NUREG/CR-5930
NIST SP 500-204

High Integrity Software Standards and Guidelines

Manuscript Completed: May 1992
Date Published: July 1992

Prepared by
Dolores R. Wallace, Laura M. Ippolito, D. Richard Kuhn

Systems and Software Technology Division
Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Prepared for

U.S. Nuclear Regulatory Commission
Washington, DC 20555
NRC FIN # L19762

ABSTRACT

This report presents results of a study of standards, draft standards, and guidelines (all of which will hereafter be referred to as documents) that provide requirements for the assurance of software in safety systems in nuclear power plants. The study focused on identifying the attributes necessary in a standard for providing reasonable assurance for software in nuclear systems. The study addressed some issues involved in demonstrating conformance to a standard. The documents vary widely in their requirements and the precision with which the requirements are expressed. Recommendations are provided for guidance addressing the assurance of high integrity software. It is recommended that a nuclear industry standard be developed based on the documents reviewed in this study with additional attention to the concerns identified in this report.

TABLE OF CONTENTS

TABLES

Table 1-1 Review documents 2
Table 1-2 Criteria documents 3
Table 1-3 Criteria template 4

EXECUTIVE SUMMARY

Under the interagency agreement RES-91-003 between the United States Nuclear Regulatory Commission (NRC) and the National Institute of Standards and Technology (NIST), NIST is responsible for providing NRC with current information regarding standards and practices for assuring safety of the software for safety systems in nuclear power plants. This report presents the results of a study that is defined by the following:

> Review current United States (US) and international standards and guidelines for high integrity software and analyze their potential for use in the nuclear industry with regard to safety applications. Some documents were selected by NRC and forwarded to NIST for review.

High integrity software is software that 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. Examples include software used in safety systems of nuclear power plants, medical devices, electronic banking, air traffic control, automated manufacturing, and military systems. Standards, draft standards, and guidelines (hereafter referred to as documents) from each of these fields were examined in this study. The documents reviewed included those recommended by NRC and a few others that collectively reflect the trends of the standards community for high integrity software. Comprehensive documents that address more than one software engineering process were examined. From over 50 documents, ten were selected for further study. Most of these documents have broad scopes and provide guidance across the entire software lifecycle for several software engineering processes. Documents of narrower scope (i.e., single topic) were used to identify criteria for specific software engineering practices and processes that should be addressed by standards for high integrity software.

The following principal questions guided the review of the documents:

- Do the requirements of the document provide assurance of the nuclear safety system software developed, maintained, and operated according to these requirements? That is, does the document contain requirements for building verifiably dependable nuclear safety system software?

- Will the requirements of the document provide NRC auditors with enough information to verify that the vendor product is in compliance with the requirements and are there clearly measurable conformance clauses? Is it expected that two auditors will arrive at the same conclusions relative to the product?

The focus of this study is on software assurance issues (i.e., assurance that the software meets safety requirements). Usually, the software will be embedded in a system in which software is only one component. Although the software lifecycle, not the system lifecycle, is the focus, the software lifecycle's relationship to the system lifecycle must be understood in order to place assurance activities in the proper context. System activities are not included in the review criteria beyond the extent necessary to evaluate software assurance requirements. While the documents in this study were reviewed relative to the assurance of software for safety systems in nuclear power plants, the criteria and recommendations contained in this report are applicable to other critical software safety systems.

Criteria for answering the principal questions should be derived from a body of knowledge or rigorous technical specifications for software engineering, especially specifications that are essential in building high integrity software. For software engineering, the body of knowledge has not yet been rigorously codified but is distributed among standards, guidelines, technical reports, conference presentations, and information proprietary to organizations. Few documents reviewed in this study reflect the growth of the knowledge base found in technical reports and conference presentations, which are not subjects of this review. The criteria used for this study are based on the authors' experience and knowledge derived from these various sources of information.

Documents were reviewed against a template (see Appendix A) of detailed criteria in several categories: levels of criticality/assurance, lifecycle phases, documentation, required software functionality against hazards, engineering practices for the development of software, project planning and management, and assurance activities which include software verification and validation, software quality assurance, software configuration management, and software hazard analysis. The template also includes procurement issues. The template's Presentation category provided a mechanism for objectivity in identifying how well an auditor can demonstrate compliance of a system with a specific document's requirements.

Each document was analyzed according to the criteria to answer the principal questions. The relative strengths and weaknesses of the documents were compared. No single standard or guideline reviewed for this study completely satisfies all the evaluation criteria, although most documents satisfy requirements for at least one category of criteria in the template.

No standard can guarantee the safety of a particular software system. In other words, no one can ever say "If a developer follows this standard, the system will be safe." The judgement of safety for a particular installation must be made by the appropriate authorities. Ideally, standards should state what is required for evaluating the safety of a system, and to determine if the software complies with the standard. That is, "These are the things that must be done to enable NRC to judge whether the system is safe enough to not pose an undue risk to public health and safety."

It is recommended that NRC considerations of consensus standards, draft standards, and guidelines include the concerns identified in this report. While information from all of the documents can be used in developing a rigorous standard, other concerns must be addressed. For example, the scope of the standard within system and software lifecycles must be clearly identified. The standard should require that software is always assured in the context of the particular system under evaluation. Guidance for assurance of high integrity software should either describe all practices or cite acceptable standards for them, encompassing at least the criteria in Appendix A.

ACKNOWLEDGEMENTS

Leo Beltracchi of the United States Nuclear Regulatory Commission (NRC) provided substantial guidance to the authors of this report. Franklin Coffman and Julius Persensky, NRC, also contributed to this report.

ABBREVIATIONS

Acronyms used in this report (other than those used to refer to documents) are listed below.

CASE	Computer Aided Software Engineering
CM	Configuration Management
DID	Design Input Documentation
FMEA	Failure Modes and Effects Analysis
FTA	Fault Tree Analysis
IV&V	Independent Verification and Validation
O&M	Operation and Maintenance
QA	Quality Assurance
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SDD	Software Design Description
SDP	Software Development Plan
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SRS	Software Requirements Specification
SSP	Software Safety Plan
SV&V	Software Verification and Validation
SVVP	Software Verification and Validation Plan
V&V	Verification and Validation

1. INTRODUCTION

Under the interagency agreement RES-91-003 between the United States Nuclear Regulatory Commission (NRC) and the National Institute of Standards and Technology (NIST), NIST is responsible for providing NRC with current information regarding standards and practices for assuring safety of the software for safety systems in nuclear power plants. This report presents the results of a study that is defined by the following:

> Review current United States (US) and international standards and guidelines for high integrity software and analyze their potential for use in the nuclear industry with regard to safety applications. Some documents were selected by NRC and forwarded to NIST for review.

While the documents in this study were reviewed relative to the assurance of software for safety systems in nuclear power plants, the criteria and recommendations contained in this report are applicable to other critical software safety systems.

1.1. Standards and Guidelines Reviewed

This report contains the results of a review of current US and international standards and guidelines (hereafter referred to as documents) for high integrity software and their analysis relative to potential use in the nuclear industry with regard to safety applications. High integrity software is software that 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, medical devices, electronic banking, air traffic control, automated manufacturing, and military systems [NIST190]. Documents from each of these fields were examined in this study including those recommended by NRC and a few others that collectively reflect the trends of the standards community for high integrity software.

Comprehensive documents that address more than one software engineering process were examined. From over 50 documents, ten were selected for further study. One of these is a book of five chapters produced by a European committee[1]; this study considered each of the five chapters as a separate document. Therefore, 14 documents were reviewed. The titles of these documents and their respective acronyms used in this report are in Table 1-1. Complete references are in Section 5.

Most of these documents have broad scopes and provide guidance across the entire software lifecycle for several software engineering processes. Documents of narrower scope (i.e., single topic) were used to identify criteria for specific software engineering practices and processes that should be addressed by standards for high integrity software. Titles and acronyms of documents that were used to provide additional criteria are listed in Table 1-2. Complete references are in Section 5.

