NIST Special Publication 500–223

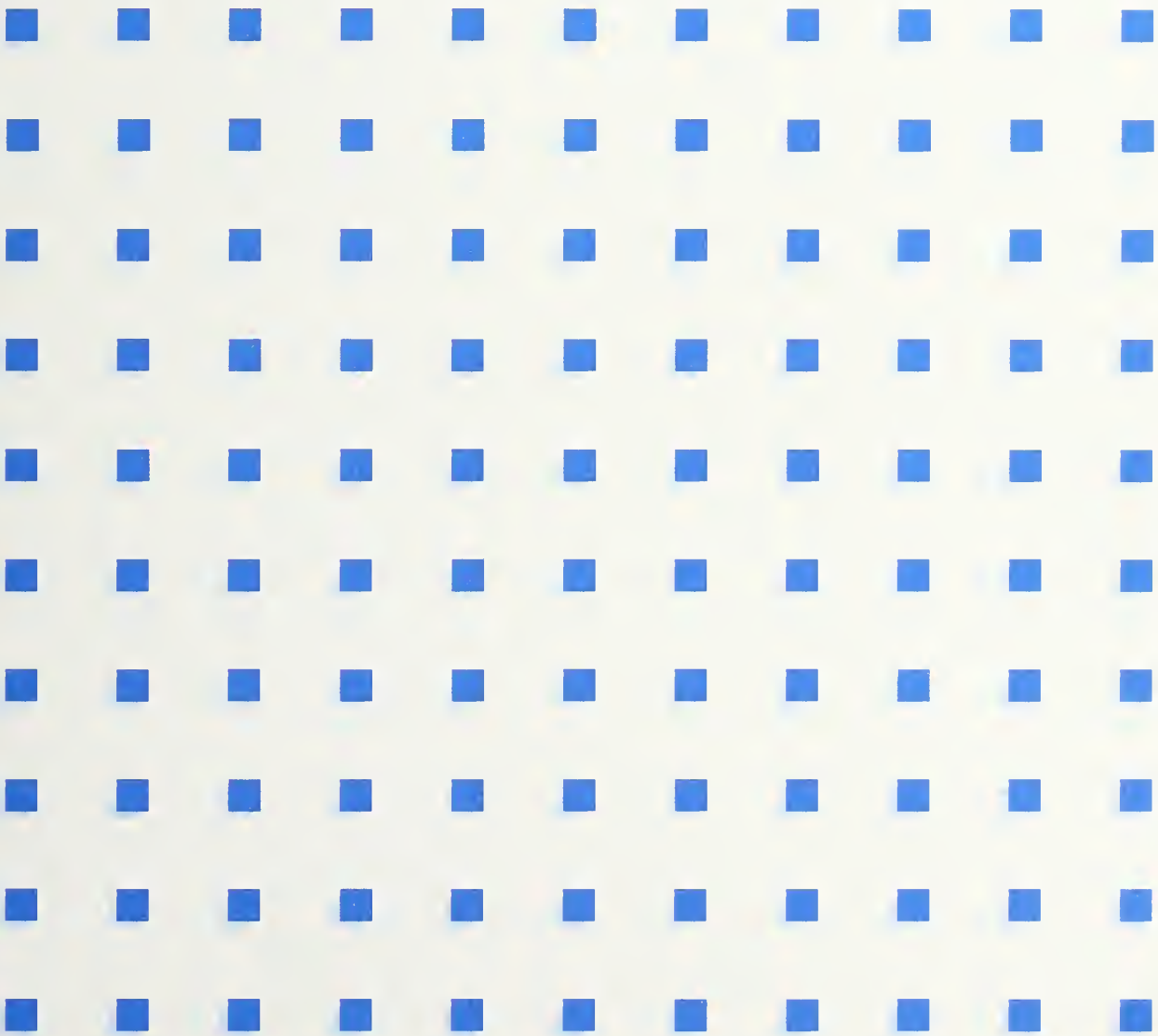
Computer Systems Technology

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology

NIST

A Framework for the Development and Assurance of High Integrity Software

Dolores R. Wallace
Laura M. Ippolito



The National Institute of Standards and Technology was established in 1988 by Congress to "assist industry in the development of technology . . . needed to improve product quality, to modernize manufacturing processes, to ensure product reliability . . . and to facilitate rapid commercialization . . . of products based on new scientific discoveries."

NIST, originally founded as the National Bureau of Standards in 1901, works to strengthen U.S. industry's competitiveness; advance science and engineering; and improve public health, safety, and the environment. One of the agency's basic functions is to develop, maintain, and retain custody of the national standards of measurement, and provide the means and methods for comparing standards used in science, engineering, manufacturing, commerce, industry, and education with the standards adopted or recognized by the Federal Government.

As an agency of the U.S. Commerce Department's Technology Administration, NIST conducts basic and applied research in the physical sciences and engineering, and develops measurement techniques, test methods, standards, and related services. The Institute does generic and precompetitive work on new and advanced technologies. NIST's research facilities are located at Gaithersburg, MD 20899, and at Boulder, CO 80303. Major technical operating units and their principal activities are listed below. For more information contact the Public Inquiries Desk, 301-975-3058.

Office of the Director
- Advanced Technology Program
- Quality Programs
- International and Academic Affairs

Technology Services
- Manufacturing Extension Partnership
- Standards Services
- Technology Commercialization
- Measurement Services
- Technology Evaluation and Assessment
- Information Services

Materials Science and Engineering Laboratory
- Intelligent Processing of Materials
- Ceramics
- Materials Reliability[1]
- Polymers
- Metallurgy
- Reactor Radiation

Chemical Science and Technology Laboratory
- Biotechnology
- Chemical Kinetics and Thermodynamics
- Analytical Chemical Research
- Process Measurements[2]
- Surface and Microanalysis Science
- Thermophysics[2]

Physics Laboratory
- Electron and Optical Physics
- Atomic Physics
- Molecular Physics
- Radiometric Physics
- Quantum Metrology
- Ionizing Radiation
- Time and Frequency[1]
- Quantum Physics[1]

Manufacturing Engineering Laboratory
- Precision Engineering
- Automated Production Technology
- Intelligent Systems
- Manufacturing Systems Integration
- Fabrication Technology

Electronics and Electrical Engineering Laboratory
- Microelectronics
- Law Enforcement Standards
- Electricity
- Semiconductor Electronics
- Electromagnetic Fields[1]
- Electromagnetic Technology[1]
- Optoelectronics[1]

Building and Fire Research Laboratory
- Structures
- Building Materials
- Building Environment
- Fire Safety
- Fire Science

Computer Systems Laboratory
- Office of Enterprise Integration
- Information Systems Engineering
- Systems and Software Technology
- Computer Security
- Systems and Network Architecture
- Advanced Systems

Computing and Applied Mathematics Laboratory
- Applied and Computational Mathematics[2]
- Statistical Engineering[2]
- Scientific Computing Environments[2]
- Computer Services
- Computer Systems and Communications[2]
- Information Systems

[1] At Boulder, CO 80303.
[2] Some elements at Boulder, CO 80303.

NIST Special Publication 500–223

A Framework for the Development and Assurance of High Integrity Software

Dolores R. Wallace
Laura M. Ippolito

Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-0001

December 1994

U.S. Department of Commerce
Ronald H. Brown, Secretary

Technology Administration
Mary L. Good, Under Secretary for Technology

National Institute of Standards and Technology
Arati Prabhakar, Director

Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) has a unique responsibility for computer systems technology within the Federal government. NIST's Computer Systems Laboratory (CSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. CSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. CSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports CSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

National Institute of Standards and Technology Special Publication 500-223
Natl. Inst. Stand. Technol. Spec. Publ. 500-223, 75 pages (Dec. 1994)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1994

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

ABSTRACT

The purpose of this document is to recommend a framework for the development and assurance of high integrity software. The framework addresses the fact that these processes must take into account properties and requirements of a high integrity system and the processes and standards used in developing other system components. This framework provides guidance to developers, assurers, and buyers of software, researchers for high integrity software systems, and vendors of Computer Aided Software Engineering tools and integrated environments.

KEYWORDS

High integrity software, project management, software assurance, software configuration management, software development, software hazard analysis, software quality assurance, software verification and validation.

ACKNOWLEDGMENTS

NIST acknowledges the contributions by SoHaR, Incorporated to section 5 of this report.

EXECUTIVE SUMMARY

The National Institute of Standards and Technology (NIST) has been working for several years on providing information to government, industry, and academia regarding high integrity software. Efforts in this area have included development of guidance on software verification, validation and testing [e.g., NIST165], hosting a workshop on high integrity software (proceedings are documented in [NIST190]), and conducting studies on high integrity standards and guidelines, software quality assurance documentation and reviews, and software error analysis (results of these studies are documented in [NIST204], [NIST4909], and [NIST209], respectively). NIST recognized the need to develop a single document which would address developing and assuring high integrity software. This document provides a technology independent framework to assist government, industry, and academia in addressing the issues of providing software for high integrity software systems. This framework proposes the activities that comprise software development and software assurance processes, independent of the technology used to perform them. Users of the framework may implement these activities with methods which are most appropriate to the software application domain.

This framework is also a starting point on which to initiate the activities supporting the Center for High Integrity Software Systems Assurance (CHISSA) established by NIST. CHISSA will foster and coordinate activities relating to high integrity software technology. It will help guide research in development, analysis, and testing techniques, conduct assessments on software system technology, and provide transfer of those technologies deemed useful to the industrial sector. CHISSA will work in cooperation with other Federal agencies, industry, and the research community to develop standards and guidelines for high integrity software. CHISSA will also address issues concerning the link between software assurance and the systems in which that software is embedded.

This document provides an initial framework for the development and assurance of high integrity software for use by CHISSA and developers, assurers and buyers of software for high integrity software systems, and by Computer Aided Software Engineering (CASE) vendors. This framework addresses two primary and concurrent processes, software development and software assurance, which each consist of several processes. In this document the software development processes are described separately from the software assurance processes although many activities may be occurring concurrently, and perhaps are performed by the same people. The software development processes build the software, while the software assurance processes provide the activities to plan, monitor and assess the software. *This separation of processes in this framework is only for the purpose of identifying the actual activities of each process; all processes contribute to the quality of the software.*

This framework proposes the major objective(s) and a detailed list of activities for each software development and software assurance process. The processes and activities occur regardless of the life cycle or methodologies. Different life cycle approaches (e.g., iteration; incremental development) may affect the choice of methods for performing the processes. The choice of technology (e.g., object-oriented) may affect how the requirement and design processes and activities are performed. In both cases, the processes and activities of this framework are applicable and form the basis for specific methods. The software development processes include

the software requirements process, software design process, code process, software integration process, software installation process, and software operation and maintenance process.

The software assurance processes include the project management process, software quality assurance process, software verification and validation process, software configuration management process, and software hazard analysis process. The software verification and validation process includes independent verification and validation, software requirements verification and validation process, software design verification and validation process, code verification and validation process, unit test process, software integration test process, software system test process, software installation test process, and software operation and maintenance verification and validation process. Complete system validation is outside the scope of this document.

This document provides an initial set of topics for functionality of software that should be considered when specifying software for use in a software-intensive system, especially where high integrity attributes are important. This framework discusses software engineering practices that aid in the development and assurance of high integrity software. It provides a basis from which to identify strengths and weaknesses in current software engineering techniques relative to high integrity software and to indicate where further research is needed.

This framework does not address procurement issues directly, that is, it does not describe processes for the acquirer of software. Its focus is deliberately on the technical engineering processes that are used to build software-intensive systems, and includes those processes for assuring the quality of the resulting system. The framework is a composite of many standards, draft standards, technical reports, journal articles and experience. The terminology in these documents is not consistent; for example, the terms developer and producer are often used to refer to those who provide software. A later version of this framework may include a mapping of principal software life cycle terminology to the most common usages found in other related documents.

As an initial document for CHISSA, on which CHISSA may base some of its activities, this framework will undergo substantive change and expansion. Future work in expanding this framework includes, but is not limited to, the following tasks:

- definition of the interfaces and the related technical problems between software and system
- development of a profile of functionality for high integrity software systems which may be further refined for application domains and may be used to identify specific technical problems
- identification of appropriate software engineering methods (or practices) mapped to application domains or technical problems which those methods resolve
- identification where current methods are inadequate and further research is needed

- examination of types of CASE tools for implementing recommended software engineering methods supporting these activities
- examination of integration capabilities of CASE tools
- definition of a comparable framework for system development and assurance both as an entity and specifically for each system component, followed by similar tasks identified for software.

GLOSSARY

Accuracy. A qualitative assessment of correctness, or freedom from error [IEEE610].

CASE (Computer Aided Software Engineering) tools. Software tools that assist with software design, requirements traceability, code generation, testing and other software engineering activities [IEEE610].

Completeness. The degree to which all of the software's required functions and design constraints are present and fully developed in the software requirements, software design, and code [SOFTENG].

Component. One of the parts that make up a system, some of which may be broken down into more components or units; it may be personnel (e.g., operator, user), procedures, materials, tools, equipment (hardware), facilities, and software [IEEE610, MIL882B].

Consistency. The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component [IEEE610].

Correctness. The degree to which software or its components is free from faults and/or meets specified requirements and/or user needs [IEEE610].

Criticality. The severity of the failure mode.

Debug. To detect, locate, and correct faults in a computer program [IEEE610].

High integrity software. Software that can and must be trusted to work dependably in some critical function, and whose failure to do so may have catastrophic results, such as serious injury, loss of life or property, business failure or breach of security. Some examples include software used in safety systems of nuclear power plants, transportation systems, medical devices, electronic banking, automatic manufacturing, and military systems [NIST204].

Quality attributes. Requirements that software must meet such as usability, efficiency, reliability, maintainability, and portability [NIST4909].

Redundancy. The presence of backup components that perform the same or similar functions as other components [IEEE610].

Risk. A measure derived from the probability of failure occurring and the severity of failure modes.

Software configuration item. An aggregation of software that is treated as a single entity in the software configuration management process [IEEE610].

Software quality assurance. The planned systematic pattern of all actions necessary to provide adequate confidence that the product, or process by which the product is developed, conforms to established requirements [NIST204].

Software verification and validation. See verification and validation.

System. A composite, at any level of complexity, of personnel, procedures, materials, tools, equipment, facilities, and software. The elements of this composite entity are used together in the intended operational or support environment to perform a given task or achieve a specific production, support, or mission requirement [MIL882B].

Testability. The degree to which software or a software component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [IEEE610].

Test case. A set of test inputs, execution conditions, and expected results developed for a particular objective, e.g., to exercise a particular program path [IEEE610].

Test coverage. The extent to which the test cases test the software requirements [ISO12207].

Test design. The test approach and associated tests [IEEE610].

Test procedure. Detailed instructions for the set-up, execution, and evaluation of results for a given test case [IEEE610].

