the requirements of the contract. It also identifies the standards, procedures, conventions, and metrics that will be applied throughout the project.

Section 2, Management Approach, summarizes ABC's planning approach (Section 2.1); tracking and oversight approach (Section 2.2); project organization, including tasks and responsibilities (Section 2.3); the top-level schedule (Section 2.4); the resources required (Section 2.5); and configuration management approach (Section 2.6).

Section 3, Technical Approach, summarizes ABC's approach to implementing the life-cycle tasks of the statement of work (Section 3.1); verification and validation approach (Section 3.2); nonconformance reporting and corrective action approach (Section 3.3); quality assessment and improvement approach (Section 3.4). The contractually required deliverables are listed in Section 3.5.

Section 4 lists all the standards, procedures, conventions, and metrics that will be applied on the contract.

1.3 Plan Maintenance

This document is intended to be an up-to-date statement of ABC's plan for managing the contract. Therefore, changes to the document will be issued as change pages as required. In accordance with the contract, change pages will be submitted to the NRC sponsor monthly as an attachment to the monthly progress report.

SECTION 2--MANAGEMENT APPROACH

ABC's management approach responds to Task 7, Project Management, and Task 8, Configuration Management, of the statement of work.

2.1 Planning Approach

ABC's planning approach, which was used to generate this plan, consists of the following steps performed in an iterative process:

1. Defining the work
2. Estimating the schedule and staffing
3. Planning for technical performance measurement
4. Planning risk management
5. Performing detailed planning

The work elements were derived from an analysis of the statement of work. Then, based on the analysis results and ABC's past experience, the work elements were structured into a work breakdown structure. In developing the master schedule, ABC took into consideration the technical complexities, the NRC's required milestones, task interdependencies, the estimated number of lines of source code, and the expected staff skill mix.

Analysis of the C1 and C2 capabilities resulted in the identification of the following three key technical performance measures: (1) the estimated source lines of code; (2) the time needed to perform the Monte Carlo analysis for the C1 capability; and (3) the rate of convergence of the new eigenvalue algorithm in the C2 capability.

Two major risks have been identified: (1) meeting the requirement to incorporate an undetermined number of new nonconformances into the new release, Version 4.0, within schedule and cost constraints and (2) the ability to meet the required rate of convergence of the new eigenvalue algorithm in the C2 capability.

Cost accounts were planned in detail to create a sound basis for setting the project budget and for controlling project work activities. A detailed outline for each deliverable document, in conjunction with ABC's past experience providing high-quality documentation, was used to determine the cost and schedule for each deliverable.

The above-discussed planning activities will be repeated if re-planning becomes necessary during the life of the project.

2.2 Tracking and Oversight Approach

Project tracking and oversight involves:

1. Monitoring, assessing, and reporting technical progress
2. Determining and reporting schedule and cost progress
3. Developing and implementing corrective action plans, as required

ABC will track actual technical results and performance against this baseline project plan. Implementation of planned verification and validation, configuration management, and quality assessment and improvement activities will be part of the day-to-day tracking and oversight responsibilities of the ABC management team.

Each ABC manager will be a hands-on manager, i.e., he/she will monitor technical, schedule, and cost status progress on an ongoing basis through: daily person-to-person and telephone contact with their assigned people, weekly staff meetings, the monthly progress meeting with their sponsor counterparts, and internal ABC information systems.

The ABC Project Manager will keep in close contact with the NRC Project Manager by telephone. The ABC Project Manager will plan for and lead the monthly progress meeting where

1. Technical, schedule, and cost status
2. Work performed during the reporting period
3. Work planned for the next month
4. Risks, problems, and concerns and recommended solutions will be discussed.

Within ABC, weekly progress reports from ABC managers and lead technical personnel and informal mechanisms, such as meetings and brainstorming sessions, will keep the Project Manager and the project members informed. ABC's earned-value cost and schedule reporting system will provide a quantitative relationship between technical, schedule, and cost progress. The two major

risks identified above will be reviewed at a minimum monthly and more frequently if required.

If tracking and oversight activities uncover variances from the baseline plan, the ABC management team will take appropriate corrective action. Corrective actions may include one or more of the following:

1. Add staff or extend work week
2. Upgrade or downgrade skill mix
3. Implement specific workarounds
4. Offset an unfavorable variance in one area with favorable variances in other areas
5. Improve productivity through training, process improvement, new tools or techniques, etc.
6. Combine previously separated activities or products

2.2.1 Supplier Control

ABC does not plan to have subcontractors on the contract.

2.2.2 Metrics

ABC will track four key metrics and keep the NRC sponsor apprised of any major changes from anticipated values. The three metrics are:

1. Estimated source lines of code
2. The time needed to perform the Monte Carlo analysis for the C1 capability
3. The rate of convergence of the new eigenvalue algorithm in the C2 capability
4. The number and types of errors uncovered by formal peer inspections

2.2.3 Security

There are no security requirements or implications on the contract.

2.2.4 Training

Prior to the start of the contract:

1. The ABC Project Manager will participate in the 1-week, case-study-oriented project manager's workshop instituted at ABC in 1991
2. All project members, both technical and management, will attend the 24-hour ABC-sponsored course on continuous improvement

2.3 Organization, Tasks, and Responsibilities

The ABC XYZ System Upgrade Project consists of the following organizational elements:

1. Project Management Office
2. Analysis and User Support Group
3. Design and Implementation Group
4. Qualification Test and Configuration Management Group

Each of the three groups reports to the Project Manager and are led by senior technical personnel who have an average of 5 years experience in the design and implementation of the XYZ software. All members of the Qualification Test and Configuration Management Group are and will be independent of the software design and implementation efforts to ensure their freedom of action.