Table 1-3 lists the categories of criteria used to respond to the principal questions that were derived from the documents listed in Table 1-2.

[1]The other books [EWICS1, EWICS3] produced by the same committee are NOT reviewed in this study.

Acronym	**Number and title** (complete references are in Section 6)
ANS7432	ANSI/IEEE-ANS-7-4.3.2-1982: Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations
CATEGORY	Guideline for the Categorization of Software in Ontario Hydro's Nuclear Facilities with respect to Nuclear Safety, Revision 0, 1991
DLP880	DLP880: (DRAFT) Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations (based on IEC Standard 880)
EWICS2-1	Dependability of Critical Computer Systems 2, Chapter 1: Guidelines to Design Computer Systems for Safety, 1989
EWICS2-2	Dependability of Critical Computer Systems 2, Chapter 2: Guidelines for the Assessment of the Safety and Reliability of Critical Computer Systems, 1989
EWICS2-3	Dependability of Critical Computer Systems 2, Chapter 3: A Questionnaire for System Safety and Reliability Assessment, 1989
EWICS2-4	Dependability of Critical Computer Systems 2, Chapter 4: A Guideline on Software Quality Assurance and Measures, 1989
EWICS2-5	Dependability of Critical Computer Systems 2, Chapter 5: Guidelines on the Maintenance and Modification of Safety-Related Computer Systems, 1989
IEC880	IEC 880: Software for Computers in the Safety Systems of Nuclear Power Stations, 1986
IECSUPP	45A/WG-A3(Secretary)42: (DRAFT) Software for Computers Important to Safety for Nuclear Power Plant, 1991
NPR6300	NPR-STD-6300: Management of Scientific, Engineering and Plant Software, 1991
P1228	P1228: (DRAFT) Standard for Software Safety Plans (IEEE Working Group), 1991
RTCA178A	RTCA/DO-178A: Software Considerations in Airborne Systems and Equipment Certification, 1985
SOFTENG	Standard for Software Engineering of Safety Critical Software, Rev. 0, 1990

Table 1-1 Review documents

Acronym	**Number and title** (complete references are in Section 6)
ANS104	ANSI/ANS-10.4-1987: Guidelines for the Software Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry
ASMENQA2	ASME NQA-2a-1990: Quality Assurance Requirements for Nuclear Facility Applications
FIPS101	FIPS 101: Guideline for Lifecycle Validation, Verification, and Testing of Computer Software, 1983
FIPS132	FIPS 132: Guidelines for Software Verification and Validation Plans, 1987
FIPS1401	FIPS 140-1: Security Requirements for Cryptographic Modules, 1990
IEEE828	ANSI/IEEE Std 828-1983: IEEE Standard for Software Configuration Management Plans
IEEE1012	ANSI/IEEE Std 1012-1986: IEEE Standard for Software Verification and Validation Plans
IEEE1058	ANSI/IEEE Std 1058.1-1987: IEEE Standard for Software Project Management Plans
IEEE7301	ANSI/IEEE Std 730.1-1989: IEEE Standard for Software Quality Assurance Plans
ISO9000	ISO 9000: International Standards for Quality Management, 1990
ITSEC	ITSEC 1.1989: Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems
MOD0055	Interim Defence Standard 00-55: The Procurement of Safety Critical Software in Defence Equipment, 1991
NIST180	NIST Special Publication 500-180: Guide to Software Acceptance, 1990
NIST190	NIST Special Publication 500-190: Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991
SAFEIT	SafeIT, Department of Trade and Industry, 1990
TCSEC	DoD 5200.28-STD: Department of Defense Trusted Computer System Evaluation Criteria

Table 1-2 Criteria documents

Levels of Criticality/Assurance

Lifecycle Phases

Documentation

Required Software Functionality Against Hazards

Software Engineering Practices

Assurance Activities

- Software Verification and Validation (SV&V)
- Software Quality Assurance (SQA)
- Software Configuration Management (SCM)
- Software Hazard Analysis

Project Planning and Management

Procurement Concerns

Presentation

Supplemental Information

General Comments

Table 1-3 Criteria template

Section 2 of this report provides a description of the criteria for each category in this template, that is, the detailed requirements. Section 3 contains a comparison of the documents for each of the review criteria. Section 4 contains the summary of this study. Appendix A identifies some of the features against which each document was analyzed. Appendix B contains the review of each document based on the criteria.

2. REVIEW CRITERIA

This study required a review of documents potentially useful for assuring high integrity software in safety systems of nuclear power plants. The focus of this study was on software assurance issues (i.e., assurance that the software meets safety requirements). Usually, the software will be embedded in a system in which software is only one component. Although the software lifecycle, not the system lifecycle, is the focus, the software lifecycle's relationship to the system lifecycle must be understood in order to place assurance activities in the proper context. In general, system activities are not included in the review criteria beyond the extent necessary to evaluate software assurance requirements.

The documents were reviewed according to criteria developed to answer the following principal questions:

- Do the requirements of the document provide assurance of the nuclear safety system software developed, maintained, and operated according to these requirements? That is, does the document contain requirements for building verifiably dependable nuclear safety system software?

- Will the requirements of the document provide NRC auditors with enough information to verify that the vendor product is in compliance with the requirements, and are there clearly measurable conformance clauses? If so, is it expected that two auditors will arrive at the same conclusions relative to the product?

To meet the first set of questions, the software system must be of high integrity. To build a high integrity system, developers, assurers, and customers need a body of knowledge or technical specifications for high integrity software. For many engineering disciplines, that body of knowledge may be found in handbooks and rigorous standards. For software engineering, the body of knowledge has not been rigorously codified. Software engineering knowledge is distributed among standards, guidelines, technical reports, conference presentations, and information proprietary to organizations. It is important to understand this distinction between software engineering and older engineering disciplines. Due to this lack of codified information, this review was based on criteria which reflect the authors' knowledge and experience derived from various sources of information, including technical reports, conference proceedings, and related standards. NIST has published two such technical reports that address high integrity software issues. One identifies the range of requirements (e.g., functional, interface, performance, security, safety, and quality) for inclusion in defining a software system [NIST180[2]]. The other report identifies current practices and issues in developing and assuring high integrity systems [NIST190].

The second set of principal questions deals with the customer and is designed so that the requirements of the documents would be interpreted objectively. The customer must be able to determine that the software system has met the requirements. A standard serves as a measuring tool to apply to the software. The customer must be able to use the standards and guidelines to determine how the software satisfies the specific contractual requirements of the software. Another important point considered was whether or not the document clearly defines what conformance to the document means. The presentation of a document heavily influenced the answers to this set of questions.

[2]Acronyms for documents used in this section are defined in Table 1-2.

The template in Appendix A summarizes some of the criteria used to review the documents. Other criteria are based on standards for specific processes, technical articles, and staff experience. The rationale for the criteria are described in the remainder of Section 2.

2.1. Levels of Criticality/Assurance

Computer software is used to drive many types of systems (e.g., air traffic control, medical devices, rail transportation, nuclear power systems, and software systems that stand alone). Failure of these systems may have different consequences, which can be assigned a level of seriousness or criticality. The most serious of these consequences is usually considered to be loss of life and is assigned the highest level of criticality. For these systems, the most rigorous software standards and practices must be invoked. Other levels of criticality take into account how serious a failure is relative to the completion of the task the system is responsible for, and how devastating the failure may be relative to destruction of property and environment, injuries, and other losses. While software systems used for safety systems in nuclear power plants are at the highest level, not all of the documents reviewed in this study are written at this level. A trend observed after examining many documents is that documents identify requirements according to levels of criticality not only for the principal system but for its support software and for software used to develop and assure it. Other factors (e.g., new technology, importance of mission, project size) may affect the needed level of assurance, but in this report the concern is the criticality level based on the consequences of failure.