Understandability. The extent to which the meaning of the software requirements, software design, and code are clear to the reader [SOFTENG].

Unit. A separately compilable piece of code [ISO12207].

Validation. The process of evaluating a system or component (including software) during or at the end of the development process to determine whether it satisfies specified requirements [IEEE610].

Verification. The process of evaluating a system or component (including software) to determine whether the products of a given development process satisfy the requirements imposed at the start of that process [IEEE610].

Verification and validation. The process of determining whether the requirements for a system or component (including software) are complete and correct, the products of each development process fulfill the requirements or conditions imposed by the previous process, and the final system or component (including software) complies with specified requirements [IEEE610].

ACRONYMS

CASE	Computer Aided Software Engineering
CHISSA	Center for High Integrity Software Systems Assurance
CI	Configuration Item
CSHA	Code-level Software Hazard Analysis
DBDD	Database Design Description
IV&V	Independent Verification and Validation
NRC	U.S. Nuclear Regulatory Commission
PM	Project Management
PMP	Project Management Plan
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SDD	Software Design Description
SDHA	Software Design Hazard Analysis
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SRHA	Software Requirements Hazard Analysis
SRS	Software Requirements Specification
SV&V	Software Verification and Validation
SVVP	Software Verification and Validation Plan
SVVR	Software Verification and Validation Report

Table of Contents

Tables

Figures

1 INTRODUCTION

High integrity software (e.g., software which must and can be trusted to work dependably and whose failure would have catastrophic results [NIST190]) is a critical factor in all aspects of modern society. It controls a wide range of essential activities including banking and commerce, manufacturing, education, transportation, health care, and entertainment. Currently, the body of knowledge required to build high integrity software is distributed among standards, guidelines, technical reports, conference presentations, and information proprietary to organizations [NIST204]. The National Institute of Standards and Technology (NIST) has been working for several years on providing information to government, industry, and academia regarding high integrity software. Efforts in this area have included development of guidance on software verification, validation and testing [e.g., NIST165], hosting a workshop on high integrity software (proceedings are documented in [NIST190]), and conducting studies on high integrity standards and guidelines, software quality assurance documentation and reviews, and software error analysis (results of these studies are documented in [NIST204], [NIST4909], and [NIST209], respectively).

NIST recognized the need to develop a *single* document which would address developing and assuring high integrity software. This document provides an initial framework to assist government, industry, and academia in addressing the issues of providing software for high integrity software systems. NIST has built a database[1] of standards, guidelines, technical articles, and books that were used to provide a basis for the software development and software assurance activities described in this framework.

This technology independent framework can be used as:

- a guideline for developers and assurers of the software
- an evaluation tool by reviewers of the software
- a basis for identifying software engineering practices (techniques) that satisfy an activity, or aggregates of activities
- a basis for mapping high integrity assurance requirements against the software engineering techniques to identify requirements not satisfied by current software engineering practices
- guidance for developers and assurers in selecting Computer Aided Software Engineering (CASE) tools appropriate for their project
- a basis for identifying interrelationships among the activities, hence aiding the CASE tool integrator.

This framework is also an initial starting point on which to initiate the activities supporting the Center for High Integrity Software Systems Assurance (CHISSA) established by NIST. CHISSA will foster and coordinate activities relating to high integrity software technology. It will help

[1]This database includes current U.S. and international documents that address high integrity software systems.

guide research in development, analysis, and testing techniques, conduct assessments on software system technology, and provide transfer of those technologies deemed useful to the industrial sector. CHISSA will work in cooperation with other Federal agencies, industry, and the research community to develop standards and guidelines for high integrity software. Issues concerning the link between software assurance and the systems in which that software is embedded will be addressed as well.

CHISSA will promote research, development, analysis, testing, and transition of software system technology which will improve the development, maintenance, and operation of software based systems. CHISSA will coordinate activities for:

- identification of high leverage research topics and potentially beneficial research results
- identification of technology issues between software and other system components
- communication between the software community and systems community to identify technologies for use by both communities
- assessment of technology in real application projects
- identification and possibly limited implementation of mechanisms for insertion of technology
- promotion of continuous training for engineers and scientists
- promotion and development of guidance and standards.

1.1 Framework Content

This framework identifies processes for the primary and concurrent processes of development and assurance of high integrity software. The processes and activities occur regardless of the life cycle or methodologies. Different life cycle approaches (e.g., iteration; incremental development) may affect the choice of methods for performing the processes. The choice of technology (e.g., object-oriented) may affect how the requirement and design processes and activities are performed. In both cases, the processes and activities of this framework are applicable and form the basis for specific methods. Software development processes are those processes that are used to construct the software, that is, define the software, design it, implement the design into software code, and integrate the software into the system. Their purpose is to build the software, and make corrections as needed. Each software development process produces outputs which ultimately lead to the final software product which is integrated with other system components and is executed in the installed system.

Software assurance processes are those processes that are used to plan and manage the software development processes, and some, like the project management and software quality assurance processes, also oversee other software assurance processes. Their purpose is to provide assurance that the software will meet its requirements and consequently support the system requirements. Software assurance processes check and analyze the decisions regarding the software and its

relationship to the system, the plans and their implementations, and they analyze and test the software outputs. Software assurance processes are an integral part of software development, that is, their execution occurs concurrently with development processes.

In some instances, software development and assurance processes appear to be intertwined, (e.g., depending on project organization, programming and unit test may appear to occur almost simultaneously.) Yet, the purpose of programming is to build, and the purpose of unit test is to provide checking. The software assurance processes verify step by step that each evolving software output is consistent with its predecessor, and hence meets system requirements. Software assurance processes provide controls to ensure that verified outputs are not replaced by other versions. Other processes (e.g., test) are used to validate that the software meets its requirements. Total validation against the system requirements must be performed as part of system development.

This framework treats the software development and software assurance processes separately to enable better planning and implementation of the activities supporting these processes. *This separation of processes in this framework is only for the purpose of identifying the actual activities of each process; all processes contribute to the quality of the software and may be performed by more than one organization.*

This framework identifies key information about the software and its relationship to other system components. The equivalent information for establishing requirements, designing, building, and validating a system are outside the scope of this document and should be contained in another, similar document. However, the software development and software assurance processes must take into account properties and requirements of the system. Figure 1-1, based on the system framework of [FUJII1, FUJII2], shows the relationship between the system and software development processes. Figure 1-2 shows the relationship between the software development and software assurance processes.[2]

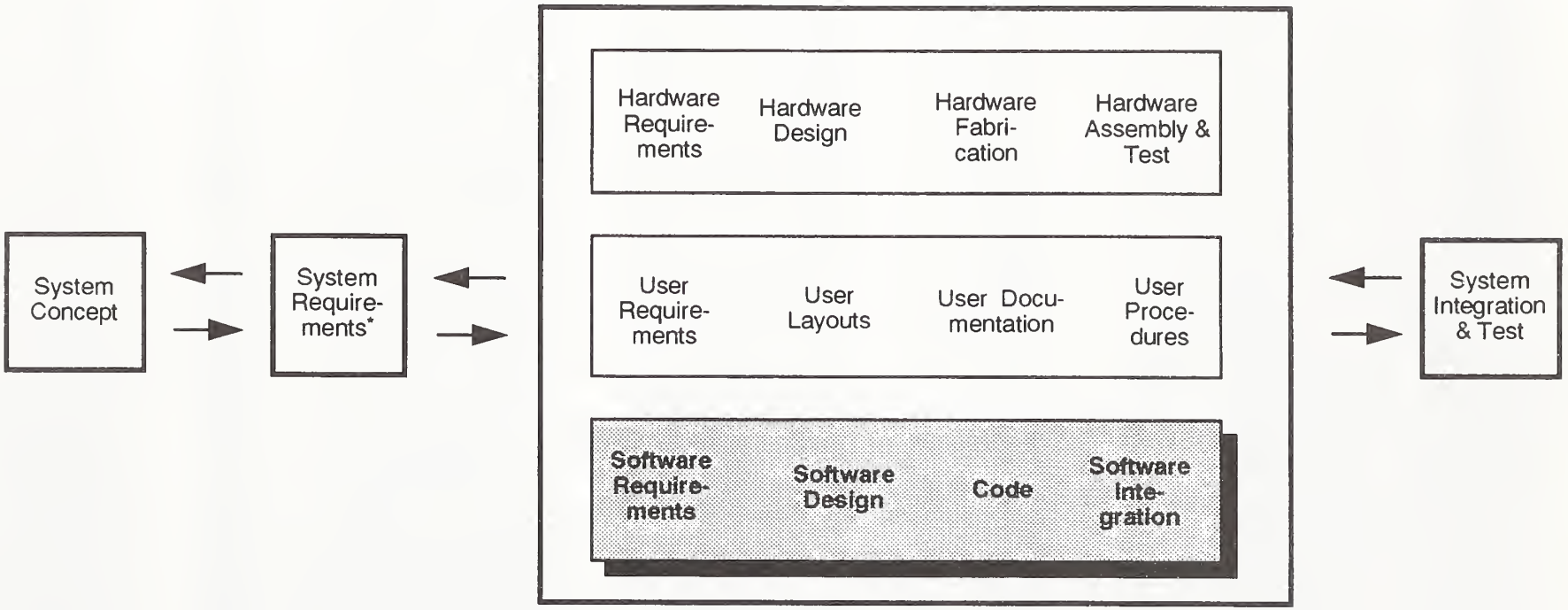
The software development and software assurance processes are iterative[3]. They may also be used to build the resulting system incrementally, that is, portions of the software are developed in an evolutionary manner, with each addition increasing the system capability. These figures accommodate every life cycle methodology; no matter how software is developed all these processes must be achieved for high integrity software assurance. Even with rapid prototyping (where someone defines what is wanted, designs how to build it, builds some part of it or a skeleton of it, and checks that things are going as planned), which may take only an hour, all these processes will have been performed.

The activities of the software development and software assurance processes are implemented with the methods most appropriate to the software application domain. Software quality

---

[2]This framework describes only the software processes which are shaded in figures 1-1 and 1-2. And, while both the software installation process and the software operation and maintenance process involve the entire system, this framework only addresses those process with respect to software.

[3]See section 2.



* Requirements allocation to System Elements (hardware, user, software)

Figure 1-1. Software Development as a Part of System Development

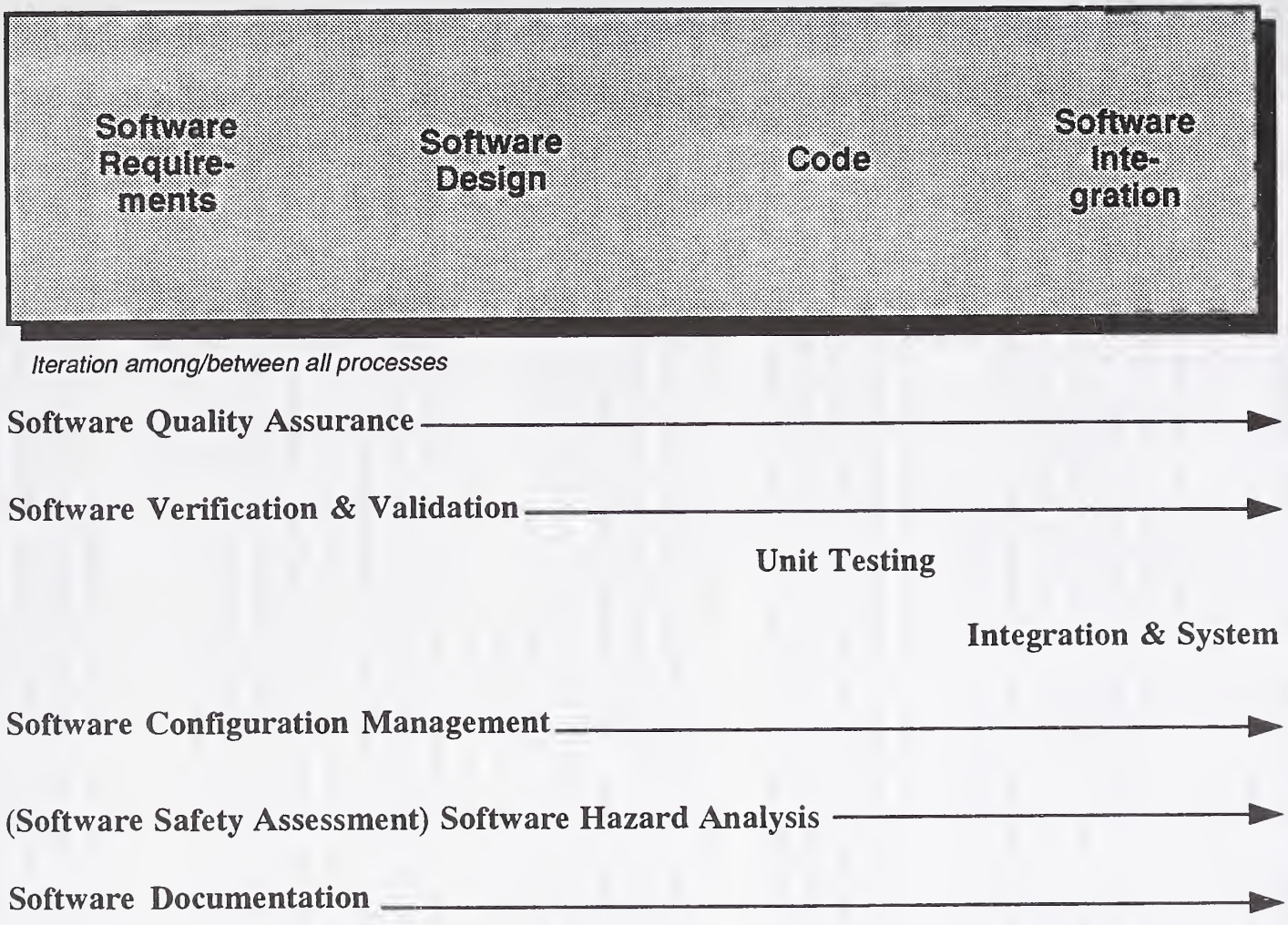


Figure 1-2. Software Assurance Relationship to Software Development

assurance monitors the usage of methods selected, and all software assurance processes monitor software outputs for appropriate results.

This framework does not specify which processes are performed by which organizations or whether or not they must be performed by separate development and assurance organizations. Definitions for independent verification and validation (technical, managerial, and financial independence between the software verification and validation team and the software development team) are provided.

This framework does not address documentation of the software development and assurance activities as a separate process. Instead, each software process addresses recording its activities in documents such as a software requirements specification or software design document. [NIST4909] provides more information on what should be included in each process's document(s). Exactly how documentation should be provided, for specific application domains, with the use of modern technology and new development paradigms is itself a major research topic and hence outside the scope of this framework.

This framework discusses software engineering practices that aid in the development and assurance of high integrity software. It is not intended to be a complete representation of accepted software engineering practices but is a basis for such a representation. It can be used to identify strengths and weaknesses in current software engineering techniques relative to high integrity software, and indicates where further research or improvement is needed.