In addition, the Quality Evaluation and Improvement Group, which is a non-project group reporting to the ABC Vice President for Quality Evaluation and Improvement, will work in partnership with the ABC XYZ System Upgrade Project management to meet the requirements of Task 9, Quality Assessment and Improvement, of the statement of work. Table 2-1 shows how the major tasks of the statement of work are assigned to the project organizational elements. The letter "P" implies that the group or Project Manager has primary responsibility and the letter "S" implies that the group or Project Manager has secondary responsibility. Note that for verification and validation activities all organizations are marked with a "P" to indicate that each organizational element is responsible for verifying its own products.

Table 2.1
Responsibilities for the Tasks of the Statement of Work

Responsibilities Tasks	Project Management Office	Analysis and User Support Group	Design and Implementation Group	Qualification Test and Configuration Management Group
Planning and Organizing	P	S	S	S
Tracking and Monitoring	P	P	P	P
Requirements Analysis and Definition		P		
Design		S	P	
Implementation		S	P	
Qualification Testing		S	S	P
Installation and Acceptance		S	S	P
User Support		P	S	S
Verification and Validation	P	P	P	P
Configuration Management	S	S	S	P
Nonconformance Reporting and Corrective Action	S	S	S	P
Quality Assessment and Improvement	P	S	S	S

2.4 Scheduling

The project master schedule is presented in Figure 2-1. This schedule was arrived by analyzing the work to be done in light of ABC's past experience and historical data on previous XYZ projects and other projects of similar size and scope. The analysis showed that NRC's milestones are realistic and can be met if the risks are managed properly. Each group of the project organization will be responsible for developing detailed schedules for their assigned products and activities.

2.5 Resources

The project staffing profile for the 24-month period of performance of the contract is shown in Figure 2-2. This

staffing profile was developed after detailed analysis of the work to be accomplished and of the requirements of the master schedule. The software engineering environment, which has been successfully used for XYZ maintenance for the past 2 years and includes the local area network of 14 QRS Series 7000 Workstations supported by the Super Groupware CASE tool set, is under control of the ABC Project Manager. Thus ABC is confident that this hardware and software suite will be adequate for all tasks through qualification testing. ABC will require 8 hours per day, 6 days a week, of exclusive use of the NRC-furnished DED computer at NRC's Rockville, Maryland, headquarters to support acceptance testing. ABC will review these requirements with the NRC at each formal review.