2.2. Lifecycle Phases

For this study, the scope of each document relative to the software lifecycle is identified. While there are many representations of the software lifecycle and names for lifecycle phases, this report uses the following names to represent the lifecycle phases:

Initiation
Requirements
Design
Code (Implementation)
Integration and Test
Installation
Operation and Maintenance (O&M)

The phases are used only to help in identifying the scope of each document; this study was not concerned with a particular lifecycle management (e.g., waterfall, spiral). While some activities in software development and assurance need to be performed at certain times in the lifecycle (e.g., system test planning during requirements), no judgments are made on lifecycle management. The priorities centered on the activities themselves and how well a document addresses a particular phase. Therefore, it is not necessarily undesirable if a document addresses only some phases. For completeness, assurance of a software system relies on all aspects of software; that includes the entire lifecycle. By identifying documents that address partial lifecycles, it may become clear which documents, or parts of them, may be used together.

In standards dealing with the system lifecycle, it is sometimes difficult to know when a requirement is imposed on the software, or takes effect only after integration of the software with system components (e.g., configuration management (CM)). The focus of this study was on the set of activities that should be levied on the software.

2.3. Documentation

Documentation of a software system serves many purposes, including the following:

- Developer uses documentation to understand the software system, to enhance teamwork among the developer's staff, to enable continuity of development if staff leave, and/or to control changes during development.
- Customer needs documentation to use the system, to decide if a change would be reasonable, to implement a change, and/or to verify that the delivered product is the requested product.
- Auditor uses the documentation to perform an audit of the system relative to its contract and any standards the vendor was to have used.
- Assurers (e.g., reviewers, verifiers, testers) need documentation not only to perform their reviews and audits but also as the information on which tests are based.

This study uses a small set of documentation (based on an examination of many documents) as a baseline for review. The list does not include documentation for software quality assurance (SQA), software CM (SCM), and software verification and validation (SV&V) because these topics are addressed in other sections of the template. This study concentrated on *software* documentation. However, system requirements may be included with software documentation when the system requirements specification levies requirements on the software. The following types of questions were considered:

- How thorough are the document's requirements for specific documentation?
- Does the document specify the content that must be described in documentation? Or, does it specify the description of elements of the content?
- Does the document provide a quantified description of attributes that should be present in the documentation (e.g., rules for maintainability, consistency)?

A standard may be used by a vendor to build a product and by an auditor to verify compliance to the standard. A useful documentation tool that aids both the vendor and auditor is a checklist of the requirements.

2.4. Required Software Functionality Against Hazards

Critical systems require defensive functions to prevent unintended functions and to allow operation to continue

despite errors and component failures. At the system level, a hazard analysis provides information concerning the types of hazards or threats that could adversely affect system behavior. From the system hazard analysis, some software functions may be identified to prevent hazards or to mitigate the effect of problems. Additionally, a software hazard analysis may be conducted to reveal other functionality. Special software functions are often included to detect, tolerate, override, or recover from failures. Examples include the use of prompts that query the operator as to whether or not a keyed-in command should actually be executed, and the use of software devoted to error detection and correction (e.g., telephone switching systems where, typically, 50% or more of the software serves this function). Defensive functions should be a consideration in a standard for safety critical software.

Some special software functions which may be generic to any system concerned with software safety or computer security are listed in Appendix A. Not all of the functions are essential in all systems, but an auditor should look for the use of these functions, or reasons why they are not needed in a particular system. The list is not necessarily complete nor does reference to such a list eliminate the need for system and software hazard analyses to identity software functions for a specific system.

2.5. Software Engineering Practices

Certain software engineering practices can either contribute or detract from the safety and reliability of a system. For example, systems constructed from modules that each perform a single, well-defined function are likely to be more reliable than those where modules perform a mixture of functions (e.g., both control and data input/output). Choice of programming language is another example. Systems written in assembly language tend to be much more error-prone than those developed in modern high-level languages such as Pascal, C, and Ada [NRC91]. The National Research Council has recommended the use of simple modules and high-level languages [NRC91].

A standard for safety critical software should give guidance on software engineering practices that contribute to high integrity. Rigid rules for the use of specific software engineering practices are not always appropriate. In some cases, it may not be possible to develop software for a particular application in a way that is normally considered good software engineering practice in other applications. For example, it is normally considered good practice to separate critical functions from the rest of the system, placing critical functions in a small module that can be more readily analyzed than a large system. It has been argued that some safety critical systems cannot be built in this way because, in some systems, safety concerns are present in all functions.

However, it is essential that developers use established and acceptable practices as much as possible, and explain cases where established practices were not used. Section A.5 lists practices for consideration in all projects, even if not all techniques are appropriate for all projects. The major types of software engineering practices considered in this review are explained below.

Specifications - A good specification must be clear, unambiguous, and analyzable, so that flaws can be found before the system is built. Specifications can be characterized as informal (English text and block diagrams), semi-formal (pseudo code or "structured English" plus standardized notation such as structure charts and data-flow diagrams), or formal (mathematics such as logic and set theory). Formal specifications make it possible to do the kind of rigorous mathematical design analysis that is conducted in traditional engineering. With less formal specifications, this is not possible; thus formality in specifications is desirable.

Component isolation - Ideally, components that are critical to system safety should be isolated from other parts of the system. Safety functions should be kept separate from one another.

Modularity - All systems should be constructed in a modular fashion, where each module performs a single, well-defined function.

Language - Programmers are more likely to make errors using assembly languages than in using high-level languages such as Ada, Pascal, Fortran and C [NRC91]. The tradeoff may be that the high-level language compiler itself may have errors but the compiler should have the same level of software assurance as the software systems that will be built with it. The extra expense of assurance for the compiler will be offset by reduced expense in assuring one or more software systems built with it. Many reasons for using high-level languages are cited in NRC91. Standards should encourage the use of high-level languages and discourage assembly language, except where absolutely necessary.

Deprecated programming practices - Many programming practices, such as floating point arithmetic or use of interrupts, result in systems whose behavior is difficult to analyze. Standards for critical software often discourage such practices in order to make the software evaluation more effective [MOD0055].

Quality Attributes - While many standards require SQA activities to check for quality of software products, few specify what the qualities should be, or specify requirements for those qualities. Standards should identify quality attributes and definitions of those attributes. For example, some of the documents require that a specification be "complete." Because "completeness" can be ambiguous, it is important to clearly state the rules of completeness. For example, one rule may be that a document require the types of requirements to be defined (e.g., functional, performance, safety, security, quality, and interface requirements from the software to the system).

2.6. Assurance Activities

This section of the template identifies the activities that will locate problems (e.g., errors, faults) in the development process and products, and will provide evidence that the software complies with its specifications. These activities are performed from the beginning of the software lifecycle, through development, to maintenance of software. Results from the activities may affect the software requirements, design, or code, and sometimes the system itself. The template in Appendix A addresses activities performed during the software lifecycle and, in some cases, these activities require as input the system requirements specification and reports of system hazard analyses. It should be noted that this template does not address activities of assurance that are specific to the final evaluation of a product before delivery (e.g, acceptance test, final audit, system validation).

2.6.1. Software Verification and Validation (SV&V)

Software verification is "the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase." Software validation is "the process of evaluating software at the end of the software development process to ensure compliance with software requirements." [IEEE1012] The definitions for *verification* and *validation* may be different in the system context than they are in the software context. In the system context, the activities of verification are separate from those of validation, which is performed on the completed system. In the software context, the distinctions between verification and validation (V&V) may be less clear. Within the full system lifecycle, SV&V may occur between system requirements specification and system validation. All testing of the software is included under SV&V. That part of system validation that exercises software is also the final step of SV&V. Final system validation is not included as part of this study because the scope of the documents under examination does not cover this step.

There are two standards which together have comprehensive requirements for SV&V. These are ANS104 and FIPS132, which references IEEE1012. The criteria in the template provide a brief summary of those standards' requirements.