This framework was originally produced to address activities for developing software systems where *safety* is the predominant issue, which is why the term "hazard" is used. For *security* the more fitting term is "threat." A later version of this document will expand upon computer security activities. One purpose of CHISSA is to identify technology issues between the software and its system. This document provides an initial set of topics for functionality of software that should be considered when specifying software for use in a software-intensive system, especially where high integrity attributes are important.

This framework does not address procurement issues directly, that is, it does not describe processes for the acquirer of software. Its focus is deliberately on the technical engineering processes that are used to build software-intensive systems, and includes those processes for assuring the quality of the resulting system. The framework is a composite of many standards, draft standards, technical reports, journal articles and experience. The terminology in these documents is not consistent; for example, the terms developer and producer are often used to refer to those who provide software. A later version of this framework may include a mapping of principal software life cycle terminology to the most common usages found in other related documents.

Sections 2 and 3 describe the software development and software assurance processes, respectively. Section 4 addresses software engineering practices. Section 5 discusses software functionality for high integrity software systems. Section 6 provides a summary of this framework and a recommendation for further research. Section 7 contains references. Appendix A contains a bibliography of the documents that provided a basis for the processes and activities of this framework.

2 SOFTWARE DEVELOPMENT

The development of high integrity software includes the software requirements process, software design process, code process, software integration process, software installation process, and software operation and maintenance process. These processes inherently include some software assurance activities (e.g., some analysis, some test), but the software assurance processes themselves are described in section 3. This framework does not specify who performs any of the processes, only what needs to be accomplished.

The software development processes are independent of any specific life cycle model, and the concepts described in this section can be applied regardless of life cycle management style. Each process is not necessarily completed before the next process is started. For example, the software may be developed incrementally, i.e., a group of requirements are specified, designed, coded, and tested, and then another group follows the same pattern. The software development processes are also iterative. Software requirements may be added, deleted or altered any time during software development. For example, a new software requirement may be found to be necessary during the software design process or after software hazard analysis has been performed. (Changes in the software requirements or software design may also necessitate a re-classification of the software criticality[4] level.) Modifications to the software requirements in turn affect all subsequent processes. Software assurance processes should be invoked according to the development processes.

For each software development process, this framework provides inputs and outputs (see Table 2-1 at the end of this section for a summary), the major objective(s) of the process, and activities within the process. The following documents were used in compiling this section: [ANS7432], [ANSP7432], [FIPS101], [IEC880], [MIL498], [NIST4909], [NIST204], [RTCA178B], and [SOFTENG].

The outputs of the software development processes can be represented in a variety of ways. For example, a software design description may be a paper document or a graphical representation stored in a computer aided software engineering (CASE) tool or some combination of text and graphics, in paper form or CASE tool repository. A user's manual may actually be part of the software, accessed online using windows and/or menus. This section does not describe the medium used to represent the outputs of a software development process, and will hereafter refer to the representation of the outputs as documentation. This section only includes the relationship of the documentation to each software development process.

2.1 Software Requirements Process

The major objectives of the software requirements process are to fulfill the system and software objectives, develop software requirements based on, and traceable back to, the system requirements, and to provide complete, consistent, correct, testable, and understandable information from which the software may be designed. This process uses the system

[4]Criticality analysis is not addressed in this document except under software requirements verification and validation (section 3.3.2). Later versions may include more detailed activities related to assessing criticality.

requirements (e.g., hardware, mechanical, user interfaces to software) and system design (including the system safety assessment (contains system hazard analysis) and the safety-related and security-related requirements), the initial project management plan (PMP), and software requirements standards identified in the software quality assurance plan (SQAP), to develop the requirements for software. The software requirements encompass functional, performance, interface, safety, security, and quality requirements [NIST180]. The software requirements process ends when its objectives and the objectives of any software assurance processes performed concurrently with it are met. The software requirements process produces a software requirements specification (SRS). A user's manual is also started, but not completed, during this process. Depending on the PMP, this process and other development and assurance processes may produce several outputs before completion of a process, resulting in a final product.

The following are activities of the software requirements process (software assurance activities listed in section 3 are performed concurrently with these activities):

- identify the system and software objectives that the software must meet
- define the measurements that will be used to assess whether the software requirements meet the system and software objectives
- check that each system requirement allocated to software should truly be allocated to software
- check that all system requirements that should be allocated to software have been allocated
- establish mechanism for traceability of system requirements and software requirements and software documentation
- implement the trace mechanism (i.e., execute the procedures that will establish the link between the system requirements and software requirements and software documentation)
- refine definition of each system requirement allocated to software to the level of detail necessary for software requirements
- specify other software requirements based on analyses of the system requirements, system interfaces, system safety assessment (including system hazard analysis), computer security assessment, and required functions for verifying system and data integrity
- specify software requirements allocated to interfaces between the system, software, and human operators
- specify database requirements
- describe each software requirement giving enough information to design each component (e.g., initiator of action, action, object of action, conditions, constraints, source, destination, mechanism, reason)

- specify requirements and assertions for addressing the safety algorithms and the states and integrity of the system and identify appropriate responses to unfavorable results of assertions

- specify and flag software requirements associated with safety or security; specify software response to hazards, threats, including fault tolerance and error recovery

- specify any constraints or assumptions associated with the software requirements (e.g., time responses for each safety function)

- analyze each system requirement allocated to software for understandability, correctness, testability, consistency, and completeness (relate any ambiguities, inconsistencies, etc. to system personnel)

- confirm that the software requirements are developed according to standards for software requirements and do not violate any standards and requirements (e.g., time responses) for other system components

- analyze the software requirements for understandability, correctness, testability, consistency, and completeness, and any other quality attributes defined in the software requirements process

- report on any outstanding problems with the software requirements back to the system requirements process

- evaluate software requirements for test coverage and testability

- identify, explain and document any open issues between system and software requirements

- generate software requirements documentation to capture all of the information from the activities and draft a user's manual

- make any necessary changes to the software development processes and outputs based on the results of the software quality assurance (SQA), software verification and validation (SV&V), and software hazard analysis processes[5].

2.2 Software Design Process

The major objectives of the software design process are to develop the software design based on, and traceable back to, the software requirements, and to provide complete, consistent, correct, testable, and understandable information from which the software code may be generated. This process uses the SRS, the initial PMP, and software design standards identified in the SQAP to

[5]Software hazard analysis will most likely reveal that changes or additions to system requirements, software requirements, and test plans, at a minimum, are needed.

develop the software design. System documentation is available for reference. The software design process ends when its objectives and the objectives of any software assurance processes performed concurrently with it are met. The software design process produces a software design description (SDD), a database design description (DBDD), and possibly a revised SRS and/or updated PMP.

The following are activities of the software design process (software assurance activities listed in section 3 are performed concurrently with these activities):

- allocate software requirements (including interface requirements) to design components
- decompose components to their lowest level of detail necessary for coding the component
- describe external and internal interfaces for each component
- flag components associated with safety or security
- design assertions, responses to assertions and other required system algorithm and integrity checks or fault tolerance protections into the software in such a manner that will not adversely affect system performance
- plan and design the structure of any necessary databases
- implement the trace mechanism (i.e., execute the procedures that will establish the link between the software design and software requirements and software documentation)
- define the measurements that will be used to assess whether the design meets its requirements and quality attributes
- analyze the software design for understandability, correctness, testability, consistency, and completeness, and any other quality attributes defined in the software requirements process
- evaluate software design for feasibility of testing
- report on any outstanding problems with the software design (including interfaces) back to the software requirements process
- modify, if necessary, the SRS, user's manual, and/or PMP
- generate an SDD and DBDD

- make any necessary changes to the software development processes and outputs based on the results of the SQA, SV&V, and software hazard analysis processes[6].

2.3 Code Process

The major objective of the code process is to develop the source code based on, and traceable back to, the software design and software requirements. This process uses the SRS, SDD, DBDD, the PMP, and code standards identified in the SQAP, to develop the code. The code process ends when its objectives and the objectives of any software assurance processes performed concurrently with it are met. The code process produces the source code, a source code manual, and supporting documentation for source code. While the code process and unit test process are often associated with each other, the unit test process is a software assurance process and is described in section 3.3.5.

The following are activities of the code process (software assurance activities listed in section 3 are performed concurrently with these activities):

- develop source code for each software design component, including external and internal interfaces
- define, procure or generate, validate, and enter data into databases, if applicable (this may be done by the end-user)
- establish measures for assessing code
- flag source code components associated with safety or security
- code assertions, responses to assertions and other required system algorithm and integrity checks or fault tolerance protections into the software in such a manner that will not adversely affect system performance
- implement the trace mechanism (i.e., execute the procedures that will establish the link between the code, software design and software requirements and software documentation)
- examine the test coverage of units
- examine the feasibility of software integration
- debug the source code
- report on any outstanding problems with the source code back to the software design process

---

[6]Software hazard analysis will most likely reveal that changes or additions to software requirements, software design, and/or test plans, at a minimum, are needed.

- modify, if necessary, the user's manual
- generate the source code manual and any supporting documentation for source code
- make any necessary changes to the software development processes and outputs based on the results of the SQA, SV&V, and software hazard analysis processes[7].

2.4 Software Integration Process

The major objectives of the software integration process are to produce executable code, and to integrate the executable code into other software or other system components. This process uses the source code to integrate software components with other software components and with the hardware in preparation for installation into the system. The software integration process ends when its objectives and the objectives of any software assurance processes performed concurrently with it are met. The software integration process produces the executable code and a software installation plan. A software maintenance manual is started, and a user's manual is completed during this process. The software integration process occurs also in accordance with the overall system integration and test plan which may mean several iterations of this software integration process until all software components have been integrated with other system components.

The integration process described in this section pertains to software and must be coordinated with system integration.

The following are activities of the software integration process (software assurance activities (especially software integration test execution) listed in section 3 are performed concurrently with these activities):

- produce the executable code
- integrate components (e.g., integrate source code with other source code, prepare executable code for integration with other system components)
- evaluate the feasibility of software installation
- provide information on installing and executing the software for each site
- modify, if necessary, and complete the user's manual
- generate a plan for addressing software requirements when system is installed at its operating site, and draft a software maintenance manual

[7]Software hazard analysis will most likely reveal that changes or additions to software requirements, software design, code, and/or test plans, at a minimum, are needed.

- make any necessary changes to the software development processes and outputs based on the results of the SQA, SV&V, and software hazard analysis processes[8].

2.5 Software Installation Process

The major objective of the software installation process is to install the software at each site, and to determine whether the software will perform as required at all the sites in which it will operate. The software installation process ends when its objectives and the objectives of any software assurance processes performed concurrently with it are met. The software installation process produces a software installation report, and a software maintenance manual is completed.

The installation procedures described in this section pertain only to software and may need to be coordinated with system installation procedures.

The following are activities of the software installation process (software assurance activities listed in section 3 are performed concurrently with these activities):

- install the software at each site
- run software configured to each installation to confirm that the software meets its operating requirements at each site
- verify that the operator understands the user's manual
- generate a software installation report, and complete the software maintenance manual
- make any necessary changes to the software development processes and outputs based on the results of the SQA, SV&V, and software hazard analysis processes[9].

2.6 Software Operation and Maintenance Process

The major objective of the software operation and maintenance process is to ensure that the software meets its requirements throughout its operation and when modifications are made to it. This process uses the integrated software, software documentation, and software operation and maintenance standards to monitor the software throughout its operation, and modify the software as necessary (e.g., for error correction, enhancements, changes to operating environment) for every site at which the software is installed. Essentially, this process will repeat groups of the preceding processes. The activities below refer to the software installed at each different site. The software operation and maintenance process ends when its objectives and the objectives of any software assurance processes performed concurrently with it are met. The software operation and maintenance process produces a software operational procedures manual (if additional

---

[8]Software hazard analysis will most likely reveal that changes or additions to code and/or test plans, at a minimum, are needed.

[9]Software hazard analysis may reveal that changes or additions to certain software development processes are needed.

information is needed beyond the user's manual), and supporting documentation for modifications of the software (e.g., anomaly reports, modification feasibility reports).

The following are activities of the software operation and maintenance process (software assurance activities listed in section 3 are performed concurrently with these activities):

- provide information on operating the software, and generate a software operational procedures manual
- provide training to users of software
- assess validity of proposed modifications (e.g., technical feasibility, impact upon software and/or hardware, possibility of additional hazards)
- use the traceability records to identify the processes from section 2 and 3 that need to be applied to the software
- identify schedule, resources, and software assurance requirements for proposed modifications
- execute approved modifications and repeat the necessary software development and software assurance processes (e.g., a change made to the software requirements necessitates re-examining the entire software development process and their corresponding software assurance processes, especially test)
- generate supporting documentation for modifications
- make any necessary changes to the software development processes and outputs based on the results of the SQA, SV&V, and software hazard analysis processes[10].

2.7 Software Development Process Inputs and Outputs

Table 2-1 lists inputs and outputs for each software development process. The inputs may be from the system development process, system assurance process, software development process, and/or software assurance process. Inputs from each previous process should be available for use by the current process. The outputs are only from the software development process. The table also lists what software development outputs (created during a preceding software development process) may be modified, and what software assurance outputs (created during a preceding software assurance process) may be impacted by the particular software development process (this mainly includes changes to software assurance plans; creation of any software assurance reports is listed in table 2-1). This table is not intended to show who creates or modifies documentation (e.g., those personnel performing software development processes may produce some software assurance documentation).

[10]Software hazard analysis may reveal that changes or additions to certain software development processes are needed.

Table 2-1. Software Development Process Inputs and Outputs

PROCESS	INPUTS	DEVELOPMENT OUTPUTS CREATED	DEVELOPMENT OUTPUTS MODIFIED	ASSURANCE OUTPUTS IMPACTED
Software Requirements	■ system requirements ■ system design (includes system safety assessment and safety- and security-related requirements) ■ PMP ■ SQAP (software requirements standards)	■ SRS ■ draft user's manual		■ SQAP ■ SVVP ■ SCMP
Software Design	■ SRS ■ PMP ■ SQAP (software design standards)	■ SDD ■ DBDD	■ SRS ■ draft user's manual	■ PMP
Code	■ SRS ■ SDD ■ DBDD ■ PMP ■ SQAP (code standards)	■ source code ■ source code manual ■ supporting documentation for source code	■ draft user's manual	
Software Integration	■ source code	■ object code ■ executable code ■ software installation plan ■ software maintenance manual	■ final user's manual	
Software Installation	■ software integrated with hardware	■ software installation report	■ software maintenance manual	
Software Operation and Maintenance	■ all software documentation		■ software operational procedures manual ■ supporting documentation for modifications	■ documentation of any assurance process invoked, at a minimum SCM documentation

3 SOFTWARE ASSURANCE

The software development processes described in section 2 are accompanied by processes that assure the quality of the software produced from those processes. These software assurance processes include project management, software quality assurance, software verification and validation (includes test), software configuration management, and software hazard analysis. Software assurance processes locate problems in the software development process and their outputs, and provide evidence that the software complies with its specifications [NIST204]. These processes are separate from but performed concurrently with, and have a direct impact on, the software development processes. This framework does not specify who performs the software assurance processes, only what needs to be accomplished. An explanation of "independence" is provided in section 3.3.1.