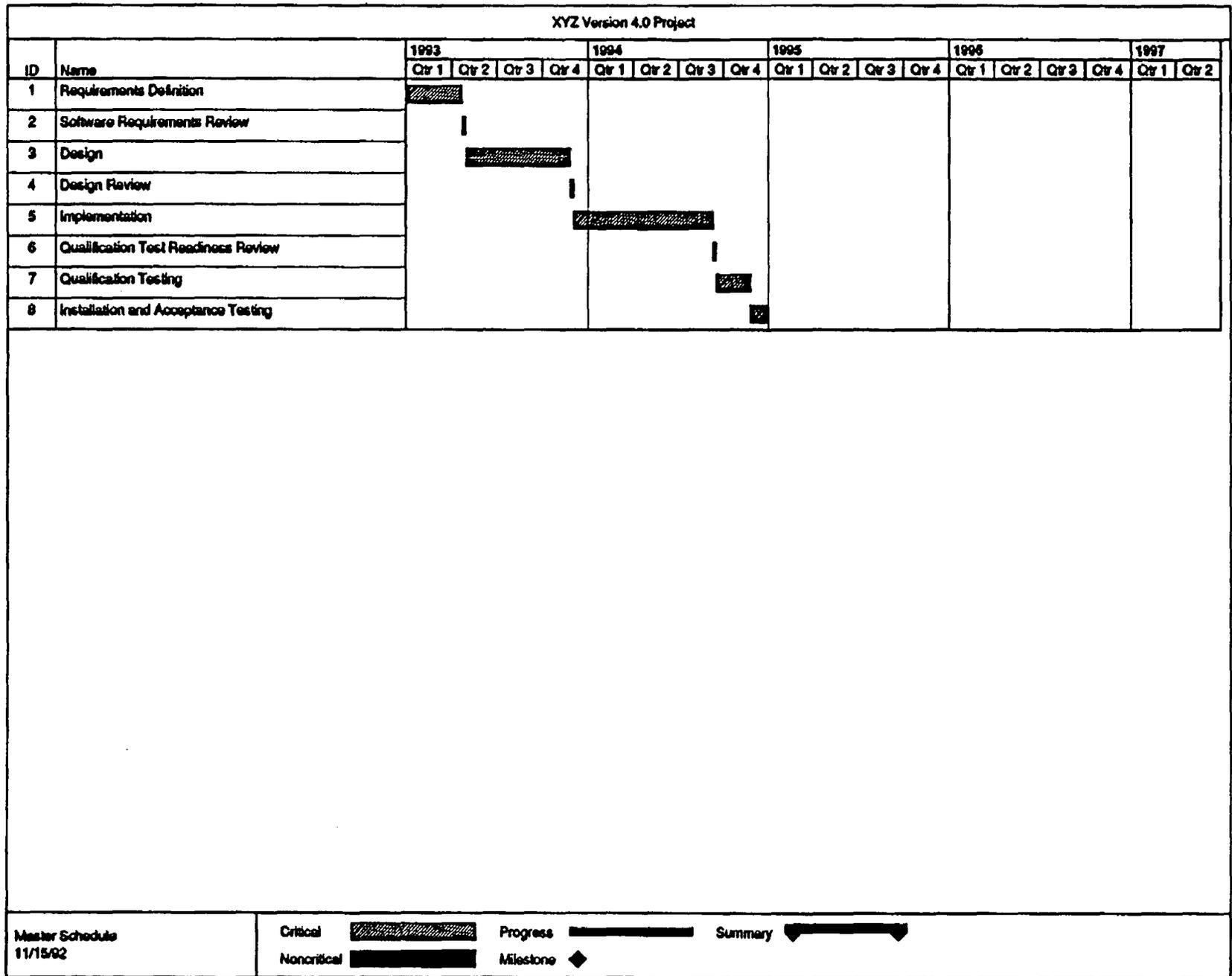


Figure 2-1. XYZ Version 4.0 Project Master Schedule

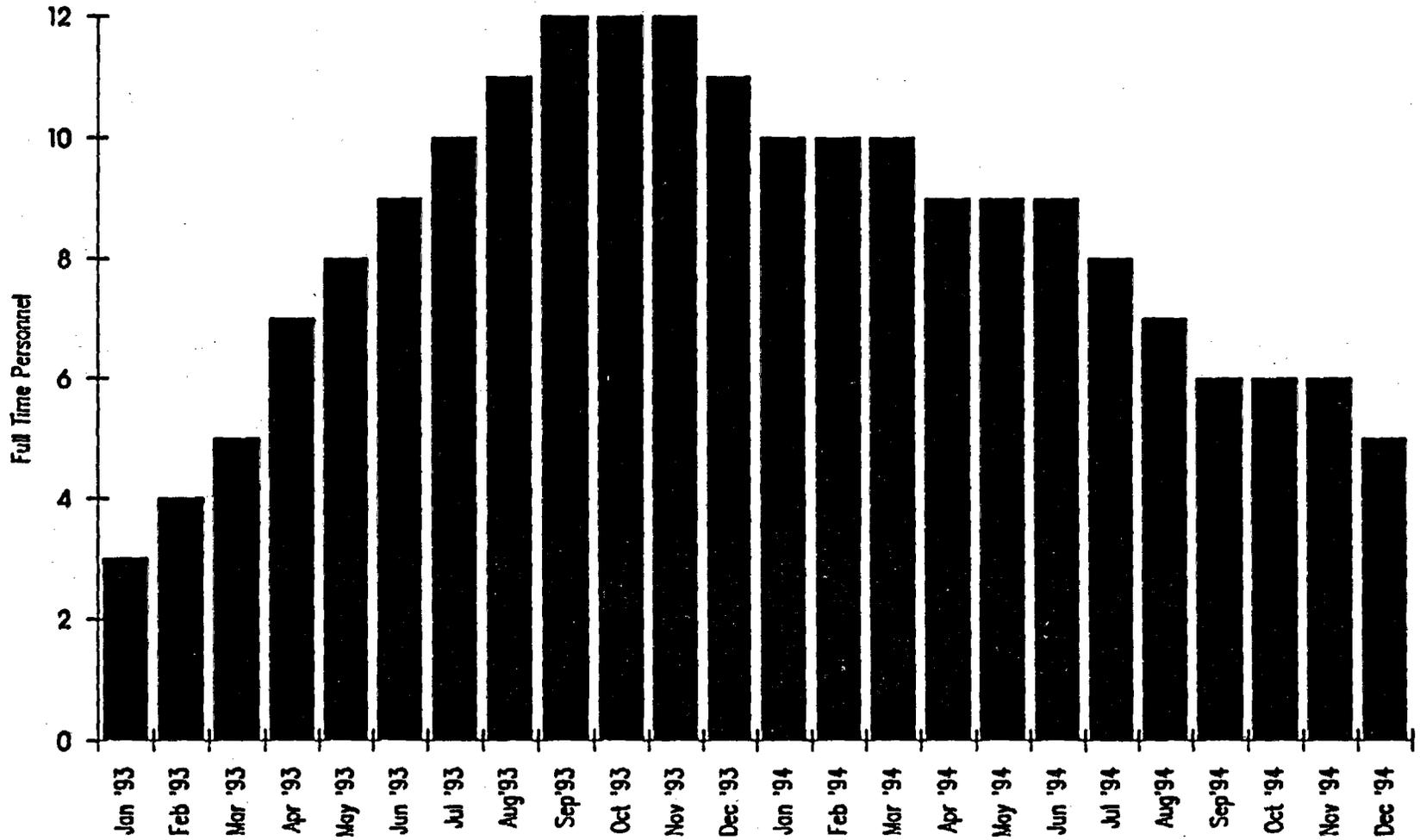


Figure 2-2. Staffing Profile for the XYZ Version 4.0 Project

2.6 Configuration Management

2.6.1 Overview

ABC will support the NRC in the four major functions of configuration management:

1. The identification and establishment of baselines
2. Controlling both changes to baselines and the release of baselines
3. Recording and reporting the status of baselines, change requests, and implemented changes
4. Verifying, through auditing, the correctness and completeness of baselines prior their release

ABC will ensure that:

1. The content and status of the software and documentation baselines are known at all times
2. Configuration management is implemented for externally-deliverable products and for appropriate products used inside the ABC Project
3. There will be a repository for storing key software engineering elements and associated configuration management records
4. The software baselines and configuration management activities are audited on a regular basis
5. The Qualification Testing and Configuration Management Group will be responsible for coordinating configuration management for the project
6. Adequate resources and budget for performing configuration management activities have been allocated
7. Staff members responsible for coordination of configuration management activities have been trained in the objectives, procedures, and methods of performing their duties

2.6.2 Baselines

ABC will establish controlled and stable baselines for planning, managing, and building Version 4.0 of XYZ.

ABC will establish a naming or labeling system that:

1. Uniquely identifies all project entities (e.g., documents, software elements, test cases)
2. Identifies changes by revision or version
3. Uniquely identifies each configuration/version of revised software for use

We will support NRC in the establishment of the following baselines that will be controlled by the NRC XYZ configuration control board (CCB):

1. The project management baseline consisting of the Software Project Plan, documented standards and procedures, and up-to-date budgets and schedules
2. The requirements baseline consisting of the software requirements documentation plus approved changes
3. The product baseline consisting of software and documentation resulting from the qualification testing major activity
4. The operational baseline consisting of software and documentation, resulting from the installation and acceptance activity, that is placed into operation

The XYZ CCB is controlled by the NRC.

ABC will establish and control the developmental configuration, which will contain the software design and implementation products (software design documentation, code, test cases, and related information) of the evolving Version 4.0 of XYZ. We will apply proven ABC internal configuration control procedures and automated tools, used in previous XYZ work, to the developmental configuration as it evolves.

2.6.3 Change Control

ABC will support the NRC in the implementation of configuration control procedures established on previous XYZ upgrade projects. Specifically we will support the NRC in the management of changes to baselines as follows:

1. Implement the directives of the NRC XYZ CCB (XYZ Procedure CM-03)
2. Follow XYZ Procedure CM-02 for initiating, recording, reviewing, approving or rejecting, and tracking requests for changes to baselines
3. Follow XYZ Procedure CM-04 for ensuring that all changes, especially those to the requirements and

the top-level design, are appropriately reviewed for "ripple" effects and incorporated into all related activities

4. Update, gain approval, and follow XYZ Procedure CM-05, to create and control the release of software baseline products

2.6.4 Status of Baselines and Changes

ABC will track the current status of baselines and changes throughout development and maintenance by:

1. Following XYZ Procedure CM-06 to record and report the status of baselines and change requests
2. Creating and distributing to affected groups and individuals standard reports, in accordance with XYZ Procedure CM-06, documenting the configuration management activities

2.6.5 Software Development Library

ABC will continue to maintain the XYZ software development library (SDL) in accordance with XYZ Procedure CM-01. This procedure will be updated if required.

2.6.6 Software, Media, and Access Control

ABC will maintain the facilities and the associated XYZ Procedure CM-01 used to maintain, store, secure, and document controlled versions of the software throughout the life cycle.

ABC will establish and maintain the facilities and procedures used to

1. Identify the media for each software product and the documentation required to store the media
2. Protect software physical media from unauthorized access and inadvertent damage or degradation throughout the life cycle.

2.6.7 Configuration Audits

ABC will follow existing XYZ configuration management procedures to prepare for, conduct, report results from, and track action items based on results of documentation audits, software configuration audits, and in-process audits.

ABC will conduct documentation audits on updates to all documents in accordance with XYZ Procedure CM-07.

Software configuration audits will be performed at the conclusion of both qualification testing and acceptance testing in accordance with XYZ Procedure CM-08.

ABC will conduct in-process audits at least once a year to assess how well the configuration management standards and procedures are being followed and how effective they are in managing the software baselines. In-process audits will be conducted in accordance with XYZ Procedure CM-09.

2.6.8 Techniques and Tools

ABC will continue to use the dBASE III XYZ Nonconformance and Corrective Action System on an industry-standard Personal Computer workstation to track nonconformances, action items, and their resolutions. In addition, the commercially available DDD software tool will be used in the management of the software development library.

SECTION 3 -- TECHNICAL APPROACH

Section 3 responds to the following eight technical tasks of the statement of work:

Task 1—Requirements Definition (Section 3.1.1)

Task 2—Design (Section 3.1.2)

Task 3—Implementation (Section 3.1.3)

Task 4—Qualification Testing (Section 3.1.4)

Task 5—Installation and Acceptance (Section 3.1.5)

Task 6—Verification and Validation (Section 3.2)

Task 9—Nonconformance Reporting And Corrective Action (Section 3.3)

Task 10—Quality Assessment and Improvement Program (Section 3.4)

Section 3.5 identifies the deliverables, their due dates, and the standards that they will follow, and Section 3.6 identifies the standards and procedures that will be used..