2.6.2. Software Quality Assurance (SQA)

The criteria for SQA are derived from IEEE7301. This standard relies heavily on the existence of project documentation. SQA is the planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements. The thrust of SQA is *product* assurance. A theme which began to develop in the late 1980's is that of process: process assurance, process improvement, quality of the process related to quality of the product. Some evolving standards are beginning to cite process rather than either phase or product for discussions on quality. The new set of SQA activities focuses on evaluating the processes by which products are developed or manufactured. This study reviews SQA activities primarily relative to product.

Because IEEE7301 requires an SCM plan (SCMP), this report notes whether IEEE7301 or a similar standard is cited in the documents. Another consideration was whether requirements for SCM were included as part of SQA. The inclusion is neither a negative nor positive attribute, but rather provides information about how complete the requirements of the document are for SCM.

2.6.3. Software Configuration Management (SCM)

SCM is the discipline of identifying and documenting characteristics of items in software development and maintenance, to baselining the items, and controlling changes to them. Typical activities consist of configuration identification, configuration control, configuration status control, and configuration audit. The criteria for SCM are taken from IEEE828. This standard is presently serving as a base document for an ISO/IEC JTC1 SC7 (Software Engineering) working group for developing a standard on SCM. Requirements for SCM should be included or referenced in a document for high integrity software.

Broad-based system documents may treat CM only at the system level. Software used in safety systems for nuclear power plants is often only one of many components comprising the total system, and, at the system level, software CM may not be adequately provided for. Hence, the treatment of SCM during software development, before system integration, was examined carefully.

2.6.4. Software Hazard Analysis

A pre-requisite for any critical system is an analysis of the hazards or threats that the system must protect against. For example, a power plant safety shutdown system must continue to function even during a power failure. While the study was mostly concerned with hazard analyses applied to software, it should be noted that software hazard analysis (e.g., software fault tree analysis (FTA)) is an integral part of system hazard analysis, and both should be conducted in order to assure that all hazards have been identified. Both types of hazard analysis are essential in designing a system for fail-safe operation (e.g., protection against division by zero). In response to a software hazard analysis, some software requirements, software design features, or software code may be changed.

2.7. Project Planning and Management

Well-defined project management procedures are as important for the development of high integrity software as they are for any quality product. The documents reviewed are broad in scope and should contain some requirements for how the development of software will be planned, managed, and monitored. Criteria in the template are derived partially from IEEE1058.

2.8. Procurement Concerns

Some evolving standards are addressing concerns of the customer. For example, the customer of a system may have some concerns about the people building and evaluating the software. Are they capable? Should evaluators be independent of the vendor? What should their training plans look like? Do the companies have a quality management policy? Some standards also address assessment of qualifications of the vendor and vendor plans for remaining qualified. Another procurement issue involves the use of automated support to build and verify the system and the use of pre-existing software in the software system; the issue is whether this software should be at the same level of assurance as the system. Some standards may include clauses that identify what conformance to the standard means; this is also a procurement concern. Any topics included in the documents that affect the customer at contract are discussed in this section of the template.

2.9. Presentation

One of the major problems with using a standard and verifying compliance with it is that all too often the "requirements" of the standard are not specified in a non-ambiguous, orderly manner. While rewriting a document, or reporting on every ambiguous statement was beyond the scope of this study, some examples of poorly-formulated statements were stated. In many cases, different categories of requirements were specified in documentation requirements. For example, required software functionality against hazards and required software engineering practices for a process should be stated separately rather than in the documentation requirements for a specific process (e.g., design).

2.10. Supplemental Information

Properties of a document that are not covered anywhere else on the template are described in this section. For example, the template does not provide criteria specifically for maintenance. Requirements of documents addressing maintenance are described in this part of the template.

2.11. General Comments

This section contains an overall analysis, based on the two sets of principal questions. The first set deals with *reasonable* assurance, not absolute assurance, and certainly judgment is subjective. The answer to the second set relies heavily on the findings in the Presentation section.

3. COMPARISON OF THE REVIEW DOCUMENTS

This section compares the documents for each major evaluation category, while Appendix B describes documents individually. The degree to which documents address each category varies widely. Some of the documents were draft documents at the time of evaluation. The purpose of including these documents is to identify the current thinking of the experts concerning standards for assurance of high integrity systems, especially software systems for the nuclear industry. The comparison of all the documents can assist in identifying proven requirements of standards, new requirements to be added, and organization and presentation format that will make standards easier to use, and easier for auditors to show compliance.

3.1. Levels of Criticality/Assurance

Among the documents reviewed for this report, most address levels of criticality, but in different ways. Canadian documents DLP880[3] and SOFTENG do not address levels at all. DLP880 implicitly assumes it is addressing critical software, and SOFTENG is for safety-critical systems (highest level of requirements). However, the purpose of the third Canadian document, CATEGORY, is to provide guidance on classifying software according to the consequences of failure but it does not associate software engineering practices with those categories.

IEC880 makes no distinctions concerning assurance needs in the main sections of the document. While Appendix B identifies recommendations for design and programming practices by three levels of priority or importance, no guidance is given on a definition of priority or importance. One would expect that because one purpose of IECSUPP is to clarify and supplement IEC880, IECSUPP would address priority. IECSUPP does this only in specifying diversity requirements, depending on reliability requirements. Even ANS7432 neglects to address levels of assurance, but it does require that the tools used for verification must have SQA measures on them commensurate with their importance to the verification process.

EWICS2-1 states that the design constraints should be associated with the level of criticality. In EWICS2-2, seven levels of criticality are related to types of systems and values for attributes like unavailability and failure probability.

RTCA178A discusses three levels of criticality, and the levels must be addressed in the certification plan for the software. RTCA178A is currently undergoing revision; the current draft defines five levels of criticality and provides guidance for determining the level of a system.

There is some disagreement within the software engineering community with respect to levels of assurance and the requirements that are appropriate for each level. Two questions must be resolved: How many levels of assurance should the standard provide? What activities are needed at each level? The levels of assurance specified in computer security standards are the result of concerns that are unique to security, so there is no reason to use the same divisions for a nuclear safety standard.

There are at least two other efforts addressing criticality assessment and levels of assurance. One is the High

[3]Acronyms for documents used in this section are defined in Tables 1-1 and 1-2.

Integrity Software project at NIST, and the other is the working group for the revision of IEEE1012. A recommendation from the Workshop on High Integrity Software held at NIST on January 22-23, 1991, is that criticality assessment should be based on four levels. The IEEE1012 working group discussions are leaning toward five levels. In both efforts, the intention is to assign engineering and assurance practices according to the level of criticality.

NIST recognizes the difficulty of assigning software engineering practices and assurance techniques to lifecycle processes based on levels of criticality. However, NIST recommends that this should be accomplished. For at least the highest level of criticality, requirements are already well-defined in several standards and guidelines; these requirements need to be stated in one document. The primary difficulty comes from the refinement of the activities, that is, eliminating or changing them for the middle levels. Different levels will cost developers and assurers overhead in establishing and managing their practices at different levels. There will also be cost involved in the training of staff and the performance of activities at different levels. There will be cost also for auditors to understand the different levels. For other evaluation methods, accredited laboratories will need to establish methods for each of the levels. It is important for standards makers to be able to demonstrate that requirements at four or five different levels are significant and sufficient to satisfy the criticality needs of the product at a given level.

3.2. Lifecycle Phases

The trend today is to discuss lifecycle not so much by phase, but by processes necessary to produce assured software. While phases are identified in this report, whether a document has requirements for necessary processes, including those for assurance, was also considered.

Some documents are special purpose documents and address those parts of the lifecycle phases that they affect. For example, the categorization document, CATEGORY, is concerned with procedures and guidelines for categorizing software; the categorization is assigned at the initiation of a project. EWICS2-1 provides guidelines for the design of safety critical systems and does not provide guidance on other lifecycle issues. The primary concern in EWICS2-1 is the process of design at the system level with some guidance on software considerations.