Software assurance processes are separate from, but performed concurrently with, software development (see fig. 1-1), and can cause an iteration (see sec. 2) of the software development processes (which may in turn cause a change in the system requirements or system design). Several of the software assurance processes overlap (e.g., some project management information may also be addressed in the software quality assurance process), however, this does not alleviate the need for all of these software assurance processes. Part of assurance is monitoring that all the processes are planned and implemented.

For each software assurance process, this framework provides inputs and outputs (see Table 3-3 at the end of this section for a summary), the major objective(s) of the process, and activities within the process. The following documents were used in compiling this section: [ANS7432], [ANSP7432], [BERLACK], [DUNN], [ESA], [FIPS101], [FIPS132][11], [FUJII3], [IEEE610], [IEEE1042], [IEEEP1059], [NIST209], [NIST4909], [NIST204], [NUREG6018], [RTCA178B], [IEC880], [SOFTENG], [THAYER], and [WALLACE].

3.1 Project Management Process

The major objective of the (software) project management (PM) process is to establish the organizational structure of the project and assign responsibilities. This process uses the system requirements documentation and information about the purpose of the software, criticality[12] of the software, required deliverables, and available time and resources, to plan and manage the software development and software assurance processes. The PM process begins before software development starts and ends when its objectives have been met. The PM process overlaps and often reiterates other software assurance processes. It establishes or approves standards, monitoring and reporting practices, high-level policy for quality (process improvement and output quality), and cites regulations. The PM process produces a project management plan (PMP) which includes references to all other software assurance documentation.

[11][FIPS132] formally adopts [IEEE1012].

[12]See footnote 5.

It is outside the scope of this document to specify who is responsible for completing each software assurance process. However, one of the most important functions of the project management process is to ensure that all software development and software assurance processes are performed and monitored. This framework does not address the differences in project management for each type of manager (e.g., customer oversight, system project manager, IV&V manager, software project manager). Instead, this document addresses the responsibilities of the software project manager; if there is only a system manager, then the system manager must address the issues for software project management. Regardless of project organization, the manager responsible for the software must ensure that all software processes in development and assurance have been addressed.

The following are activities of the PM process:

Planning

- set objectives or goals - determine the desired outcome for the project
    - analyze and document the system and software requirements; define the relationships between the system and software activities
    - determine management requirements and constraints (resource and schedule limitations)
    - define success criteria; always includes delivery of software that satisfies the requirements, on time and within costs
- plan for corrective action
- develop project strategies - decide on major organizational goals (e.g., quality) and develop a general program of action for reaching those goals
- develop policies for the project - make standing decisions on important recurring matters to provide a guide for decision making
- determine possible courses of action - develop and analyze different ways to conduct the project; anticipate possible adverse events and project areas; state assumptions; develop contingency plans; predict results possible courses of action
- make planning decisions - evaluate and select a course of action from among alternatives
    - choose the most appropriate course of action for meeting project goals and objectives
    - make tradeoff decisions involving costs, schedule, design strategies, and risks

- select methods, tools, and techniques (both technical and managerial) by which the output and final product will be developed and assured, and the project will be managed (may also be included under software quality assurance)

- set procedures and rules for the project - establish methods, guides, and limits for accomplishing the project activities

- select scheduling process appropriate for development and assurance methods and language

- prepare budgets - allocate estimated costs (based on project size, schedule, staff) to project functions, activities, and tasks, and determine necessary resources

- document project plans - generate, implement, update, and distribute a PMP; the SQAP, SCMP, staffing and test plans may also be generated at this time

Organizing

- identify and group required tasks - tasks are grouped into logical entities (e.g., analysis tasks, design tasks, coding tasks, test tasks) are mapped into organizational entities

- select and establish organizational structures - define how the project will be organized (e.g., line organization, staff organization) using contractual requirements and the principles of independent verification and validation

- create organizational positions - specify job titles and position descriptions

- define responsibilities and authorities - decide who will have the responsibility of completing tasks and who has the authority to make decisions related to the project

- establish position qualifications - identify the qualities personnel must have to work on the project (e.g., experience, education, languages)

- document organizational structures - document lines of authority, tasks, and responsibilities in the project plan

Staffing

- fill organizational positions - fill the job positions established during organizational planning with qualified personnel

- assimilate newly assigned personnel - familiarize newly assigned personnel with any project procedures, facilities, or plans

- educate and train personnel as necessary (may also be included under software quality assurance)

- provide for general development of project staff members
- evaluate and appraise personnel
- compensate project personnel (e.g., salary)
- terminate project assignments - reassign or terminate personnel at the end of a project
- document staffing decisions - document staffing plans, training policies, etc.

Leading

- provide leadership - the project manager provides direction to project members by interpreting plans and requirements
- delegate project authority
- build project teams
- coordinate project activities - for example, define the software development and software assurance activities and their relationships to each other; determine how third party software, subcontracting, IV&V, safety and security will be managed
- facilitate communications - establish and implement mechanisms for communication within the software project, between software and system personnel, etc.
- resolve project conflicts
- manage changes - monitor progress of processes and implement changes; study and incorporate recommendations from development and assurance processes concerning processes and outputs
- document directing decisions - document decisions concerning lines of communication and coordination

Controlling

- develop standards of performance - select or approve standards to be used for the software development and software assurance activities (may also be included under software quality assurance)
- establish monitoring and reporting systems - establish and implement mechanisms and measurement practices for monitoring and reporting on software development and software assurance activities (e.g., milestones, deliverables, schedules)
- analyze results - compare achievements with standards, goals, and plans

- apply corrective actions - bring requirements, plans, and actual project status into conformance

- document the controlling methods listed above.

3.2 Software Quality Assurance Process

The major objectives of the software quality assurance (SQA) process are to ensure that the software development and software assurance processes comply with software assurance plans and standards, and to recommend process improvement. This process uses the system requirements, and information about the purpose and criticality[13] of the software to evaluate the outputs of the software development and software assurance processes. It begins before the software requirements process and ends when its objectives have been met. A software quality assurance plan (SQAP) and review and audit reports are produced during the SQA process.

The following are activities of the SQA process:

- select/approve standards, practices, methods, and tools to be used during the SQA process (including the standards used during the software development process)

- train programming staff in new techniques, methods, and tool use (may also be included under project management)

- review software development and software assurance plans for completeness and appropriateness

- evaluate effectiveness of current development and assurance methods and tools

- review software development and software assurance processes to ensure they comply with software assurance plans

- plan project management, quality, and test programs as policy or standards dictate

- apply, in accordance with the PMP, statistical process control techniques for process measurement

- use audits and reviews (e.g., software requirements review), analysis tools, and test to find defects at the earliest possible time

- enforce library control, change control, distribution, and storage per project management plan and relevant policies or standards; audit to verify that results are reported completely and accurately

- record all defects found and follow-up to make certain they are corrected

---

[13]See footnote 5.

- collect defect data and subsequent analysis of defect, fault detection, and failure modality
- use defect data to improve processes
- recommend changes to those software development and software assurance processes that do not meet their objectives
- apply error analysis and statistical analysis techniques to data collected from processes and products
- generate and analyze various data for early indication of adverse product or project control trends
- gather, analyze, and evaluate user feedback
- survey potential software vendors and their performance
- evaluate the fidelity with which plans and applicable standards are followed
- empower staff to prevent defective code, outputs of development and user documentation from being entered into the software or delivered
- generate and implement an SQAP.

3.3 Software Verification and Validation Process

The major objective of the software verification and validation (SV&V) process[14] is to comprehensively analyze and test the software concurrently with processes of software development and software maintenance to determine that the software performs its intended functions correctly, ensure that it performs no unintended functions, and measure its quality and reliability [NIST165]. SV&V is a detailed engineering assessment for evaluating how well the software is meeting its technical requirements, and in particular its safety, security and reliability objectives and for ensuring that software requirements are not in conflict with any standards or requirements applicable to other system components. There are SV&V activities to analyze, review, demonstrate or test the outputs of every software development and maintenance process; these SV&V activities may directly impact software development processes.

In this framework, the terms software verification and software validation are used together, i.e., software verification and validation. Software is *verified* at the end of *each* software development process (or increment of the process) to determine if the outputs of that software development process meet the requirements established at the beginning of that software development process. Validating (or "evaluating," in this framework) that the software correctly implements the system requirements for which the software is responsible is conducted

[14]For consistency within this framework concerning usage of the word "process," process as used in section 4.3 is equivalent to the word "task" in [WILEY] and [IEEE1012].

concurrently with and at the end of *all* the software development processes. The SV&V planning and analysis processes are conducted against system requirements at the highest level of planning, and then at the software requirements, which may be traced to the system requirements. Many SV&V processes, such as planning for software system test, require activities during processes generally associated with early development. Often, staff who perform verification of the requirements may be staff who prepare preliminary plans for software system tests; the development of the test plans and designs may lead to discovery of requirements errors. Planning and managing an entire SV&V process requires understanding of interrelationships among the SV&V activities and the advantage of applying knowledge from one activity to another. While in some instances this framework separates out software verification processes from software validation processes, these may be performed concurrently. The final, and ultimate, system validation must be planned in conjunction with test of all system components.

In the recently approved standard for digital computers in safety systems for nuclear power generating stations [IEEE7432], figure E1 shows the relationship of SV&V activities to other development activities and describes SV&V in Appendix E. SV&V in this framework expands on those V&V activities, for detailed software V&V processes. The SV&V process includes testing but is much more thorough than testing alone. The intention of the SV&V process is to ensure the absence of errors and measuring the quality and reliability of the system, which testing alone does not accomplish [RTCA178B]. The final goal of the SV&V process is that system objectives have been attained. This is evident only once system validation is completed (see fig. 1-1).

Table 3-1 provides a summary of the major processes of SV&V, as defined in [FIPS132] and expanded in [WALLACE]. SV&V analyzes the data from the SV&V processes to assess the quality and reliability of the software [WALLACE]; many techniques for performing these analyses are described in [NIST209]. Other guidance for assessing the reliability of the system may be found in [MUSA1, MUSA2, BUTLER]. According to [FIPS132] and [WALLACE], SV&V has some responsibilities during early system processes, as indicated also by figure E1 in [IEEE7432]. SV&V addresses verification of the *initial* project documentation, sometimes referred to as concept documentation (i.e., system concept, system requirements, and system design documentation). Activities supporting this SV&V activity are included for simplicity in the requirements verification process.

The SV&V process produces a software verification and validation plan (SVVP), individual plans and reports for activities, summary reports, anomaly reports, and a final software verification and validation report (SVVR). Some of the test documentation is prepared in advance of the test execution. For example, the system test plan for the software is developed concurrently with the software requirements process. Different management and technical staff may be responsible for different types of test (see sec. 3.3.1.).

The major objective of the SV&V process is stated at the beginning of this section. The activities unique to each sub-process of SV&V are identified in subsections of section 3, which are based primarily on [BEIZER], [ESA], [FIPS132], [FUJII3], [IEC880], [IEEEP1059], [NUREG6018], and [WALLACE]. Details on test documentation are provided by [FIPS132] and [IEEE829].

Table 3-1. Major Processes of SV&V

- Verification that outputs of each development and operations and maintenance process:
  - comply with previous software development process requirements and outputs (e.g., for completeness, correctness, consistency, and accuracy)
  - satisfy the standards, practices, and conventions of its software development process
  - establish the proper basis for initiating the next software development process activities.
- Validation that each output complies with established software and system requirements and assessment of the quality and reliability of the software.
- Unit test - test of an individual software element or groups of software elements (units) to verify the implementation of the design.
- Software integration test - test of software units that have been integrated with themselves and, according to the specific system integration plan, with hardware units, after each integration step, to verify that the integrated software correctly implements software requirements and design.
- Software system test - test the complete system to validate that the software as a complete entity complies with its operational requirements and satisfies system objectives.
- Installation test - examination of installation materials to ensure all software is included, testing to verify all site parameters or conditions, and checking that the installed software is the software subjected to SV&V.

3.3.1 Independent Verification and Validation

Some SV&V processes may be performed by two different groups (or different individuals within a group) whose objectives and activities for the process will have some differences, resulting in different evaluation strategies to demonstrate the objectives. While three types of independence are described in this framework, assignment of the processes is a management activity, which may be influenced by regulation and contract, and is outside the scope of this framework.

The use of a different organization for SV&V is called independent verification and validation (IV&V). The revision of [IEEE1012] may include the explanation of IV&V from the chapter on IV&V in [WILEY] for managerial, technical, and financial independence, as shown in the remainder of this section.

Technical independence requires that members of the IV&V team (organization or group) may not be personnel involved in the development of the software. This team must have some knowledge about the system design or have related experience and engineering background enabling them to understand the system. The IV&V team must not be influenced by the development team when the IV&V team is learning about the system, problems encountered, and proposed solutions for building the system. This technical independence is crucial in the team's ability to detect the subtle software requirements, software design and coding errors that escape detection by development testing and quality assurance reviews.

The technical IV&V team may need to share tools from the computer support environment (e.g., compilers, assemblers, utilities) but should execute qualification tests on these tools to ensure that the common tools themselves do not contain errors which may mask errors in the software being analyzed and tested. The IV&V team uses or develops its own set of test and analysis tools separate from the developer's tools whenever possible.

Managerial independence means the responsibility for IV&V belongs to an organization outside the contractor and program organizations that develop the software. While assurance objectives may be decided by regulations and project requirements, the IV&V team independently decides the areas of the software/system to analyze and test, techniques to conduct the IV&V, schedule of activities (within the framework of the system schedules), and technical issues to act upon. The IV&V team provides its findings in a timely fashion simultaneously to both the development team and the systems management who acts upon the reported discrepancy and findings.

Financial independence means that control of the IV&V budget is retained in an organization outside the contractor and program organization that develop the software. This independence protects against diversion of funds or adverse financial pressures or influences that may cause delay or stopping of IV&V analysis and test activities and timely reporting of results.

The extent that each of these parameters is vested in the IV&V team's responsibilities defines the degree of independence achieved. Based on the definitions of IV&V and how much IV&V a specific project requires, some SV&V processes may be conducted by both the developer and another organization. An example may be unit test. Unit test by one organization may focus on demonstrating that specific objectives have been met (e.g., safety objectives), which may differ from the objectives of the developer (e.g., logic structure, test coverage) [IEEEP1059].