3.1 Implementing the Life-Cycle Tasks of the Statement of Work

The statement of work calls for the Contractor to perform the following life-cycle major activities:

1. Requirements Definition
2. Design
3. Implementation
4. Qualification Testing
5. Installation and Acceptance

3.1.1 Task 1—Requirements Definition

ABC will analyze the CI and C2 Requirements Document provided by NRC. We will use ABC's structured analysis approach (use of data flow diagrams, data, dictionaries, and mini-specifications) to perform the requirements analyses as documented in ABC Standards SA-01, SA-02, and SA-03. ABC will provide suggested changes to the CI and C2 Requirements Document and will ensure that the requirements are correct, complete, verifiable, consistent with XYZ Version 3.4 requirements, and technically feasible.

When NRC approves the requirements after the Software Requirements Review, the approved requirements will form the basis for the software plans, products, and activities.

ABC will ensure that the documented requirements define the response of the software to anticipated classes of input data (including erroneous data) and provide the information and detail necessary to design the software (e.g., mathematical models, equations, data requirements).

ABC will perform verification and validation planning activities in parallel with requirements analysis and definition activities.

Requirements changes will be controlled throughout the development and maintenance efforts in accordance with the proven configuration management procedures cited in Section 2.6 of this plan.

3.1.2 Task 2—Design

ABC will use structured design techniques as documented in ABC Standard SD-01 to analyze, both individually and collectively:

1. The baselined requirements for the C1 and C2 capabilities
2. The requirements of Version 3.4 of XYZ
3. The requirements associated with nonconformances to Version 3.4 of XYZ

This analysis will be conducted to determine the optimal software design for Version 4.0 of XYZ such that:

1. Changes to the existing XYZ software architecture are minimized
2. Existing XYZ capability remains operable
3. Changes to the design meet the requirements associated with
 - a. The C1 and C2 capabilities
 - b. Corrective actions to all nonconformances

As the design evolves, events (e.g., identification of additional nonconformances) will necessitate the modification of the requirements and design documentation. ABC will manage changes to formally baselined

(requirements) documentation and internally baselined (design) documentation in accordance with the procedures cited in Section 2.6 of this plan.

3.1.3 Task 3—Implementation

ABC will design and code all new software units and make changes to existing software units in accordance with XYZ Standard IM-01, which defines XYZ Project coding standards. All new and modified units will undergo three inspections:

1. An inspection of the unit design and unit test plan in accordance with ABC Procedure INS-01
2. An inspection of the unit code in accordance with ABC Procedure INS-02
3. An inspection of the unit test results in accordance with ABC Procedure INS-03

ABC will conduct integration tests to ensure that all interfaces among the new and changed units perform correctly. ABC will inspect all integration test plans and procedures in accordance with ABC Procedure INS-04 and integration test reports in accordance with ABC Procedure INS-05.

As the software is implemented, events (e.g., additional insight into data flow patterns) may necessitate the modification of the design, requirements, and/or verification and validation documentation. ABC will manage changes to documentation in accordance with well-defined change control procedures.

3.1.4 Task 4—Qualification Testing

ABC will perform formal qualification testing at ABC's Windy Canyon site using a set of test cases based on:

1. The existing baseline qualification test cases for Version 3.4 of XYZ
2. New test cases that will qualify the new capabilities, C1 and C2

3. New or modified test cases that will qualify the corrective actions to the major nonconformances that will be incorporated into Version 4.0

Qualification tests will be witnessed by a member of the Quality Evaluation and Improvement Group. ABC will review and analyze the qualification test results to assure NRC that the implemented software meets requirements and that the software produces correct results for all approved test cases.

3.1.5 Task 5—Installation And Acceptance

After qualification testing has been successfully concluded, ABC will install Version 4.0 of XYZ at NRC's Rockville, Maryland, headquarters. Then, with an NRC sponsor representative present, ABC will conduct the acceptance tests, which will be a subset of the qualification tests.

3.2 Verification and Validation Approach

This section:

1. Summarizes the verification and validation activities that ABC plans to perform (Section 3.2.1)
2. Discusses the formal life cycle reviews and audits that ABC will conduct (Section 3.2.2)
3. Discusses the formal peer inspections that ABC will use (Section 3.2.3)
4. Identifies the levels of testing that ABC will conduct (Section 3.2.4).

3.2.1 Summary of Verification and Validation Activities

Table 3-1 summarizes the verification and validation activities that ABC plans to perform on the XYZ System Upgrade Project.

**Table 3.1
Verification and Validation Activities by Major Life Cycle Activity**

Major Life Cycle Activity	Verification and Validation Activities
Requirements Definition	<ul style="list-style-type: none"> • Inspect requirements • Conduct the Software Requirements Review
Design	<ul style="list-style-type: none"> • Inspect design • Develop qualification test plan • Develop acceptance test plan • Conduct the Design Review
Implementation	<ul style="list-style-type: none"> • Develop unit test plans • Inspect unit designs, unit code, and unit test plans • Perform unit testing • Inspect unit test results • Develop integration test plans • Inspect integration test plans and procedures • Perform integration testing • Inspect integration test results • Develop qualification test procedures
Qualification Testing	<ul style="list-style-type: none"> • Perform qualification testing • Witness qualification testing (by independent group) • Write qualification test report • Develop acceptance test procedures
Installation and Acceptance	<ul style="list-style-type: none"> • Perform acceptance testing • Witness acceptance testing (by NRC sponsor) • Write acceptance test report • Participate in the Post Mortem Review

3.2.2 Formal Life Cycle Reviews and Audits

A formal review, with NRC and ABC management and technical personnel participating, will be held at or near the end of each major activity of the life cycle. The objective of the formal reviews is to evaluate the deliverable products, the progress, and to a lesser degree, the processes of the most recent life-cycle phase. Table 3-2 summarizes the formal major life cycle reviews and audits that will be performed on the contract.