Some of the documents deal strictly with requirements for the software lifecycle (e.g., IEC880, IECSUPP, SOFTENG, and RTCA178A). However, they do place the software phases in context with system phases. DLP880 and ANS7432 also address system requirements and integration of hardware and software. The EWICS documents are concerned at the system level, with some specific references to software. In almost all of these documents, there is often confusion as to whether system or software is the focus of activity.

For safety systems in nuclear power plants in which software is embedded and must always be related to the system, the following issues on lifecycle are especially important:

- Does the document relate software activities to system requirements?

- By treating software as part of the system, does the document remove necessary emphasis on software (e.g., CM requirements at the system level only)?

o Are all lifecycle processes covered well by combining requirements of the documents?

No single document provides sufficient requirements for the first two questions. The combination of the documents provides coverage, at least minimally, in the lifecycle processes. While IEC880, in spite of its problems with presentation, comes close, it does not address project management. Other documents do a better job in other areas (e.g., NPR6300 on reuse and corrective action). While ANS7432 does address the software-system relationship, overall its requirements for software for all lifecycle processes are either minimal or nonexistent. If taken as a whole, the EWICS documents address (system) design and maintenance, but provide little guidance on other aspects of the software lifecycle processes. However, EWICS2-4 and SOFTENG are the only two documents which address quality attributes and measures. With respect to maintenance, IEC880 and EWICS2-5 provide more guidance than the other documents.

RTCA178A provides rather generic requirements for most lifecycle processes. It is currently under revision; the revision will probably be oriented more toward processes, not phases, and may contain rigorous requirements.

The combination of the best features of the documents reviewed may provide reasonable coverage for lifecycle processes.

3.3. Documentation

None of the documents specifically address documentation. Their requirements for documentation range from a simple statement for each type of document to a complete description of the quality attributes of a document. ANS7432 falls into the terse category (e.g., completeness, consistency, and documentation standards are implied). The most complex set of documentation requirements are in DLP880 where documentation is to be written in formal specification languages.

Only one document, SOFTENG, provides rigorous guidance on the quality attributes that should be inherent in documentation. For each document, criteria are identified for each required quality attribute.

One of the features of several documents, especially IEC880 and SOFTENG, is that documentation requirements included requirements for the software itself. For example, designing modules with a single well-defined function is a software engineering practice that designers apply through their thinking processes to structure the system for the best possible design for the system's operational capability and assurance. The primary purpose of implementing this practice is not documentation. In other cases, functional requirements were hidden in documentation requirements. It is recommended that standards make the distinction between documentation requirements and requirements for software engineering practices and software functionality against hazards. The documentation requirement may be that the software engineering practices should be documented separately, for example, MOD0055's requirements for a "Code of Design Practices."

In most of the reviewed documents, separate documentation is specified for each lifecycle phase or process. An exception is RTCA178A which treats the software development and verification plan as one document.

Several documents contained checklists of varying degrees of clarity and completeness. This is a positive feature.

3.4. Required Software Functionality Against Hazards

IEC880 and EWICS2-3 included lists of functions that can be used to counter specific hazards. Of these two, the questionnaire in EWICS2-3 is the more comprehensive. IEC880 contains annotations indicating what each function is "good for" and "good against," but these are generally obvious, so the annotations provide little useful guidance. For example, the annotation for retry procedures indicates that retries are useful against sporadic hardware faults, and range checking of variables is said to help guard against "yet undetected errors."

The inclusion of checklists of functions to guard against hazards is helpful in a standard, but it is probably not appropriate to mandate specific functions when a standard covers a broad category of systems. Some functionality that is considered essential for all nuclear safety systems (e.g., range checking) might be required, but, in general, some functions may not be appropriate for all systems. This is consistent with standards for high integrity systems supporting security. Standards for general purpose secure systems have less prescribed functionality than those for a more narrow range of applications, such as data encryption. It may be beneficial to consider supplemental standards for special purpose systems.

3.5. Software Engineering Practices

Software engineering practices are either those techniques recommended or required to prevent errors from being entered into the system during construction, or are properties to be built into the system for high integrity. An example of the first type is the use of formal specification languages, and an example of the second type is the use of modularity.

The documents reviewed vary enormously in their recommendations regarding software engineering practices. Most contain at least some guidance on good practices. The only two documents which do not cite any software engineering practices are RTCA178A and NPR6300. There is little agreement on the practices that are mentioned. While EWICS2-4 and EWICS2-5 do not address software engineering practices, the other EWICS documents provide comprehensive coverage and address software engineering practices. Summaries of the more significant software engineering practices are given below approximately in order of consensus among the documents.

Modularity and critical component isolation - The software engineering practices cited by most of the documents are the use of modularity in design and the isolation of critical components. In this regard, the documents are consistent with current thinking in the software engineering community and with other standards for critical software.

Programming Language - Several documents state principles for programming languages. The consensus in this area is for use of high-level languages (e.g., C, Fortran, Ada) rather than assembler, and for languages that support automatic checking of data types and function arguments. For example, Ada or C++ will warn if a function is called with an integer argument when it is expecting a character string. Here again, the documents are consistent with current thinking in the software engineering community and with other standards for critical software. Another programming consideration is the use of structured programming (the use of restricted control structures rather than arbitrary branching). Structured programming is now nearly universally accepted as good practice. The documents reviewed reflect this acceptance.

Formal methods - Over the past decade, there has been increasing interest in the use of formal methods for complex software. *Formal methods* refers to the use of mathematical logic and related areas of mathematics to specify and model the behavior of software. Formal methods are required by only one of the documents reviewed (DLP880). Another, the supplement to IEC880 (IECSUPP), says that "formal methods should be considered for the highest requirement of safety importance." EWICS2-3 gives preference to the use of formal specifications over informal ones, but does not require the use of formal methods. IEC880 notes only that "a formal specification language may be a help to show coherence and completeness of the software functional requirements." A trend toward greater reliance on formal methods is evident in the documents reviewed. DLP880 and IECSUPP were 1991 drafts; EWICS2-3 was written in 1989, and IEC880 in 1986. In the area of formal methods, the nuclear documents reviewed lag behind other standards for critical software. In the fields of computer security and data communications, formal methods have become accepted practice, and are required at the higher levels of all significant security standards and by MOD0055.

Documentation of software engineering practices - A mechanism frequently used is to define documentation as a description of a lifecycle process. For example, in IEC880 and SOFTENG, documentation for a software requirements specification often specifies the principles or functions the system must embody, or in the case of SQA, the activities to be performed. Software engineering practices are hidden in documentation requirements. Those practices are discussed in this report in Section 3.3. Taken together, the documents reviewed gave adequate treatment to most aspects of software engineering practices, except in the area of formal methods. (This statement reflects the documents in general. As noted, there was much variation.)

Quality attributes - Only SOFTENG and EWICS2-4 provided either specific requirements or measures for quality attributes like completeness, consistency, and maintainability.

3.6. Assurance Activities

In the nuclear power industry, as in many other industries, software is one component of a company's business, and of a power plant. At the top management level, the view is of the whole, not a part. Therefore, system CM and system validation are the engineering concepts that make sense to executives of manufacturing companies. For software companies, executives think in terms of *software* CM and *software* validation. The difference is non-trivial and has caused much misunderstanding in the development of standards. Software is deeply embedded in systems where software cannot fully stand alone, yet some other components of these systems are not only plug-in but are built to precise, accredited standards. CM and testing of these components during their development is an expected activity. Software should be treated similarly. For testing this has not been always possible. For one thing, software systems have been unique for each system in which they will be embedded. There is no precise set of validated specifications. Second, their full functionality usually can only be simulated and cannot be tested in real-time during software development.

The purpose of this study was to discover how well the documents provide assurance of the software and how well the software will fulfill its system responsibilities. Few of the documents reviewed focused entirely on software. There was a lot of second-guessing whether system level requirements were applied at the software development level or only at the point when software was integrated with the system. If the requirements are unclear, how can auditors check for exact compliance? If accredited, precise standards for software existed, as they do for other

components (e.g., pipes, power cables), then this review would have been simpler. This study reemphasized the growing recognition that the software industry must identify some precise standards for software that permit measurement of its quality.