3.3.2 Software Requirements Verification and Validation Process

Verification of the software requirements may also include an examination of documentation produced earlier in the system life cycle (e.g., initial feasibility studies, concepts on which the system has been designed). Inputs to the software requirements verification and validation process may be documents written in natural languages, formal mathematical languages, graphics and charts. When formal mathematical languages are used, other forms of representations may be provided to different users of the specifications. In this case, requirements verification must ensure fidelity between the forms of representation.

The following are activities of the software requirements verification and validation process:

- conduct a software traceability analysis[15] - trace software requirements to system requirements (and vice versa) and check the relationships for accuracy, completeness, consistency, and correctness; check that allocation is appropriate and complete

- conduct a software requirements evaluation - evaluate the software requirements for accuracy, completeness, consistency, correctness, testability, and understandability; assess

[15]This is an analysis of the trace established during the software requirements (development) process.

how well the software requirements accomplishes the system and software objectives; identify critical areas of software by assessing criticality of software requirements

- for individual requirements, measure completeness by verifying existence and correctness of defining properties: initiator of action, action, object of action, conditions, constraints, source, destination, mechanism, reason

- verify correctness and appropriateness of requirements and assertions for addressing the safety algorithms and the states and integrity of the system and responses to unfavorable results of assertions and that the operation of the assertions will not adversely impact system performance

- verify correctness and appropriateness of fault tolerance requirements and that their operation of the assertions will not adversely impact system performance

- conduct a software interface analysis - evaluate software requirements with hardware, user, operator and software interface requirements for accuracy, completeness, consistency, correctness, and understandability

- coordinate with system software test planning.

3.3.3 Software Design Verification and Validation Process

Software design verification occurs after the software requirements have undergone the verification process. By verifying that the software design meets its software requirements, the software design verification and validation process also supports validation that the software design meets system requirements, which was an objective of software requirements verification and validation. There may be several instantiations of the software requirements and software design verification before all of the system is verified.

The following are activities of the software design verification and validation process:

- conduct a software design traceability analysis[16] - trace software design to software requirements, and vice versa, and check the relationships for accuracy, completeness, consistency, and correctness

- conduct a software design evaluation - evaluate the software design for accuracy, completeness, consistency, correctness, and testability; evaluate design for compliance with software design standards (and, if appropriate, language standards) and software engineering practices; assess software design against assigned quality attributes

- conduct a software design interface analysis - evaluate software design with hardware, operator and software interface requirements for accuracy, completeness, consistency, and correctness

[16]This is an analysis of the trace established during the software design (development) process.

- verify that requirements for assertions, responses to assertions and other required system algorithm and integrity checks or fault tolerance protections have been designed into the software and are complete and accurate and will not adversely affect system performance
- apply software error, measurement, and statistical analysis techniques
- coordinate with software integration test planning.

3.3.4 Code Verification and Validation Process

Many of the activities to verify correct implementation of software design into code require tedious checking of details within the code; automation provides protection against human error in gathering the code information for analysis and also can speed the process. Code verification is the last opportunity to find and remove errors that could cause unnecessary costs and delays from advancing poor code into any of the test processes.

Code validation is accomplished through unit test which is described in section 3.3.5.

The following are activities of the code verification process:

- conduct a source code traceability analysis[17] - trace source code to software design, and vice versa, and check the relationships for accuracy, completeness, consistency, and correctness
- conduct a source code evaluation - evaluate the source code for accuracy, completeness, consistency, correctness, and testability; evaluate source code for compliance with code standards (and, if appropriate, language standards) and software engineering practices; assess source code against assigned quality attributes
- conduct a source code interface analysis - evaluate the source code with hardware, operator, and software interfaces for accuracy, completeness, consistency, and correctness
- apply software error, measurement, and statistical analysis techniques
- apply algorithm analysis and timing and sizing analysis techniques
- evaluate draft code-related documents (e.g., user manual, commentary within the code) with source code for completeness, consistency, and correctness
- coordinate with unit test[18].

---

[17]This is an analysis of the trace established during the code (development) process.

[18]Unit test is actually a part of code verification and validation.

3.3.5 Unit Test Process

Unit test is the test of the software elements at the lowest level of development. Units may be aggregates of software elements. Planning for unit test should occur concurrently with the software design process.

The following are activities of the unit test process:

- test planning - establish the objectives of the unit test, the strategies to be employed, the coverage requirements, reporting and analysis, and close-out of anomalies
- generate, monitor, and update an unit test plan to accomplish objectives
- trace design to test design, cases, procedures, and execution results
- confirm that anomalies during test are software anomalies, and not problems detected for other reasons
- test case and procedures generation - develop test cases and procedures for unit test and continue tracing as required by software test plans
- perform unit test - check software components individually for typographical, syntactic, and logic errors to ensure that each correctly implements the software design and satisfies the software requirements; execute the test cases; analyze results to verify anomalies; recommend changes to software design or code and conduct re-testing as necessary
- apply software error, measurement, and statistical analysis techniques
- document test activities and results.

3.3.6 Software Integration Test Process

Software integration test is performed to examine how units interface and interact with each other with the assumption that the units and the objects (e.g., data) they manipulate have all passed their unit tests [BEIZER]. Software integration tests check how the units interact with other software (e.g., libraries) and hardware. The software integration test schedule depends upon the development and integration schedules for software units, hardware, and other components. For large systems, software integration test planning may require intense communication among all system personnel to ensure that the overall test objectives can be achieved by the selected test strategy. For each major integration that has passed interface and interaction tests, functional tests may be developed and executed [BEIZER]. When all system components have been integrated and have successfully passed software integration tests, then the system moves into system test for testing of the system through the complete system process.

The following are activities of the software integration test process:

- test planning - establish the objectives of the software integration test, the strategies to be employed, the coverage requirements, reporting and analysis, and close-out of anomalies

- generate, monitor, and update a software integration test plan to accomplish objectives

- trace software requirements to test design, cases, procedures, and execution results

- test case and procedures generation - develop test cases and procedures for unit test and continue tracing as required by software test plans

- perform software integration test - check the inter-component communication links and test aggregate functions formed by groups of components; confirm that anomalies during test are software anomalies, and not problems detected for other reasons; ensure any changes to software requirements, software design or code are made and conduct re-testing as necessary; conduct functional, structural, performance, statistical and coverage testing of successfully integrated components after each software integration process and successful testing of interfaces and interactions

- apply measurement and statistical analysis techniques

- document test activities and results.

3.3.7 Software System Test Process

Software system test, in the context of SV&V, involves the conduct of tests to execute the completely integrated system. Software system test is the validation that the software meets its requirements. Validation of the complete system may involve many tests involving all system components. The software system tests exercise only those system functions that invoke software. The perspective is on the software aspects of the system, and whether the software behaves as intended relative to complete system performance. These tests must be conducted in such a manner as to stress and break the system based on software responses to system inputs (e.g., from sensors, operators, databases). Tests and data collected from the tests are designed to provide an operational profile of the system which support a statistical analysis of the system reliability [MUSA1, MUSA2, BUTLER]. This section of the framework addresses only the tests that validate that the software implements the system requirements; other tests for other components and perspectives are necessary for complete system validation.

While software system tests are conducted once the system has been built, it is imperative that planning for these tests is conducted concurrently with the software requirements process because:

- analyzing the software requirements for test requirements may result in finding requirements errors and/or discovery of untestable requirements

- development or procurement of test facilities (e.g., model of operational environment) and CASE tools (e.g., test case generators, test data base) may require as much time as development, and these resources must be planned.

The following are activities of the software system test process:

- test planning - establish the objectives of the software system test, the strategies to be employed, the coverage requirements, reporting and analysis, and close-out of anomalies
- generate, monitor, and update a software system test plan to accomplish objectives
- trace system and software requirements to test software design, cases, procedures, and execution results
- test case and procedures generation - develop test cases and procedures for unit test and continue tracing as required by software system test plans
- test the operation of the software as an entity (sometimes a simulated environment may be used); confirm that anomalies during test are software anomalies, and not problems detected for other reasons; ensure any changes to software (software requirements, software design, code, or test cases) have been made and conduct re-testing as necessary
- apply measurement and statistical analysis techniques
- document test activities and results.

3.3.8 Software Installation Test Process

Software installation test is the final step before launching full customer acceptance testing. The intent of software installation test is not system validation nor acceptance testing. The purpose of installation test is only to demonstrate that the correct software has been delivered and that the software interfaces are correct relative to any interfaces at the installation site. Acceptance testing, which involves the user/customer, is outside the scope of this document.

The following are activities of the software installation test process:

- conduct an installation configuration audit - determine that all software outputs needed to operate the system is present; check that the software installed in the system is the software that underwent SV&V
- develop and execute tests that will examine and stress site-unique parameters (e.g., printer interface, operating system interface, monitor interfaces)
- generate applicable documentation
- generate an SVVR (or generate it at the end of the SV&V process).

3.3.9 Software Operation and Maintenance Verification and Validation Process

The operation of computer software requires periodic checks that the integrity of the system has been maintained, that any changes to the system which affect its operation have been documented and operators have received training in new or changed procedures.

SV&V of the maintenance of software, (e.g., adaptive, corrective, perfective [FIPS106]), requires planning for SV&V based on the extent of the maintenance and hence a revisit of all the software development processes to identify to what extent each SV&V processes must be performed.

The following are activities of the software operation and maintenance verification and validation process:

- conduct an anomaly evaluation - evaluate the severity of anomalies during software operation and their effect on the system
- conduct a proposed change assessment - assess proposed changes to the software and their effect on the system to determine SV&V activities to be repeated and conduct them again
- develop a SV&V plan and repeat processes according to section 3.3.

3.4 Software Configuration Management Process

The major objectives of the software configuration management (SCM) process are to track the different versions of the software, and ensure that each version of the software contains the exact software outputs generated and approved for that version. It must be established before software development starts and continues throughout the software development processes. SCM is responsible for ensuring that any changes to any software outputs during the development processes are made in a controlled and complete manner.

The SCM process produces a software configuration management plan (SCMP). When the software is integrated with system components, system configuration management begins. However, any changes to the software necessitates that SCM be invoked.

The following are activities of the SCM process:

- generate an SCMP

Software configuration identification

- identify configuration items (CIs), i.e., select the most significant and critical functions that will require constant attention and control throughout software development
- assign a unique identifier/number to each CI

- establish baselines for CIs, i.e., documents that have been formally reviewed and agreed upon, that thereafter serve as the basis for further development, and that can be changed only through formal change control procedures
    - functional baseline - the completion and acceptance of the system requirements specification--the prerequisite for the development of the software requirements specification (SRS) for each CI
    - allocated baseline - the review and acceptance of the SRSs--the prerequisite of the development of the software design description (SDD) for all components making up a CI
    - developmental configuration (developer-controlled "rolling" baseline) - all of the documents and code accepted and committed for configuration control up to the establishment of the product baseline
    - product baseline - established with the successful conclusion of a configuration audit--prerequisite to the operation and maintenance of the software

Problem reporting, tracking and corrective action

- document when a software development activity does not comply with its plan, output deficiency, or anomalous behavior, and the corrective action taken

Change control

- document, evaluate, resolve, and approve changes to the software

Change review

- assess problems and changes, implement approved changes, provide feedback to processes affect by changes

Traceability analysis

- trace forward and backward through the current software outputs to establish the scope of impacted software

Configuration control

- delegate authority for controlling changes to software; determine method for processing change requests

Configuration status accounting

- keep records detailing the state of the software product's development, e.g., record changes made to the software, status of documents, changes in process, change history, release status, etc.

Configuration audits and reviews

- audit configuration items before release of product baseline or updated version of product baseline; review to determine progress and quality of product
    - functional configuration audit - prove that a CI's actual performance agrees with its software requirements stated in the SRS
    - physical configuration audit - ensure that the documentation to be delivered with the software represents the content of the software product

Archive, retrieval and release

- archive software outputs (with backups) so it can not be changed without authorization and it will not deteriorate data to ensure it can be retrieved if necessary; describe software being released to ensure it is authorized

3.5 Software Hazard Analysis Process

The overall objective of the software hazard analysis process is to ensure that software hazards and hazards related to interfaces between the software and the system have either been eliminated or their risk has been mitigated. This process uses the system requirements, preliminary hazard list, preliminary hazard analysis, system hazard analysis, SRS, SDD, DBDD, and safety-related history of similar systems to identify software hazards and evaluate the risk of software hazards, and then eliminates or reduces the risk of the hazards. It begins before the software requirements process and ends when its objectives have been met. The software hazard analysis process produces a software safety plan and documents that report on the results of the different software hazard analyses (i.e., software requirements hazard analysis report, software design hazard analysis report, code-level software hazard analysis report, software safety testing report, software/user interface analysis report, software change hazard analysis report).

The following are activities of the software hazard analysis process:

- for software requirements hazard analysis (SRHA) examine the system requirements, software requirements and software design to ensure that system safety requirements have been properly defined, and that they can be traced from the system requirements to the software requirements, software design, user's and operational procedures manuals; incorporate recommendations and requirements into the software design description and the software test plans

- for software design hazard analysis (SDHA) define and analyze safety critical software components (e.g., assessing their degree of risk, relationships to other components) and the design and software test plans (e.g., ensuring safety requirements are properly defined in the design); make changes to the software design description and the software test plans; make recommendations for coding

- for code-level software hazard analysis (CSHA) analyze the source and object code, system interfaces, and software documentation to ensure safety requirements are included; make recommendations to change the software design, code, and software testing

- for software safety testing test safety-critical software components under normal conditions and abnormal environment and input conditions; after the software is corrected it is then retested under the same conditions

- for software/user interface analysis make modifications to the design to control hazards that were not eliminated or controlled in the system design phase by implementing, e.g., the ability of the operator to terminate an event or process

- for software change hazard analysis analyze all changes (resulting from preceding software hazard analyses) made to the software to ensure they do not create new hazards of effect existing hazards

- make any necessary changes to those software development process outputs that do not meet the above software hazard analysis objectives.

3.6 Software Assurance Process Inputs and Outputs

Table 3-2 lists inputs and outputs for each software assurance process. The inputs may be from the system development process, system assurance process, software development process, and/or software assurance process. The outputs are only from the software assurance process. The table also lists what software assurance outputs (created during a preceding software assurance process) may be modified, and what software development outputs (created during a preceding software development process) may be impacted by the particular software assurance process. This table is not intended to show who creates or modifies documentation.