ABC will deliver five classes of deliverables associated with each formal review:

1. The documents to be reviewed
2. The agenda for the review
3. The hardcopy presentation materials used at the review
4. The minutes that document the activities and results of the review
5. The updated documents that were reviewed

**Table 3.2
Formal Life Cycle Reviews and Audits**

Major Life Cycle Activity	Formal Reviews and Audits
Requirements Definition	<ul style="list-style-type: none"> • Software Requirements Review
Design	<ul style="list-style-type: none"> • Design Review
Implementation	<ul style="list-style-type: none"> • Qualification Test Readiness Review
Qualification Testing	<ul style="list-style-type: none"> • Software Configuration Audit
Installation and Acceptance	<ul style="list-style-type: none"> • Software Configuration Audit • Post Mortem Review

For each formal review, ABC will:

1. Deliver documents for NRC review 2 weeks prior to the start date of the formal review
2. Identify in each review agenda the ABC review participants and their specific responsibilities during the review
3. Assign a person to capture key discussion items and actions items, especially those related to specific assignments for updating the documentation that is the object of the review.
4. Document in the review minutes all proposed revisions to the reviewed documents and all actual changes to the reviewed documents, and place the updated documents under configuration control after approval by the NRC

The paragraphs below discuss each formal life cycle review and audit.

3.2.2.1 Software Requirements Review

ABC will conduct the Software Requirements Review at the end of requirements definition activity. The objectives of this review are to:

1. Review the requirements associated with the new capabilities C1 and C2 and the known nonconformances
2. Review ABC's suggested changes to the draft requirements specification supplied by NRC

3. Assure NRC that ABC understands and agrees on the intent, completeness, verifiability (through testing or other means), consistency, and technical feasibility of the requirements

4. Review the Software Project Plan

3.2.2.2 Design Review

Because the architecture for XYZ is well understood by both NRC and ABC, there will be only one design review. The objectives of the Design Review are to ensure that:

1. The proposed design is complete (meets all the requirements and design completion criteria), verifiable (through testing or other means), consistent, and technically feasible
2. All new and modified software units have been identified and all interfaces between and among the units have defined
3. All elements of the database have been defined down to the data item level
4. Qualification and acceptance test plans are reviewed and the test environment is ready to meet project needs

3.2.2.3 Qualification Test Readiness Review

ABC will conduct the Qualification Test Readiness Review when integration testing has been successfully completed and the qualification test procedures are ready for NRC review. The objective of this review is to assure that the as-built software; the software documentation; and qualification test environment is ready for formal

qualification testing. In particular, the thoroughness of informal (unit and integration) testing will be reviewed.

3.2.2.4 Software Configuration Audits

ABC will conduct two Software Configuration Audits: the first at the completion of qualification testing and second at the completion of acceptance testing. The objective of this audit is to ensure that the as-built software:

1. Meets its requirements as baselined in the software requirements documentation
2. Conforms to its technical documentation
3. Does not contain any unauthorized changes

3.2.2.5 Post Mortem Review

ABC will support the NRC in the Post Mortem Review after the software is accepted. This review will capture the lessons learned from the ABC XYZ System Upgrade Project for use by future XYZ and other similar projects.

3.2.3 Formal Peer Inspections

Because XYZ is Level 1 software, ABC will:

1. Subject each intermediate product and final product of development and maintenance (i.e., all documentation, all code) to an internal peer inspection
2. Make available to the NRC the written procedure and the product standards that govern peer inspections
3. Make available, if requested by the NRC, records that document the results of all peer inspections

3.2.4 Testing

ABC will use four levels of testing:

1. Unit testing
2. Integration testing

3. Qualification testing

4. Acceptance testing

Each new and modified software unit will be separately unit-tested. In unit testing, all paths through the code will be tested. Software components that contain one or more new or modified software units will be integration-tested. Timing of critical elements of code will be tested at the unit or integration level, as appropriate.

Qualification testing and acceptance testing were discussed in Sections 3.1.4 and 3.1.5, respectively.

3.2.5 Software Test Environment

The XYZ software test environment that was baselined in July 1992 will be used on the project.

3.3 Nonconformance Reporting and Corrective Action Approach

ABC will use the in-place XYZ Project nonconformance reporting and corrective action system, baselined by the XYZ CCB on April 12, 1992, and documented in XYZ Procedure NRCA-01.

3.4 Quality Assessment and Improvement Approach

ABC will use the quality assessment and improvement approach documented in the ABC report Continuous Improvement at the ABC Corporation, March 1991.

3.5 Deliverables

Contract deliverables are listed in Table 3-2. All documentation will be inspected in accordance with ABC Procedure INS-12

3.6 Standards and Procedures

Table 3-3 summarizes the standards and procedures that will be used on the contract. Documentation standards for the principal software documents are not listed; the de facto documentation format standards of the existing software documentation will be used.