For the assurance activities, P1228 focuses on safety issues and requires specific assurance activities in all the categories of Section 3.5 of this report. Under P1228, all documentation for assurance activities may serve as special sections of plans for those activities (e.g., safety requirements for the SV&V plan (SVVP), safety requirements for the SQA plan (SQAP), for the SCMP). The assumption of the P1228 draft is that the other IEEE Software Engineering Standards, or similar standards, will be used. For computer security planning, which may be important to the nuclear industry as dependency grows on large databases and distributed computing, perhaps the documentation requirements of P1228 can be adapted for computer security.

3.6.1. Software Verification and Validation (SV&V)

The difference between system and software viewpoints stands out in the documents reviewed for this study. For example, ANS7432 is concerned with computer system validation and not particularly concerned with software issues. Software verification is the software testing; in the software world, this can be confusing. Part of the rationale for not treating SV&V as separate functions in IEEE1012 is to avoid this confusion. The final step of SV&V is the system validation, as in system standards; SV&V consists of these activities applied as the software evolves to assure the internal properties of the software and the external relationships to the system.

DLP880 refers to software verification but is in reality SV&V. One caution with DLP880 is the assumption that the vendor may produce the verification plan which is then implemented by an independent team. One should not think this is the only meaning of IV&V, because the fullest possible benefit of independence is the independent planning process in which the IV&V brings a different perspective to the types of analysis and test strategies.

Two documents, EWICS2-1 and P1228, focus on the safety requirements; this is acceptable since these documents are intended to augment other more general standards and the intent is to ensure attention to the safety functions. EWICS2-3 should be used by verifiers to guide them in checking features of the software, and auditors to check how well the developers and verifiers have followed guidelines.

IEC880 specifically addresses software verification, and is reasonably thorough. There are weaknesses, however. The major weakness is that of presentation; a reader has to search several places before finding all the requirements for a given process, in this instance, verification. Technical weaknesses include a lack of specific requirements for requirements traceability.

Some documents recommend several test strategies and test conditions (e.g., stress test, logic test, boundary test) but the selections are not the same in all the documents. For example, IEC880 has long lists of strategies and conditions but omits stress testing, while SOFTENG includes stress testing. Error analysis should be a requirement in all SV&V or SQA standards or sections of standards addressing SV&V or SQA. Error analysis is important for uncovering a type of error (e.g., misunderstanding of trigonometry) that could appear elsewhere in the system. When the type of error is made because of a misunderstanding or a wrong specification, it is important to check other places in the program that are based on the same assumptions, especially if the same person is responsible. Otherwise, a potentially critical error could slip through.

While RTCA178A provides the most detailed and best organized set of requirements for software verification (including validation), the software verification assurance matrix is too high-level to be truly useful for auditors.

From the review of these documents, including the base documents IEEE1012 and ANS104, recommendations for improving standards for SV&V include

- clearer relationship and requirements to the system
- practices based on levels of criticality
- distinctive requirements for different test types
- detailed checklists
- application of SV&V when modern development technologies are used (e.g, no document addressed SV&V for prototyping or expert systems)
- error analysis
- definition of the quality attributes for which verification is required

3.6.2. Software Quality Assurance (SQA)

As a general comment, NRC should note that these documents are concerned strictly with SQA of the product, not the vendor processes. Current and evolving SQA standards are addressing process as well as product. When NRC audits a particular product, will NRC be concerned about whether a vendor has changed processes mid-stream, or for the next product? Probably not. For a current audit, NRC will likely be interested in whether a given product has the required quality level. But, when new SQA standards are written, what happens if they require activities for both process and product? NRC needs to study this question to determine if process quality is outside the scope of their requirements.

Several documents addressed SQA in a general manner. For example, ANS7432 requires that SQA be addressed in the software development plan (SDP). IEC880 simply requires an SQAP. This is not sufficient because it will not be clear what is required of the vendors. For audit and review purposes, NRC must know the following:

- the minimum set of SQA activities that are to be performed
- to what degree SCM and SV&V are included in SQA

Some documents permit national standards to be used. NRC would need to ensure specification of an SQA standard or expect their auditor to know every SQA standard quite well.

Design and code inspections can be either SQA or SV&V activities. In addition, requirement for inspections was not consistent in the documents. Both ANS7432 and EWICS2-1 provide detailed procedures for SQA of design

and EWICS2-4 addresses SQA entirely.

There is a growing recognition that SQA procedures are needed for existing software programs and for reuse of software modules. NPR6300 provides detailed guidance and it appears that IECSUPP will address the topic also.

The SQA sections of the documents, like those for SV&V, are in general weak concerning anomaly reporting, corrective action and follow-up, and error analysis.

3.6.3. Software Configuration Management (SCM)

Software CM is another process that sometimes is addressed only at the system level. While the size of software systems used in safety systems for nuclear power plants may be small, the critical role of software for safety mandates that SCM be required for all the lifecycle processes and products of such software. IEC880's requirement for system CM is insufficient. ANS7432 and EWICS2-3 simply ignore the topic. Several documents, DLP880, SOFTENG, RTCA178A, and NPR6300, require SCM activities with varying degrees of rigor. The international community (ISO/IEC JTC1 SC7) is presently using IEEE828 as a base document for producing an international standard on SCM. When the international standard is published, NRC and the standards community in general may consider simply citing this SCM standard directly.

3.6.4. Software Hazard Analysis

Typically the initial hazard analysis is performed from a total system, environmental perspective and the results may affect the system requirements and design. From the software perspective, the results of that analysis should be an input to the software assurance activities. By examining these results, the software experts identify what potential hazards have an impact on the software, or may be mitigated or prevented by software. For the same reason, results of other system hazard analyses should be required inputs to software assurance activities. Additionally, hazard analyses specific to software should be conducted. There is some debate over whether the software hazard analyses and related analyses should be considered development or assurance activities. The recommendation here is that the perspective of software assurance may lend itself somewhat better to conducting software hazard analyses and using system hazard analyses to check the safety impact of the software.

While many of the documents of the study addressed system hazard analysis, it was quite difficult to determine if the hazard analysis was conducted on software features. When the hazard analysis report was strictly system, its report was not usually required as an input to any software development or assurance activities.

Two documents specifically addressed software safety analysis. P1228 requires the results of the preliminary system hazard analysis and requires additional software safety analyses throughout the lifecycle. SOFTENG requires a code hazard analysis and a report with ten requirements. Examples of items that must be identified in the report include input conditions which could lead to the software causing an unsafe state, failure modes related to instructions in code which could lead to an unsafe subsystem failure, and code modifications which would eliminate the identified failure modes.

3.7. Project Planning and Management

Most of the documents in this study either do not address project management activities or do so indirectly through other governing principles. For example, requirements on planning for SQA and SCM may be considered requirements for project planning. SOFTENG includes requirements for project management in requirements for the SDP. EWICS2-4 addresses project management through acceptance criteria. The P1228 draft expects a project management plan, and requires that the plan be augmented to address software safety issues.

3.8. Procurement Concerns

One concern that NRC has is whether or not assurance activities should be performed by independent teams. ANS7432 uses a non-binding statement in the foreword to recommend independence and requires independence of the verification group at the system level. Others, like DLP880 and IEC880, recommend that verification plans be written so that an independent team may implement the plan. IECSUPP suggests complete separation of development and verification teams. EWICS2-1 and EWICS2-2 ask for an IV&V assessment. P1228 also recommends IV&V. SOFTENG asks for independence between development and verification; management of the developers is different from managers of the verifiers (but does not require a separate organization). These recommendations present another problem; nowhere is there a standard definition of independence and of the tasks of IV&V. A non-exclusive list of possible meanings of IV&V duties includes the following:

- The independent team writes all test plans and executes them.
- The independent team performs static analyses on the software design.
- The independent team only performs test execution.