Table 3-2. Software Assurance Process Inputs and Outputs

PROCESS	INPUTS	ASSURANCE OUTPUTS CREATED	ASSURANCE OUTPUTS MODIFIED	DEVELOPMENT OUTPUTS IMPACTED
Project Management	■ system requirements ■ software purpose ■ software criticality ■ deliverables required ■ available time & money	■ PMP	■ SQAP	
Software Quality Assurance	■ system requirements ■ software purpose ■ software criticality ■ software development process outputs ■ software assurance process outputs	■ SQAP ■ review and audit reports	■ PMP ■ SVVP ■ SCMP ■ software safety plan	■ SRS ■ user's manual ■ SDD ■ source code manual ■ software installation plan ■ software maintenance manual ■ operational procedures manual
Software Verification and Validation	■ system requirements ■ software development processes outputs	■ SVVP ■ software test plans ■ test designs, cases, procedures ■ task, summary, anomaly reports ■ SVVR		■ system test plan ■ software development processes outputs
Software Configuration Management	■ software development processes outputs ■ software assurance processes outputs	■ SCMP		
Software Hazard Analysis	■ system requirements ■ preliminary hazard list ■ preliminary hazard analysis ■ system hazard analysis ■ SRS ■ SDD ■ DBDD	■ software safety plan ■ SRHA report ■ SDHA report ■ CSHA report ■ software safety testing report ■ software/user interface analysis report ■ software change hazard analysis report	■ SCMP ■ software test plans ■ software test procedures, cases	■ system specifications ■ software development processes outputs ■ software test plans ■ software test procedures, cases

4 SOFTWARE ENGINEERING PRACTICES

Software engineering practices are those techniques recommended either to prevent errors from being entered into the software during development, or are properties to be built into high integrity software [NIST204]. The following is a summary of some software engineering practices that may enhance the quality of the software.[19]

Formal methods may be used to specify/model the requirements mathematically. A recent study supports the concept that formal methods may eliminate ambiguity in the requirements but cannot ensure completeness. The report suggests that better methods of technology transfer and better automated support are needed before formal methods can be widely used [NIST626]. [FUJII] includes a methodology for describing software specifications in English. The use of either formal methods or the [FUJII] approach requires analyzing the completeness and meaning of each requirement. However, one example in [FUJII] demonstrates that neither method can eliminate all ambiguity nor prove the completeness of the total set of requirements. Formal methods can also be used for verifying the requirements and for design proof of correctness.

Prototyping, simulation, and modeling can be used in developing software requirements, and in the software design process. The software requirements process may use simulation and modeling to determine if it is feasible to build a product to the requirements [NIST213]. The software design process can use simulation and modeling to determine the effectiveness of alternative designs [NIST213]. Rapid prototyping and simulation analysis are useful in the verification and validation of the software requirements, software design, and code. The project management process may use simulation and modeling to perform tradeoff studies of alternative strategies [NIST213].

The way in which the software is designed contributes greatly to its quality. Component isolation separates safety critical components from other components, making analysis of, and changes to, these components easier to accomplish. Modularity ensures that changes to one component minimally affect other components. Information hiding prevents components' actions from interfering with other components. Redundancy is used to prevent or recover from failures. Interaction with the operator or user of the software system during the design of the software/human interface can also be helpful.

Using a software design methodology that is well suited to the software application is important. Today, new technology is forcing a second look at design methods, specifically object-oriented design (OOD). NIST conducted a study of the attributes of OOD relative to safety-critical software for the United States Nuclear Regulatory Commission (NRC). The purpose was to describe attributes of OOD (e.g., classes, encapsulation, inheritance) relative to their capability for supporting features desired in software for safety systems (e.g., modularity, functional diversity, traceability, and non-ambiguity). The results were presented at the NRC/NIST workshop in September 1993 and published in the workshop proceedings [NIST216].

---

[19]See section 5 for a discussion on more work with software engineering practices.

The use of high-level languages has also been recommended for quality software [NIST204]. Using high level, standard languages and their standards lessens programming errors. Eliminating programming practices that have been demonstrated to be problematic (e.g., floating point arithmetic, use of interrupts) simplifies analyzing system behavior. It is also important to use a language with a thoroughly tested compiler.

Reverse engineering can aid in developing software requirements, recreating documentation for preexisting software, and providing a basis for reusability of software. Re-engineering can be used to change software design when software requirements change [NIST213].

There are also software engineering practices that apply to the software assurance processes. Use of cost-modeling and risk assessment techniques can aid the project management process. The use of selected software hazard analysis techniques (e.g., software fault tree analysis, petri nets) can aid in software assurance by identifying the critical parts of the software. Inspections, reviews, and audits can be applied to all software processes under the software quality assurance process. Software error, measurement, statistical, algorithm, database, technical, control and data flow, and timing and sizing analysis techniques are useful in the software verification and validation process. Test strategies such as equivalence partitioning, cause-effect, boundary value, stress, event directed, data flow, logic flow, performance, timing, sizing, random, top-down, bottom-up, sandwich, statistical testing, functional testing, and performance testing, when applied appropriately, contribute to the quality of the software.

5 SOFTWARE FUNCTIONALITY

While this framework does not address development of the entire system, information about the system must be provided to the software development and software assurance teams. This section of the framework is not complete and is intended only as an overview of aspects about high integrity software systems that these teams should be provided[20]. This section was developed in cooperation with SoHaR, Inc.

The software activities are dependent on the system engineering functions to define requirements in at least the following areas:

- the service to be performed by the system in each operating mode
- failure modes of the hardware required for these functions, fault detection requirements (including calibration and self-test) and fault tolerance provisions and algorithms
- specification of actions to be avoided by the system
- identification of the human interfaces for (a) normal operation, (b) exceptional operating states (recovery from hardware failures, etc.), and (c) maintenance and other non-operational states
- system level test activities and the software support required for these (test drivers, simulators, enabling/disabling provisions for certain functions)
- attribute requirements: quality assurance, configuration management, reliability, and availability.

For each of the above items both the system requirements and the specific subset to be implemented in software must be identified. Other information that must be provided to the software developers includes description of the external system interfaces, user procedures, user and maintainer skill levels, safety and security requirements and any functions designed to mitigate or check for problems during system operation.

5.1 Definition of System Service

The definition of system service enables the software designer to provide required software functions. The following are specific items that apply to high integrity software:

- operating modes (system start-up, routine operation, maintenance or test mode, shut-down)
- allowable transitions between modes

[20]The information in this section is appropriate for systems which are software-intensive and whose failure may cause significant social, environmental, or financial damage.

- method and frequency of invocation in each mode (cyclic, by event, operator command)
- possible states of the controlled system at time of invocation.

5.2 Failure Modes, Error Detection and Fault Tolerance Requirements

In some high integrity software systems, hardware installation may provide for redundant channels for specific functions relative to attributes like safety or security. Software may be required to validate operational channels, identify faulty channels to the operators, perform automatic switching between channels to maintain the system operation after a fault has been diagnosed, and to initiate alerts when the system is no longer fully functional. These activities are collectively referred to as surveillance. Sensors are substantial contributors to the system failure probability, and frequently sensors have a higher degree of redundancy than other hardware components. Sensor surveillance is therefore discussed in a separate subsection below.

5.2.1 Sensor Surveillance

Sensors operate under more severe environmental conditions than other parts of the system. Their output normally contains a noise component--it can drift, and it is frequently affected by variations in the power supply. In addition, the sensor can experience transient or permanent failure. In analog systems sensor surveillance is a labor intensive activity that is usually automated (i.e., implemented in software) in digital systems. The sensor surveillance software typically analyzes a time series of sensor outputs, extracts a current estimate of the true value of the sensed quantity from the noisy raw measurements, and must make decisions about the validity of the current estimate (i.e., whether the sensor has failed). If a failure has been identified there may be further decisions required about the value of the affected variable that is utilized in the system, e.g., to minimize sensor switching for transient failures it can be temporarily held at the last valid level and the affected sensor sampled again during the next interval. The design of sensor surveillance software requires identification of sensor and power supply redundancies, the preferred sensor configurations, and the following data:

- sensor failure modes
- sensor range under normal plant conditions
- sensor range under abnormal plant conditions
- mechanical and electrical limits on sensor output
- maximum expected change in output between samplings
- noise characteristics of the sensor and power supply
- worst expected drift characteristics of the sensor
- allowable time interval between abnormal sensor output and safety (security) action

- typical time history of conditions requiring safety (security) action.

Sensor surveillance normally includes the wiring to the control components; i.e., a failure in the connection will be treated as a sensor failure. Where separate surveillance of the wiring is desired, the software designer will need data that permit a differentiation between sensor and wiring failures.

5.2.2 Surveillance of Other System Components

Other system components typically include the computer and output devices, such as a relay network. In some cases the output interface includes actuation of control rods or pumps. Surveillance of computer operation includes at least a self-health check, but it can also include monitoring of computers in other channels, of analog-to-digital interfaces, and of intra- and inter-channel communications.

The surveillance of the output devices involves comparison of the commanded state (as generated within the computer) with the actual state and reported by an independent measurement. For relay networks this measurement is usually provided by an auxiliary contact that operates in synchronism with the main contacts; rod position can be determined from dedicated sensors, and pump operation from centrifugal switches or tachometers.

The software designer needs the following data to support required functionality:

- computer and output device failure modes
- error detection and correction requirements arising from these
- the topology of the intra- and inter-channel communications
- alternate allowable topologies to deal with component failures
- data formats used by each communications path
- maximum expected delay between output command and output activation
- allowable delay between detection of a faulty state and annunciation.

5.3 Actions to be Avoided

Actions to be avoided fall into two broad categories:

- actions to be avoided in normal computer operation
- actions to be avoided after computer or software failure.

The first category includes actions that may result from failures outside the computer, such as an erroneous sensor measurement or inappropriate operator actions (mode changes). Examples are:

- prohibition of repeated output commands (sending a command twice)
- definition of prohibited output sequences
- actions to be avoided during or immediately following a mode change
- prohibition of actions after detection of a sensor failure
- prohibition of actions after detection of an output device failure.

Examples of the second category are:

- actions to be avoided after self-diagnosis of a failure
- actions to be avoided after detecting failure of another computer
- prohibited actions after entering a software exception handler.

In addition to these requirements that are derived from the system specification certain actions to be avoided may be established on the basis of software considerations, e.g., prohibition of certain calling sequences.

5.4 Human Interfaces

Although some systems are frequently intended to serve functions in which the human response may be too slow or uncertain, they are not insulated from interfaces with operators and maintainers. Under failure-free conditions of these systems the operator initiates mode changes and monitors plant and system status indications furnished by the automated system. Under exceptional states of these systems, such as recovery from a hardware failure, the operator is responsible for taking corrective action, such as initiating maintenance. And once the system is in a maintenance mode, human skill and judgment is required to bring it back to operation. These essential human interfaces demand that the software developer be aware of:

- availability and capabilities of the operational staff
- desired staff initiated test provisions for the system
- human interfaces of present or predecessor systems (to avoid introduction of inconsistent input or display formats)
- alternate actions that may be initiated by the operational staff for a given plant condition (including remotely initiated actions)

- alternate indications of a given plant condition available to the operators
- training facilities for the operational staff (to permit integration of training for the system under development
- availability and capabilities of the maintenance staff
- diagnostic provisions desired by the maintenance staff
- plant operating procedures while system maintenance is in progress
- procedures for restoring the system to operation following maintenance.

5.5 System Test Provisions

To facilitate system test it is frequently desirable to (a) disable or modify certain software controlled functions, (b) add temporarily functions normally supplied by the system environment, and (c) provide indications and records of test progress. If these requirements are realized at the outset, patching or other irregular software structures can be avoided. Requirements for the following functionality should be provided, associated with the test phases for which they will be activated:

- functions to be disabled or modified, e.g., feedback of output actuation
- differences in input timing or sequencing
- single channel operation (vs. multiple channels in the plant)
- fault insertion capability (including superposition of noise)
- simulation of operator commands
- programmed or random generation of inputs or internal states
- indications or recording of internal states, test sequence numbers, and generated outputs.

5.6 Attribute Requirements

System level attribute requirements must be propagated and interpreted for the software development. The primary attribute requirements arise from quality assurance, configuration management, reliability and availability. Security and portability (ability to operate on multiple computer types) requirements may also be invoked. In most cases these requirements must be interpreted for software development, and this interpretation is a joint system engineering and software engineering responsibility.

The most stringent requirements are usually intended only for the code associated with the activation of a particular safety function (e.g., in a nuclear power plant, a reactor shut-down). But

the extent of that software segment and the attribute requirements for other segments must be identified by joint system engineering and software engineering analysis techniques. Typical topics are:

- status (safety-critical or not) of:
  - sensor surveillance software
  - software for monitoring and diagnostic indications
  - mode change software.
- attribute requirements for:
  - the above functions judged to be not safety-critical
  - test support software (subsection 5 above)
  - software exclusively used in non-operational modes.

6 SUMMARY

This framework proposes the activities that comprise software development and software assurance processes, independent of the technology used to perform them. Users of the framework may implement these activities with methods which are most appropriate to the software application domain.

This document is an initiating activity in support of the Center for High Integrity Software Systems Assurance (CHISSA), whose purpose is to foster and coordinate activities relating to high integrity software technology. High integrity software needs to be developed and assured in a planned and systematic manner. Development of this software includes processes for the software requirements, software design, code, integration of the code, installation of the software, and the continuing operation and maintenance of the software.

The development of the software is controlled and monitored by assurance processes which encompass managing the entire software project, assuring the quality of the software, verifying and validating the software against its requirements, managing the different configurations of the software, and eliminating or mitigating software hazards.

The processes of software development and assurance are not stand-alone tasks; information about the system must be provided to the software team throughout the software development and assurance processes. Information specific to systems is crucial in developing and assuring high integrity software; this framework identifies some information for the system that affects software functionality. This framework does not address software documentation in detail because the issues of documentation should be addressed in a separate research project.

This framework will undergo substantive change and expansion. Future work in expanding this framework includes, but is not limited to, the following tasks:

- definition of the interfaces between software and system
- development of a profile of functionality for high integrity software systems which may be further refined for application domains and may be used to identify specific technical problems
- identification of appropriate software engineering methods (or practices) mapped to application domains or technical problems which those methods resolve
- identification where current methods are inadequate and further research is needed
- examination of types of CASE tools for implementing recommended software engineering methods supporting these activities
- examination of integration capabilities of CASE tools
- definition of a comparable framework for system development and assurance both as an entity and specifically for each system component.

7 REFERENCES

[ANS501]
ANSI/ANS-50.1, "(DRAFT #6) Nuclear Safety Design Criteria for Light Water Reactors," American Nuclear Society, January 1993.

[ANS7432]
ANSI/IEEE-ANS-7-4.3.2-1982, "Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations," American Nuclear Society, 1982.

[ANSP7432]
P-7-4.3.2, draft 7, "American National Standard - Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," Sponsor: Nuclear Power Engineering Committee of the IEEE Power Engineering Society.

[BEIZER]
Beizer, Boris, Software Testing Techniques, Van Nostrand Reinhold, New York, 1990.

[BELTRACCHI]
Beltracchi, Leo, "NRC Research Activities," NIST Special Publication 500-216, *Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop (NUREG/CP-0136)*, U.S. Department of Commerce/National Institute of Standards and Technology, March 1994.

[BERLACK]
Berlack, Ronald H., "Configuration Management," *Encyclopedia of Software Engineering*, Volume 1, John Wiley & Sons, Inc., 1994.