**Table 3.3
Contract Deliverables**

ID	Deliverable	Due Date (Weeks after Contract Award)
1.1	Draft Suggested Changes to the XYZ Requirements	12 weeks
1.2	Agenda for the Software Requirements Review	12 weeks
1.3	Hardcopy Presentation Materials for the Software Requirements Review	12 weeks
1.4	Minutes for the Software Requirements Review	15 weeks
1.5	Final Suggested Changes to the XYZ Requirements Specification	16 weeks
2.1	Draft Suggested Changes to the XYZ Design Document	46 weeks
2.2	Draft Qualification Test Plan	46 weeks
2.3	Draft Acceptance Test Plan	46 weeks
2.4	Agenda for the Design Review	46 weeks
2.5	Hardcopy Presentation Materials for the Design Review	46 weeks
2.6	Minutes for the Design Review	49 weeks
2.7	Final Suggested Changes to the XYZ Design Document	50 weeks
2.8	Final Qualification Test Plan	50 weeks
2.9	Final Acceptance Test Plan	50 weeks
3.1	Draft Qualification Test Procedures	87 weeks
3.2	Draft Acceptance Test Procedures	87 weeks
3.2	Agenda for the Qualification Test Readiness Review	87 weeks
3.3	Hardcopy Presentation Materials for the Qualification Test Readiness Review	87 weeks
3.4	Minutes for the Qualification Test Readiness Review	89 weeks
3.5	Final Qualification Test Procedures	90 weeks
3.6	Final Acceptance Test Procedures	90 weeks
4.1	Qualification Test Report	98 weeks
4.2	Software Configuration Audit Report 1	99 weeks
5.1	Acceptance Test Report	101 weeks
5.2	Software Configuration Audit Report 2	102 weeks
7.1	Monthly Progress Reports	Monthly
9.1	Nonconformance Reporting And Corrective Action System Reports	As generated

**Table 3.4
Standards and Procedures**

Identification	Title
XYZ Procedure CM-01	XYZ Software Development Library Procedures
XYZ Procedure CM-02	XYZ Change Request Procedure
XYZ Procedure CM-03	XYZ Configuration Control Board Procedures
XYZ Procedure CM-04	XYZ Change Request Ripple Effects Procedures
XYZ Procedure CM-05	XYZ Baseline Release Procedure
XYZ Procedure CM-06	XYZ Status Reporting Procedures
XYZ Procedure CM-07	XYZ Documentation Auditing Procedure
XYZ Procedure CM-08	XYZ Software Configuration Audit Procedure
XYZ Procedure CM-09	XYZ In-Process Configuration Management Auditing Procedure
XYZ Standard IM-01	XYZ Coding Standard
XYZ Procedure NRCA-01	XYZ Nonconformance Reporting and Corrective Action Procedures
ABC Standard SA-01	ABC Data Flow Diagram Standard
ABC Standard SA-02	ABC Data Dictionary Standard
ABC Standard SA-03	ABC Minispecification Standard
ABC Standard SD-01	ABC Structured Design Standard
ABC Procedure INS-01	ABC Unit Design and Unit Test Plan Inspection Procedure
ABC Procedure INS-02	ABC Unit Code Inspection Procedure
ABC Procedure INS-03	ABC Unit Test Results Inspection Procedure
ABC Procedure INS-04	ABC Integration Test Plan and Test Procedure Inspection Procedure
ABC Procedure INS-05	ABC Integration Test Results Inspection Procedure
ABC Procedure INS-12	ABC Documentation Inspection Procedure

APPENDIX B

Glossary

adaptive maintenance. Maintenance performed to make a software product usable in a changed environment.

baseline. A product (software or documentation or both) that has been formally reviewed and agreed upon by the developer and sponsor, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

requirements baseline. The baselined documentation that specifies the requirements that a software product must meet.

product baseline. The software and documentation that are baselined at the successful completion of qualification testing.

operational baseline. The software and documentation that are 1) baselined at the successful completion of installation and acceptance testing and 2) are placed into an operational status as a production product.

change control. The process of evaluating, approving or disapproving, and coordinating changes to baselines. Also called configuration control.

code. One or more computer programs or part of a computer program.

configuration control. See change control.

configuration management. The process of 1) identifying and defining the baselines associated with a software product; 2) controlling the changes to baselines and release of baselines throughout the life cycle; 3) recording and reporting the status of baselines and the proposed and actual changes to the baselines; and 4) verifying the correctness and completeness of baselines.

corrective action. General name for the process by which nonconformances are corrected, verified, and controlled.

corrective maintenance. Maintenance performed specifically to overcome existing faults.

developer. The organization, usually a contractor, that develops or maintains the software.

developmental configuration. The developer's software and associated technical documentation that defines the evolving software products during development. The developmental configuration is under the developer's internal configuration control and contains the software design and implementation products (software design documentation, code, test cases, and related information).

error. A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

formal testing. The process of conducting testing activities and reporting results in accordance with an approved test plan.

independent verification and validation (IV&V). Verification and validation by an organization that is both technically and managerially separate from the organization responsible for developing the software. See verification. See validation.

informal testing. The process of conducting testing activities without an approved test plan.

Level 1 software. Technical application software used in a safety decision by the NRC.

Level 2 software. Technical or non-technical application software not used in a safety decision by the NRC.