Additionally, can the "independent" team be simply another department within a vendor's organization? What conditions make the team "independent?" Unless the document clearly specifies a definition of IV&V, requirements for IV&V are ambiguous.

Most of the documents do not specify contractor capability assessment although the EWICS documents do ask for compliance with ISO9000 which requires assessment of contractor's quality system. P1228 requires the software safety plan (SSP) to specify qualifications for the personnel performing software safety activities.

The permission to tailor a standard to a project may present two problems. One, the tailored version may not produce a satisfactory assurance of the product. Two, an auditor may have difficulty assessing compliance. SOFTENG has a statement that all requirements of the standard must be met for compliance.

For an auditor to verify vendor compliance to a standard, it is helpful to have a statement of conformance within the standard. Only two of the reviewed documents have firm conformance clauses. SOFTENG states that conformance means all its requirements must be met. Of the five separate chapters in EWICS2, all except one have strong conformance clauses that list specific requirements. For example, EWICS2-1 requires written procedures that identify the existence of activities corresponding to each and every step of the guideline. EWICS2-3 consists of a questionnaire; a conformance clause is inappropriate.

Several of the documents suggest that the use of pre-existing software in a product falls under the requirements of a document. Some also require the same level of assurance for automated development or assurance tools. For example, P1228 states that pre-existing software must be in compliance with P1228 and the verification of support tools depends on the level of assurance of the system. While IEC880 has requirements on the use of operating systems, it does not require that automatic development and verification aids be tested.

3.9. Presentation

The documents reviewed have a variety of problems with their presentation. The major problem lies with usage of words to indicate requirements: "shall," "should," "must," "may." When the words "shall" and "should" appear in the same paragraph, it can be confusing to vendors, assurers, customers, and auditors. Requirements and recommendations need to be clearly distinctive from one another. Ambiguous statements in several documents (e.g., "the software must be easy to test" [IEC880]) are meaningless. An example of language that does impose meaningful constraints on qualities may be found in SOFTENG.

Another concern is that features required to be present in the software, development practices, and descriptions of the software are often specified in documentation. It is recommended that standards keep separate different categories of requirements.

3.10. Supplemental Information

The Supplemental Information section for each document (see Sections B.x.10 of this report) contains information about the documents that was not covered by the template categories. The important findings are that only two documents (IEC880, EWICS2-5) address software maintenance in detail and only NPR6300 addresses problem reporting and corrective action in detail.

4. SUMMARY

In general, no standard can guarantee the safety of a particular software system. In other words, no one can ever say "If a vendor follows this standard, the system will be safe." The judgement of safety for a particular installation must be made by the appropriate authorities. Ideally, standards should state what is required for evaluating the safety of a system, and to determine if the software complies with the standard. That is, "These are the things a vendor must do to enable NRC to judge whether the system is safe enough to not pose an undue risk to public health and safety."

No single standard or guideline reviewed completely satisfies the evaluation criteria, and there is relatively little consensus among the documents. None of the documents positively answered the first set of principal questions "Do the requirements of the document provide assurance of the nuclear safety system software developed, maintained, and operated according to these requirements? That is, does the document contain requirements for building verifiably dependable nuclear safety system software?" However, most of the documents satisfy at least one category of criteria.

The documents reviewed have several common problems. It is unclear whether the system or software is the focus of the activities. There is a lack of system information provided to the software. Process requirements are specified under documentation requirements. Specification of required software functionality against hazards is usually insufficient. And, in general, conformance to most of the documents would be difficult to demonstrate.

The scope of a standard must be clearly identified relative to system and software lifecycles. The requirements for software must ensure that the software is always assured relative to its relationship to the system. Documentation requirements should be limited to what should be *included* in the document and how this content should be presented. Requirements for software engineering practices and software functionality against hazards (what should be *done*) should be listed separately. The functions listed in Appendix A are not all essential to all systems, however, standards should provide guidance on what functions are necessary for specific types of systems.

It is important to develop a standard with rigorous requirements for documentation, required software functionality against hazards, software engineering practices, SV&V, SQA, SCM, software hazard analysis, and project management. It is acceptable to cite specific standards to provide requirements for any of the categories (e.g., IEEE1012[4] or ANS104 for SV&V). In fact, a standard that is self-contained or has reference to specific standards helps to prevent omissions and conflicts among standards. Software engineering practices and assurance techniques should be assigned based on levels of criticality/assurance. The activities for each level may all be defined within one standard, or different standards may be developed for each level of criticality/assurance.

The language of a standard must be non-ambiguous. Requirements should be clearly stated and contained within the body of the standard (e.g., not in appendices). Until software engineering practice is rigorously codified in handbooks as in other engineering fields, standards for the assurance of high integrity software should describe all practices or cite acceptable standards for them, which encompass at least the criteria in Appendix A.

[4]Acronyms used in this section are defined in Tables 1-1 and 1-2.

There was some consensus among the documents on certain software engineering practices including the use of modularity and critical component isolation, high level programming languages, formal methods, documentation of software engineering practices, and quality attributes. But, in general, the guidance is enormously varied.

Frequently, the documents did not distinguish between documentation requirements and those for software engineering practices and software functionality against hazards. This distinction should be made in the documents. Besides the benefit of vendors' improved understanding of their requirements, the distinction should facilitate the task of verifying compliance with a standard.

Procurement issues such as independent evaluations, contractor capability assessments, pre-existing software and support software should be addressed in standards. At a minimum, a standard definition of independent organization should be developed. While such a definition may have variants, a standard for high integrity software can cite the required variant of independence. The customer is also interested in whether the system has built in conformance with the standard. To verify vendor compliance to a standard, it is helpful to have a statement of conformance within the standard.

It is recommended that NRC considerations of consensus standards, draft standards, and guidelines include the concerns identified in this report. While information from all of the documents can be used in developing a rigorous standard, other concerns must be addressed. For example, the scope of the standard within system and software lifecycles must be clearly identified. The standard should require that software is always assured in the context of the particular system under evaluation. Guidance for assurance of high integrity software should either describe all practices or cite acceptable standards for them, encompassing at least the criteria in Appendix A.

5. REFERENCES

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.

ANS41

ANSI/ANS-4.1-1978, "Design Basis Criteria for Safety Systems in Nuclear Power Generating Stations," American Nuclear Society, 1987.

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.

ASMENQA1

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

ASMENQA2

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

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.

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.

EWICS1 Redmill, F. J. (ed.), Dependability of Critical Computer Systems 1, The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1988.

EWICS2-1

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 1, "Guidelines to Design Computer Systems for Safety," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS2-2

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 2, "Guidelines for the Assessment of the Safety and Reliability of Critical Computer Systems," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS2-3

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 3, "A Questionnaire for System Safety and Reliability Assessment," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS2-4

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 4, "A Guideline on Software Quality Assurance and Measures," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS2-5

Redmill, F. J. (ed.), Dependability of Critical Computer Systems 2, Chapter 5, "Guidelines on the Maintenance and Modification of Safety-Related Computer Systems," The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1989.

EWICS3

Bishop, P. G. (ed.), Dependability of Critical Computer Systems 3 - Techniques Directory, The European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Science Publishers LTD, 1990.

FIPS101

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

FIPS132

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

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.

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.

IEEE603

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

IEEE828

ANSI/IEEE Std 828-1983, "IEEE Standard for Software Configuration Management Plans," The Institute of Electrical and Electronics Engineers, Inc., 1983.

IEEE830

ANSI/IEEE Std 830-1984, "IEEE Standard for Software Requirements Specifications," The Institute of Electrical and Electronics Engineers, Inc., 1984.

IEEE983

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

IEEE1012

ANSI/IEEE Std 1012-1986, "IEEE Standard for Software Verification and Validation Plans," The Institute of Electrical and Electronics Engineers, Inc., November 14, 1986.