[BUTLER]
Butler, R. and G. Finelli, "The Infeasibility of Experimental Quantified Life-Critical Software Reliability," *Proceedings of SIGSOFT'91: Software for Critical Systems*, Association for Computing Machinery, December 1991.

[DUNN]
Dunn, Robert H., "Quality Assurance," *Encyclopedia of Software Engineering*, Volume 2, John Wiley & Sons, Inc., 1994.

[ESA] ESA PSS-05-10 Issue 1, "Guide to Software Verification and Validation," European Space Agency, February 1994.

[FIPS101]
FIPS 101, "Guideline for Lifecycle Validation, Verification, and Testing of Computer Software," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1983 June 6.

[FIPS106]
FIPS 106, "Guideline on Software Maintenance," U. S. Department of Commerce/National Bureau of Standards (U.S.), 1984 June 15.

[FIPS132]
FIPS 132, "Guideline for Software Verification and Validation Plans," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1987 November 19.

[FUJII1]
Fujii, Roger U., "Software Engineering For Instrumentation and Control," American Nuclear Society, *Nuclear Plan Instrumentation, Control, and Man-Machine Interface Technologies, Oak Ridge, TN*, April 1993.

[FUJII2]
Fujii, Roger U., "How Much Software Verification and Validation is Adequate for Nuclear Safety?" NIST Special Publication 500-216, *Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop (NUREG/CP-0136)*, U.S. Department of Commerce/National Institute of Standards and Technology, March 1994.

[FUJII3]
Fujii, Roger U., "Independent Verification and Validation," *Encyclopedia of Software Engineering*, Volume 1, John Wiley & Sons, Inc., 1994.

[IEC880]
IEC 880, "Software for Computers in the Safety Systems of Nuclear Power Stations," International Electrotechnical Commission, 1986.

[IEEE603]
IEEE Std 603-1980, "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations," The Institute of Electrical and Electronics Engineers, Inc., November 24, 1980.

[IEEE610]
ANSI/IEEE Std 610.12-1990, "Glossary of Software Engineering Terminology," The Institute of Electrical and Electronics Engineers, Inc., 1990.

[IEEE1012]
ANSI/IEEE Std 1012-1986, "IEEE Standard for Software Verification and Validation Plans," The Institute of Electrical and Electronics Engineers, Inc., February 10, 1987.

[IEEE1042]
ANSI/IEEE Std 1042-1987, "IEEE Guide to Software Configuration Management," The Institute of Electrical and Electronics Engineers, Inc., March 10, 1988.

[IEEE7432]
ANSI/IEEE Std 7432-1993, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," The Institute of Electrical and Electronics Engineers, Inc., 1993.

[IEEEP1059]
IEEE Std P1059-1994, "(DRAFT 7.1) IEEE Guide for Software Verification and Validation Plans," Institute of Electrical and Electronics Engineers, Inc., May 24, 1993.

[IEEEP1228-H]
P1228, "(DRAFT H) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., 4/27/92.

[ISO12207]
ISO/IEC DIS 12207-1, "(DRAFT) Information Technology--Software--Part 1: Software Life Cycle Process," International Electrotechnical Commission, 1994.

[MIL498]
MIL-STD-498, "Software Development and Documentation," Department of Defense, 30 November 1994.

[MIL882B]
MIL-STD-882B, "System Safety Program Requirements," Department of Defense, 30 March 1984.

[MUSA1]
Musa, J.D., A. Iannino, and K. Okumoto, Software Reliability, Measurement, Prediction, Application, McGraw-Hill, New York, 1987.

[MUSA2]
Musa, J.D., and A.F. Ackerman, "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, May 1989.

[NIST165]
NIST Special Publication 500-165, "Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards," U.S. Department of Commerce/National Institute of Standards and Technology, September 1989.

[NIST190]
NIST Special Publication 500-190, "Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991," U.S. Department of Commerce/National Institute of Standards and Technology, August 1991.

[NIST204]
NIST Special Publication 500-204, "High Integrity Software Standards and Guidelines," U.S. Department of Commerce/National Institute of Standards and Technology, September 1992.

[NIST209]
NIST Special Publication 500-209, "Software Error Analysis," U.S. Department of Commerce/National Institute of Standards and Technology, April 1993.

[NIST213]
NIST Special Publication 500-213, "Next Generation Computer Resources: Reference Model for Project Support Environments (Version 2.0)," U.S. Department of Commerce/National Institute of Standards and Technology, November 1993.

[NIST216]
NIST Special Publication 500-216, "Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop (NUREG/CP 0136)," U.S. Department of Commerce/National Institute of Standards and Technology, March 1994.

[NIST626]
NIST GCR 93/626, "An International Survey of Industrial Applications of Formal Methods Volume 1 Purpose, Approach, Analysis, and Conclusions," U.S. Department of Commerce/National Institute of Standards and Technology, March 1993.

[NIST4909]
NISTIR 4909, "Software Quality Assurance: Documentation and Reviews," U.S. Department of Commerce/National Institute of Standards and Technology, September 1992.

[NUREG6018]
NUREG/CR-6018, "Survey and Assessment of Conventional Software Verification and Validation Methods," U.S. Nuclear Regulatory Commission, April 1993.

[RTCA178B]
RTCA/DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," RTCA, Inc., December 16, 1992.

[SOFTENG]
"Standard for Software Engineering of Safety Critical Software," Draft, Rev. 0, Ontario Hydro, December 1990.

[THAYER]
Thayer, Richard H. and Richard Fairley, "Project Management," *Encyclopedia of Software Engineering*, Volume 2, John Wiley & Sons, Inc., 1994.

[WALLACE]
Wallace, Dolores R., "Verification and Validation," *Encyclopedia of Software Engineering*, Volume 2, John Wiley & Sons, Inc., 1994.

[WILEY]
Encyclopedia of Software Engineering, John Wiley & Sons, Inc., 1994.

APPENDIX A. BIBLIOGRAPHY OF HIGH INTEGRITY SOFTWARE DOCUMENTS

A.1 Standards and Guidelines

AF800-5

AFSC/AFLCP 800-5, "(DRAFT) Software Independent Verification and Validation (IV&V)," Air Force Systems Command and Air Force Logistics Command, 1988.

AF800-45

AF PAMPHLET 800-45, "Software Independent Verification and Validation (IV&V)," Department of the Air Force, 1 May 1991.

AFISC

AFISC SSH 1-1, "Software System Safety," Headquarters Air Force Inspection and Safety Center, 5 September 1985.

ANS103

ANSI/ANS-10.3-199x, (DRAFT 5), "Documentation of Computer Software," American Nuclear Society, 3/7/92.

ANS104

ANSI/ANS-10.4-1987, "Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry," American Nuclear Society, May 13, 1987.

ANS501

ANSI/ANS-50.1, "(DRAFT #6) Nuclear Safety Design Criteria for Light Water Reactors," American Nuclear Society, January 1993.

ANS7432

ANSI/IEEE-ANS-7-4.3.2-1982, "Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations," American Nuclear Society, 1982. AND ANSI/IEEE-ANS-7-4.3.2-19XX, Draft 2, as of November, 1991.

ANSP7432

P-7-4.3.2, draft 7, "American National Standard - Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," Sponsor: Nuclear Power Engineering Committee of the IEEE Power Engineering Society.

ANSIX99

ANSI X9.9-1986, "Financial Institution Message Authentication (Wholesale)," X9 Secretariat, American Bankers Association, August 15, 1986.

ANSIX917

ANSI X9.17-1985, "Financial Institution Key Management (Wholesale)," X9 Secretariat, American Bankers Association, April 4, 1985.

AQAP13
AQAP-13, "NATO Software Quality Control System Requirements," NATO, August 1991.

ASMENQA1
ASME NQA-1-1989, "Quality Assurance Program Requirements for Nuclear Facilities," The American Society of Mechanical Engineers, September 15, 1989.

ASMENQA2
ASME NQA-2a-1990, "Quality Assurance Requirements for Nuclear Facility Applications," The American Society of Mechanical Engineers, November 1990.

ASMENQA3
ASME NQA-3-1989, "Quality Assurance Program Requirements for the Collection of Scientific and Technical Information for Site Characterization of High-Level Nuclear Waste Repositories," The American Society of Mechanical Engineers, March 23, 1990.

ASMESUPP
Supplement 17S-1, ASME NQA-1-1989, "Supplementary Requirements for Quality Assurance Records," The American Society of Mechanical Engineers.

ASQCA3
ANSI/ASQC A3-1987, "Quality Systems Terminology," American Society or Quality Control, 1987.

BOEING
"(DRAFT) BA&E (Boeing Aerospace and Electronics) System Safety Instruction - System Safety Engineering in Software Development," The Boeing Company, 11/11/89.

BSI89
"89/97714-Guide to the Assessment of Reliability of Systems Containing Software," British Standards Institution, 12 September 1989.

CATEGORY
"Guideline for the Categorization of Software in Ontario Hydro's Nuclear Facilities with respect to Nuclear Safety," Revision 0, Nuclear Safety Department, June 1991.

CENSUS
"Programming Standards and Guidelines Manual," Bureau of the Census, March 27, 1991.

CSA89
CAN/CSA-Q396.1.2-89, "Quality Assurance Program for Previous Developed Software Used in Critical Applications," Canadian Standards Association, January 1989.

CSC003

CSC-STD-003-85, "Computer Security Requirements--Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments," Department of Defense, 25 June 1985.

DLP880

DLP880, "(DRAFT) Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations (based on IEC Standard 880)," David L. Parnas, Queen's University, Kingston, Ontario, March, 1991.

DOD2167A

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

DOT86

"Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Devices for Federal E.F.T. Use," United States Department of the Treasury, September 1, 1986.

ESA

ESA PSS-05-10, Issue 1, "Guide to Software Verification and Validation," European Space Agency, February 1994. **with ESA Guide to the Software Engineering Standards**

FAA026

FAA-STD-026, "National Airspace System (NAS) Software Development," U.S. Department of Transportation, Federal Aviation Administration, March 31, 1989.

FDA89

"(DRAFT) Reviewer Guidance for Computer-Controlled Devices," Medical Device Industry Computer Software Committee, January 1989.

FDA91

"Reviewer Guidance for Computer-Controlled Medical Devices Undergoing 510(k) Review," Office of Device Evaluation, Center for Devices and Radiological Health, Food and Drug Administration.

FIPS74

FIPS PUB 74, "Guidelines for Implementing and Using the NBS Data Encryption Standard," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1981 April 1.

FIPS81

FIPS PUB 81, "DES Modes of Operation," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1980 December 2.

FIPS101

FIPS PUB 101, "Guideline for Lifecycle Validation, Verification, and Testing of Computer Software," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1983 June 6.

FIPS106

FIPS 106, "Guideline on Software Maintenance," U. S. Department of Commerce/National Bureau of Standards (U.S.), 1984 June 15.

FIPS132

FIPS PUB 132, "Guideline for Software Verification and Validation Plans," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1987 November 19.[21]

FIPS140

FIPS PUB 140 FS 1027, "General Security Requirements for Equipment Using the Data Encryption Standard," General Services Administration, April 14, 1982.

FIPS461

FIPS 46-1, "Data Encryption Standard," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1988 January 22.

FIPS1401

FIPS 140-1, "Security Requirements for Cryptographic Modules," U.S. Department of Commerce/National Institute of Standards and Technology, 1990 May 2.

IEC880

IEC 880, "Software for Computers in the Safety Systems of Nuclear Power Stations," International Electrotechnical Commission, 1986.

IEC9126

ISO/IEC 9126, "Information Technology--Software Product Evaluation--Quality Characteristics and Guidelines for their Use," International Electrotechnical Commission, 1991-12-15.

IECSUPP

45A/WG-A3(Secretary)42, "(DRAFT) Software for Computers Important to Safety for Nuclear Power Plants as a Supplement to IEC Publication 880," International Electrotechnical Commission Technical Committee: Nuclear Instrumentation, Sub-Committee 45A: Reactor Instrumentation, Working Group A3: Data Transmission and Processing Systems, May 1991.

[21]See IEEE1012.

IECSUPP-94

45A/WG-A3(Secretary)48, "(DRAFT) Nuclear Power Plants - Instrumentation and Control Systems Important to Safety - First Supplement to IEC Publication IEC 880," IEC SC45A, May 1994.

IECTC56

IEC/TC56, "89/97714 - (DRAFT) Guide to the Assessment of Reliability of Systems Containing Software," British Standards Institution, 12 September 1989.

IECWG9'89

IEC/TC65A WG9, IEC 65A(Secretariat)94, "89/33006 DC - (DRAFT) Software for Computers in the Application of Industrial Safety-Related Systems," British Standards Institution, November 1989.

IECWG9'91

IEC/TC65A WG9, IEC 65A(Secretariat)122, "Software for Computers in the Application of Industrial Safety-Related Systems," Version 1.0, 26th September 1991.

IECWG10'89

IEC/TC65A WG10, "89/33005 DC - (DRAFT) Functional Safety of Programmable Electronic Systems," British Standards Institution, November 1989.

IECWG10'92

IEC/TC65A WG10, "(DRAFT) Functional Safety of Electrical/Electronic/Programmable Electronic Systems," 1992.

IECWG10'93

IEC/TC65A WG10, "(DRAFT) Functional Safety: Safety Related Systems

IEEE603

IEEE Std 603-1980, "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations," The Institute of Electrical and Electronics Engineers, Inc., November 24, 1980.

IEEE610

ANSI/IEEE Std 610.12-1990, "Glossary of Software Engineering Terminology," The Institute of Electrical and Electronics Engineers, Inc., February, 1991.

IEEE7301

ANSI/IEEE Std 730.1-1989, "IEEE Standard for Software Quality Assurance Plans," Institute of Electrical and Electronics Engineers, Inc., October 10, 1989.

IEEE730

ANSI/IEEE Std 730-1989, "IEEE Standard for Software Quality Assurance Plans," Institute of Electrical and Electronics Engineers, Inc., January 22, 1990.

IEEE828

ANSI/IEEE Std 828-1990, "IEEE Standard for Software Configuration Management Plans," Institute of Electrical and Electronics Engineers, Inc., February 15, 1991.

IEEE829

ANSI/IEEE Std 829-1983, "IEEE Standard for Software Test Documentation," Institute of Electrical and Electronics Engineers, Inc., August 19, 1983.

IEEE830-84

ANSI/IEEE Std 830-1984, "IEEE Guide to Software Requirements Specifications," Institute of Electrical and Electronics Engineers, Inc., July 29, 1984.

IEEE830-93

ANSI/IEEE Std 830, "(DRAFT) IEEE Recommended Practice for Software Requirements Specifications," Institute of Electrical and Electronics Engineers, Inc., 8/10/93.

IEEE982-1

ANSI/IEEE Std 982.1-1988, "IEEE Standard Dictionary of Measures to Produce Reliable Software," Institute of Electrical and Electronics Engineers, Inc., August 10, 1989.