Level 3 software. Technical or non-technical application software not used in a safety decision by the NRC and having local or limited use by the NRC.

nonconformance. Any failure of any software document, code, data structure, or process, to meet its requirements or standards. Often called a problem, discrepancy, fault, or error.

non-developmental software. Deliverable software that is not developed under the contract but is provided by the developer, the Government, or a third party. Non-developmental software may be referred to as reusable software, Government-furnished software, or commercially available software depending on its source.

qualification testing. A process that allows the sponsor to determine whether a software product complies with its requirements. quality assurance. A planned and systematic pattern of all actions necessary to provide adequate confidence that a software product conforms to established technical requirements.

release. A configuration management action whereby a particular version of software is made available for a specific purpose (e.g., released for test, released to operations)

reusable software. Software developed in response to the requirements for one application that can be used, in whole or in part, to satisfy the requirements of another application.

software engineering environment. The set of automated tools, firmware devices, and hardware necessary to perform the software engineering effort, including establishing and maintaining the software development library. The automated tools may include but are not limited to computer-aided software engineering (CASE) tools, compilers, assemblers, linkers, loaders, operating system, debuggers, simulators, emulators, test tools, documentation tools, and data base management systems.

software development library. A controlled collection of software, documentation, and associated tools and procedures used to facilitate the orderly development and subsequent maintenance of software. The software development library contains the developmental configuration as part of its contents. A software development library provides storage of and controlled access to software and documentation in human-readable form, machine readable form, or both. The software development library may also contain management data pertinent to the software development project.

software life cycle. The period of time that starts when a software product is conceived and ends when the product is retired from use.

software maintenance. Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.

software plans. A collective term used to describe the developer's plans, methodologies, standards, and procedures for software management, software engineering, verification and validation, documentation, product evaluation, and configuration management. software test environment. The set of automated

tools, firmware devices, and hardware necessary to test software. The automated tools may include but are not limited to test tools such as simulation software, code analyzers, etc. and may also include those tools used in the software engineering environment.

software unit. An element of the software design that can be compiled or assembled and is relatively small (e.g., 100 lines of high-order language code).

sponsor. The NRC organization that sponsors and manages the software development/maintenance effort. The sponsor acts as the acquirer or buyer for the user.

sustaining engineering. The process that includes software maintenance and the software engineering activities that ensure that the integrity of the software's original requirements set and design are retained.

testing. The process of exercising or evaluating a software product or part of a software product by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results.

test case. A specific set of test data and associated procedures developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

test plan. A document prescribing the approach to be taken for intended testing activities. The plan typically identifies the items to be tested, the requirements being tested, the testing to be performed, test schedules, personnel requirements, reporting requirements, evaluation criteria, any risks requiring contingency planning.

test procedure. Detailed instructions for the setup, operation, and evaluation of results for a given test. A set of associated procedures is often combined to form a test procedures document.

test report. A document describing the conduct and results of the testing of a software product or a component of a software product.

user. The organization or persons who will use the software product being developed.

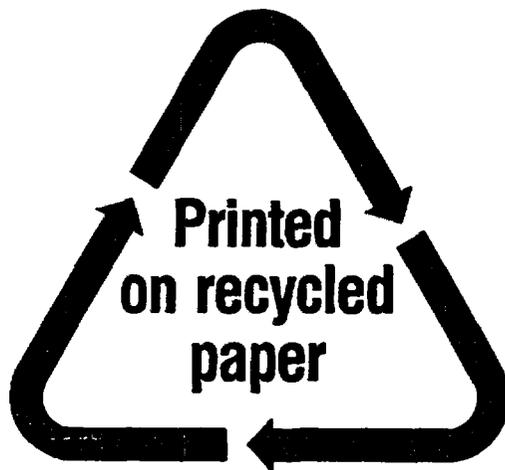
verification. The process of determining whether or not the products of a given activity or phase of the software development life cycle meets its requirements.

validation. The process of evaluating a software product at the end of the software development process to ensure compliance with software requirements.

APPENDIX C

Reference Documents

1. ANSI/ASME NQA-1-1983, Quality Assurance Program Requirements for Nuclear Facilities
2. ANSI/ASME NQA-2a-1990 Addenda (Part 2.7) to ASME NQA-2a- 1989 Edition Quality Assurance Requirements for Nuclear Facility Applications
3. ANSI/IEEE Std 730-1989, IEEE Standard for Software Quality Assurance Plans
4. ANSI/IEEE Std 983-1986, IEEE Guide for Software Quality Assurance Planning
5. ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology
6. ANSI/IEEE Std 828-1983, IEEE Standard for Software Configuration Management Plans
7. ANSI/IEEE Std 829-1983, IEEE Standard for Software Test Documentation
8. ANSI/IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans
9. DOD-STD-2167A, Military Standard, Defense System Software Development, 29 February 1988
10. DOD-STD-2168, Military Standard, Defense System Software Quality Program, 29 April 1992.
11. DOD-STD-1521B, Military Standard, Technical Reviews and Audits for Systems, Equipments, and Computer Software, 4 June 1985.
12. Capability Maturity Model for Software, CMU/SEI-91-TR-24, Software Engineering Institute, Carnegie Mellon University, August 1991



Federal Recycling Program

UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, D.C. 20555-0001

FIRST CLASS MAIL
POSTAGE AND FEES PAID
USNRC
PERMIT NO. G-67

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, \$300