IEEE982-2

ANSI/IEEE Std 982.2-1988, "IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software," Institute of Electrical and Electronics Engineers, Inc., August 10, 1989.

IEEE983

ANSI/IEEE Std 983-1986, "IEEE Guide for Software Quality Assurance Planning," Institute of Electrical and Electronics Engineers, Inc., February 20, 1986.

IEEE990

ANSI/IEEE Std 990-1987, "IEEE Recommended Practice for Ada As a Program Design Language," Institute of Electrical and Electronics Engineers, Inc., October 1, 1987.

IEEE1002

ANSI/IEEE Std 1002-1987, "IEEE Standard Taxonomy for Software Engineering Standards," Institute of Electrical and Electronics Engineers, Inc., June 4, 1987.

IEEE1008

ANSI/IEEE Std 1008-1987, "IEEE Standard for Software Unit Testing," Institute of Electrical and Electronics Engineers, Inc., July 28, 1986.

IEEE1012

ANSI/IEEE Std 1012-1986, "IEEE Standard for Software Verification and Validation Plans," The Institute of Electrical and Electronics Engineers, Inc., February 10, 1987.[22]

[22]Adopted by the Federal government as FIPS PUB 132.

IEEE1016

ANSI/IEEE Std 1016-1987, "IEEE Recommended Practice for Software Design Descriptions," Institute of Electrical and Electronics Engineers, Inc., October 6, 1987.

IEEE1028

ANSI/IEEE Std 1028-1988, "IEEE Standard for Software Reviews and Audits," Institute of Electrical and Electronics Engineers, Inc., June 29, 1989.

IEEE1042

ANSI/IEEE Std 1042-1987, "IEEE Guide to Software Configuration Management," Institute of Electrical and Electronics Engineers, Inc., March 10, 1988.

IEEE1058

ANSI/IEEE Std 1058-1987, "IEEE Standard for Software Project Management Plans," Institute of Electrical and Electronics Engineers, Inc., October 6, 1988.

IEEE1074

ANSI/IEEE Std 1074-1991, "IEEE Standard for Developing Software Lifecycle Processes," The Institute of Electrical and Electronics Engineers, Inc., 1991.

IEEE7432

ANSI/IEEE Std 7432-1993, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," The Institute of Electrical and Electronics Engineers, Inc., 1993.

IEEE1063

ANSI/IEEE Std 1063-1987, "IEEE Standard for Software User Documentation," Institute of Electrical and Electronics Engineers, Inc., February 2, 1989.

IEEE1228

IEEE Std 1228-1994, "IEEE Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, August 9, 1994.

IEEEP1059

IEEE Std P1059-199X, "(DRAFT 7.1) IEEE Guide for Software Verification and Validation Plans," Institute of Electrical and Electronics Engineers, May 24, 1993.

IEEEP1228-C

P1228, "(DRAFT C) Draft Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, November 13, 1990.

IEEEP1228-D

P1228, "(DRAFT D) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., March 6, 1991.

IEEEP1228-E

P1228, "(DRAFT E) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., July 19, 1991.

IEEEP1228-G

P1228, "(DRAFT G) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., 1/14/92.

IEEEP1228-H

P1228, "(DRAFT H) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., 4/27/92.

IEEEP1228-J

P1228, "(DRAFT J) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., 2/11/93.

IEEEGUIDE

"Guide to Software Design Descriptions," Institute of Electrical and Electronics Engineers, 1993.

IEEETEST

"(DRAFT) Guidelines for Assuring Testability," The Institution of Electrical Engineers, May 1987.

IFIP104

IFIP WG 10.4, "Dependability: Basic Concepts and Terminology," IFIP Working Group on Dependable Computing and Fault Tolerance, October 1990.

ISASP84

ISA-SP84, Draft 10, "(DRAFT) Programmable Electronic Systems (PES) for use in Safety Applications," Instrument Society of America, August 1992.

ISO9000

ISO 9000, "International Standards for Quality Management," May 1990.

ISO12207

ISO/IEC DIS 12207-1, "(DRAFT) Information Technology--Software--Part 1: Software Life Cycle Process," International Electrotechnical Commission, 1994.

ITSEC89

ITSEC 1.1989, "Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems," GISA - German Information Security Agency, 1989.

ITSEC90

ITSEC 1.1990, "(DRAFT) Information Technology Security Evaluation Criteria (ITSEC)," Harmonised Criteria of France-Germany-the Netherlands-the United Kingdom, 02 May 1990.

JPL93

JPL D-10058, "Software Systems Safety Handbook," PREPARED BY Jet Propulsion Laboratory FOR National Aeronautics and Space Administration, May 10, 1993,

MIL347

MIL-HDBK-347, "Mission-Critical Computer Resources Software Support," Department of Defense, 22 May 90.

MIL498

MIL-STD-498 (DRAFT), "Software Development and Documentation," Department of Defense, 30 November 1994.

MIL882B

MIL-STD-882B, "System Safety Program Requirements," Department of Defense, 30 March 1984.

MIL882C

MIL-STD-882C, "Systems Safety Program Requirements," Department of Defense, DISTRIBUTION STATEMENT A.

MIL1521B

[Proposed Updates to] MIL-STD-1521B, "Technical Reviews and Audits for Systems, Equipments, and Computer Software," Logicon Input to the JLC/CSM, June 16, 1989.

MILSDD

MIL-STD-SDD, "(DRAFT) Software Development and Documentation," Department of Defense, 22 December 1992.

MILSWM

MIL-HDBK-SWM (DRAFT), "Software Measurement Selection and Use," Department of Defense, 14 January 1994.

MOD0055'89

Interim Defence Standard 00-55, "(DRAFT) Requirements for the Procurement of Safety Critical Software in Defence Equipment," Ministry of Defence, UK, May 1989.

MOD0055'91

Interim Defence Standard 00-55, "The Procurement of Safety Critical Software in Defence Equipment," Parts 1 and 2, Ministry of Defence, UK, 5 April 1991.

MOD0056'89

Interim Defence Standard 00-56, "(DRAFT) Requirements for the Analysis of Safety Critical Hazards," Ministry of Defence, UK, May 1989.

MOD0056'91

Interim Defence Standard 00-56, "Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment," Ministry of Defence, UK, 5 April 1991.

NASAMGMT

"Management Plan Documentation Standard and Data Item Descriptions (DID)," NASA, 2/28/89.

NASAPROD

"Product Specification Documentation Standard and Data Item Descriptions (DID)," NASA, 2/28/89.

NCSC005

NCSC-TG-005, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria," National Computer Security Center, 31 July 1987.

NCSC021

NCSC-TG-021, "Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria," National Computer Security Center, April 1991.

NIST165

NIST Special Publication 500-165, "Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards," U.S. Department of Commerce/National Institute of Standards and Technology, September 1989.

NIST190

NIST Special Publication 500-190, "Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991," U.S. Department of Commerce/National Institute of Standards and Technology, August 1991.

NIST204

NIST Special Publication 500-204, "High Integrity Software Standards and Guidelines," U.S. Department of Commerce/National Institute of Standards and Technology, September 1992.

NIST209

NIST Special Publication 500-209, "Software Error Analysis," U.S. Department of Commerce/National Institute of Standards and Technology, April 1993.

NIST213

NIST Special Publication 500-213, "Next Generation Computer Resources: Reference Model for Project Support Environments (Version 2.0)," U.S. Department of Commerce/National Institute of Standards and Technology, November 1993.

NIST216

NIST Special Publication 500-216, "Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop (NUREG/CP 0136)," U.S. Department of Commerce/National Institute of Standards and Technology, March 1994.

NIST626

NIST GCR 93/626, "An International Survey of Industrial Applications of Formal Methods Volume 1 Purpose, Approach, Analysis, and Conclusions," U.S. Department of Commerce/National Institute of Standards and Technology, March 1993.

NIST4909

NISTIR 4909, "Software Quality Assurance: Documentation and Reviews," U.S. Department of Commerce/National Institute of Standards and Technology, September 1992.

NPR6300

NPR-STD-6300, "Management of Scientific, Engineering and Plant Software," Office of New Production Reactors, U.S. Department of Energy, March 1991.

NSA8616

NSA Spec. 86-16, "Security Guidelines for COMSEC Software Development," National Security Agency, 10 July 1986.

NSS1740

NSS 1740.13, "(Interim) NASA Software Safety Standard," National Aeronautics and Space Administration, June 1994.

NSWC8933

NSWC TR 89-33, "Software Systems Safety Design Guidelines and Recommendations," Naval Surface Warfare Center, March 1989.

NUREG6018

NUREG/CR-6018, "Survey and Assessment of Conventional Software Verification and Validation Methods," U.S. Nuclear Regulatory Commission, April 1993.

NUREG6101

NUREG/CR-6101 & UCRL-ID-114839, "Software Reliability and Safety in Nuclear Reactor Protection Systems," U.S. Nuclear Regulatory Commission, June 11, 1993.

PES87

"Programmable Electronic Systems in Safety Related Applications," Parts 1 and 2, Health and Safety Executive, 1987.

RTCA178A

RTCA/DO-178A, "Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, March, 1985.

RTCA178B
RTCA/DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," RTCA, Inc., June 29, 1993.

SAFEIT
"SafeIT," Volumes 1 and 2, Interdepartmental Committee on Software Engineering, June 1990.

SOFTENG
"Standard for Software Engineering of Safety Critical Software," Rev. 0, Ontario Hydro, December 1990.

SOFTENG2
"Software Engineering of Category II Software," Rev. 00, Ontario Hydro, 1993 05.

TCSEC
DOD 5200.28-STD, "Department of Defense Trusted Computer System Evaluation Criteria," Department of Defense, December 1985.

UL1998
UL 1998, "The Proposed First Edition of the Standard for Safety-Related Software," Underwater Laboratories, August 17, 1990.

USEREXP
"User Expectations and Requirements for Software Engineering Standards (Discussion Draft), Software Engineering Standards Long-Range Planning Study Group, November 22, 1991.

WL-1037
WL-TR-1037, "Evaluation and Validation Reference Manual," Version 3.0, Wright Laboratory, Wright-Patterson AFB, Ohio, May 1991.

WL-1038
WL-TR-1038, "Evaluation and Validation Guidebook," Version 3.0, Wright Laboratory, Wright-Patterson AFB, Ohio, May 1991.

A.2 Books

BEIZER
Beizer, Boris, Software Testing Techniques, Van Nostrand Reinhold, New York, 1990.

EWICS1
Redmill, F. J. (ed.), Dependability of Critical Computer Systems 1, Elsevier Science Publishers LTD, 1988.

EWICS2

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Elsevier Science Publishers LTD, 1989.

EWICS3

Bishop, P. G. (ed.), Dependability of Critical Computer Systems 3 - Techniques Directory, Elsevier Science Publishers LTD, 1990.

MUSA1

Musa, J.D., A. Iannino, and K. Okumoto, Software Reliability, Measurement, Prediction, Application, McGraw-Hill, New York, 1987.

RAHEJA91

Raheja, Dev G., Assurance Technologies - Principles and Practices, McGraw-Hill, Inc., 1991.

WILEY

Encyclopedia of Software Engineering, John Wiley & Sons, Inc., 1994.

A.3 Papers

BELL

Bell, R. and S. Smith, "An Overview of IEC Draft Standard: Functional Safety of Programmable Electronic Systems."

BUTLER

Butler, R. and G. Finelli, "The Infeasibility of Experimental Quantified Life-Critical Software Reliability," *Proceedings of SIGSOFT'91: Software for Critical Systems*, Association for Computing Machinery, December 1991.

FUJII1

Fujii, Roger U., "Software Engineering For Instrumentation and Control," American Nuclear Society, *Nuclear Plan Instrumentation, Control, and Man-Machine Interface Technologies, Oak Ridge, TN*, April 1993.

HANSEN

Hansen, Mark D., "Survey of Available Software-Safety Analysis Techniques," Annual Reliability and Maintainability Symposium - 1989 Proceedings, 1989.

JOANNOU

Joannou, P.K., J. Harauz, D.R. Tremaine, N. Ichiyen, A.B. Clark, "The Canadian Nuclear Industry's Initiative in Real-Time Software Engineering," Ontario Hydro and AECL CANDU, Ontario, Canada.

JUNK

Junk, William S., "Annotated Bibliography - Software Safety," April 24, 1990.

LEVESON83

Leveson, Nancy G. and Peter R. Harvey, "Analyzing Software Safety," *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 5, September 1983.

LEVESON86

Leveson, N.G., "Software Safety: Why, What, and How," *Computing Surveys*, Vol. 18, No. 2, June 1986.

LEVESON87

Leveson, Nancy G., Janice L. Stolzy, "Safety Analysis Using Petri Nets," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 3, March 1987.

LEVESON89

Leveson, Nancy, "Software Safety," Presentation to IEEE Software Safety Working Group, October 1989.

LEVESON91

Leveson, Nancy G., Stephen S. Cha and Timothy J. Shimeall, "Safety Verification of Ada Programs Using Software Fault Trees," *IEEE Software*, July 1991.

LEVESON92

Leveson, Nancy G. and Clark S. Turner, "An Investigation of the Therac-25 Accidents," University of California, Irvine, CA, November 1992.

LEVINSON

Levinson, Stanley H. and H. Tazewell Daughtrey, "Risk Analysis of Software-Dependent Systems," Probabilistic Safety Assessment International Topical Meeting, Clearwater Beach, FL, January 1993.

MUSA2

Musa, J.D., and A.F. Ackerman, "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, May 1989.

PETERSON

Peterson, James L., "Petri Nets," *Computing Surveys*, Vol. 9, No. 3, September 1977.

SESAW91-1

DeWalt, Michael P., "Comparison of FAA DO-178A and DOD-STD-2167A Approaches to Software Certification," Software Engineering Standards Application Workshop sponsored by IEEE Computer Society, San Francisco, May 1991.

SESAW91-2

Sanz, Julio Gonzalez, "Standardization for Safety Software: Current Status and Perspectives," Software Engineering Standards Application Workshop sponsored by IEEE Computer Society, San Francisco, May 1991.

SESAW91-3

Wright, Cynthia L. and Anthony J. Zawilski, "Existing and Emerging Standards for Software Safety," Software Engineering Standards Application Workshop sponsored by IEEE Computer Society, San Francisco, May 1991.

TYSZER

Tyszer, J., P. Parent, J. Rajski and V. K. Agarwal, "The Hierarchical Description of Stochastic Petri Nets," Department of Electrical Engineering, McGill University.

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NIST Technical Publications

Periodical

Journal of Research of the National Institute of Standards and Technology—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published bimonthly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program in support of the efforts of private-sector standardizing organizations.

Order the **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NIST Interagency Reports (NISTIR)—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and nongovernment). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Institute of Standards and Technology
Gaithersburg, MD 20899-0001