IEEE1016

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

IEEE1042

ANSI/IEEE Std 1042-1987, "IEEE Standard for Guide to Software Configuration Management," The Institute of Electrical and Electronics Engineers, Inc., 1987.

IEEE1058

ANSI/IEEE Std 1058.1-1987, "IEEE Standard for Software Project Management Plans," The Institute of Electrical and Electronics Engineers, Inc., 1988.

IEEE1074

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

ISO9000

ISO 9000, "International Standards for Quality Management," International Standards Organization, ISO Central Secretariat, Case Postale 56, CH-1211, Geneve 20, Switzerland, May 1990.

ITSEC

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

MOD0055

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

NIST180

NIST Special Publication 500-180, "Guide to Software Acceptance," U.S. Department of Commerce/National Institute of Standards and Technology, April 1990.

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.

NPR0006

NPR-STD-0006, "NPR Configuration Management Plan," Office of New Production Reactors, U.S. Department of Energy.

NPR6300

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

NPR6301

NPR-STD-6301, "Configuration Item Identifiers," Office of New Production Reactors, U.S. Department of Energy.

NRC91

"Computers at Risk," National Research Council, National Academy Press, 1991.

P1228

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

RTCA178A

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

SAFEIT

"SafeIT," Volumes 1 and 2, Interdepartmental Committee on Software Engineering, ICSE Secretariat, Department of Trade and Industry, London SW1E6SW, UK, June 1990.

SOFTENG

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

TCSEC

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

APPENDIX A. DESCRIPTION OF CRITERIA TEMPLATE

The template for analyzing a standard or guideline is designed such that the following questions can be answered:

FIRST SET OF PRINCIPAL QUESTIONS:

Do the requirements of the document provide assurance of the nuclear safety system software developed, maintained, and operated according to these requirements? That is, does the document contain requirements for building verifiably dependable nuclear safety system software?

SECOND SET OF PRINCIPAL QUESTIONS:

Will the requirements of the document provide NRC auditors with enough information to verify that the vendor product is in compliance with the requirements, and are there clearly measurable conformance clauses? If so, is it expected that two auditors will arrive at the same conclusions relative to the product?

The features identified in this template represent the major topics against which the documents were analyzed.

A.1. Levels of Criticality/Assurance

- Are there levels? If yes, how many?
- Are the levels associated with practices or assurance techniques?
- Is independence of evaluators associated with levels?

A.2. Lifecycle Phases

What phases of the lifecycle does the document address using the following as a baseline lifecycle:

Initiation
Requirements
Design
Code (Implementation)
Integration & Test
Installation
O&M

A.3. Documentation

- Which of the following are NOT specified by the document (at a minimum these documents should be specified):
  - requirements document
  - design document
  - code documentation
  - user manual
  - test documentation
  - installation document
  - operations manual
  - maintenance manual
  - project planning documentation
  - source code
- Is a standard referenced?
- Is format specified?
- Is content specified?
- Are there traceability requirements to other documentation?
- Is there a checklist for specific documents?
- Are the required attributes (e.g., completeness) of the document expressed in quantified language?

A.4. Required Software Functionality Against Hazards

Does the document address

- o self-testing of critical functions
- o memory, storage integrity checking
- o redundant hardware components
- o redundant software components
- o fault tolerance, automated error recovery
- o two or more independent operations specified for activation of critical functions
- o safety from single bit failures in flags, addresses
- o safe shutdown or safe state in case of error
- o safety/security checking functions that are non-bypassable
- o parameter checking
- o checks on sequence of operations
- o malicious manipulation of system and data
- o security control: access
- o override of operator `mis-key' or other unintended functions

A.5. Software Engineering Practices

- o What methodologies are used to build the system? What features of design should the system have?
- o What kind of specifications are addressed: formal? semi-formal? informal?
- o Are critical components isolated from other parts of the system?
- o Is modularity/information hiding used?
- o What kind of language is specified (high-level, assembler)?
- o Does the document prohibit or recommend limiting the use of particular programming and design practices, and, if so, why?
- o Definitions of and criteria for quality attributes.

A.6. Assurance Activities

A.6.1. Software Verification and Validation (SV&V)

- Is a standard referenced?
- Does the SV&V cover all lifecycle phases (initiation, requirements design, code, integration and test, installation, O&M)?
- Does the document require the following activities:
    - traceability analysis
    - evaluation (e.g., analysis, review, audit) of development products (see document list)
    - separation of test types (unit, integration, system, acceptance)
    - documentation requirements for testing
    - management of SV&V
    - review of SV&V products (e.g., test results)
- Does the document provide the following:
    - checklists
    - suggested or required techniques for performing SV&V
    - guidance on independence of developers and V&Vers will be noted under Section A.8, and information on levels based on the criticality under Section A.1

Note: Standards used to provide the requirements for this section are FIPS132/IEEE1012 and ANS104.

- Testing activities:
    - Is a standard referenced?
    - Are there detailed testing requirements? Minimal requirements? None?
    - Are the following specified:
        - unit test (component, module)
        - integration test
        - subsystem test
        - system test (hardware/software)
        - acceptance test
        - reliability testing

- test plans
- test cases and procedures
- procedures for anomaly resolution
- test strategies
- test coverage

- Are test plans for system, acceptance test required during the requirements process?

A.6.2. Software Quality Assurance (SQA)

- Is a standard referenced?

- Are additional (other than what is specified in Section A.6.1) testing activities specified?

- Under SQA, is a minimum set of documents specified?

- Does SQA establish the standards, practices and procedures for the project?

- Are the following reviews specified:

  - software requirements review
  - software design review (both high-level and detailed)
  - review of SVVP

- Are audits specified?

- Are any metrics specified, or is SQA required to specify any metrics?

- Does SQA specify problem reporting and corrective action procedures?

- Is SQA responsible for identifying procedures for maintaining and storing controlled versions of the software?

- Is there a checklist for any of the SQA activities?

Note: Look for requirements for organization of SQA and for activities based on levels of assurance.

A.6.3. Software Configuration Management (SCM)

- Is a standard referenced?
- Are both system and software configuration activities identified?
- Are there requirements for the following:

  - interface control

- a configuration control board
- configuration baseline identification
- configuration control
- configuration status accounting
- configuration reviews and audits

Note: Look for requirements for organization, and for activities based on levels of assurance.

A.6.4. Software Hazard Analysis

- Is there an initial system hazard analysis from which software "impact" is analyzed?
- As additional system hazard analyses are conducted, are results provided to software processes?
- Is there a software hazard analysis?
- Does the software hazard analysis include a criticality analysis?
- When in the lifecycle is software hazard analysis to be performed?
- What techniques are specified?

A.7. Project Planning and Management

- Is a standard referenced?

- Are there detailed requirements? minimal requirements? none?

- Are measurements on progress required?

- When assurance activities reveal problems:

  - are those problems traced to specific processes?
  - has management granted authority to anyone to change the affected processes?

- Are progress or problem measurements used to change items other than the affected process (e.g., training of current staff, change of staff)?

- Is there guidance regarding authority/communication for changes to products?

A.8. Procurement Concerns

- Are there requirements for IV&V?

- Is a contractor capability assessment specified?

- Does the document contain conformance clauses?

- Are there requirements on pre-existing software or software used for development and assurance activities?

A.9. Presentation

- How well are the topics organized within the document?
- Is ambiguous language used (should vs. shall)?
- Cite examples of problems.
- Is material repeated throughout the document?
- Are words such as "easy," "suitable," and "appropriate" used without further definition?

A.10. Supplemental Information

Include important information not covered in the sections above.

A.11. General Comments

Commentary on the document as a whole.

APPENDIX B. REVIEW OF STANDARDS AND GUIDELINES

The review of each of the documents followed the format of the template in Table 1-3. The principal questions are answered under the General Comments section of each template. Because of the large volume of information, comments under each template heading are kept as terse as possible and are not always in sentence form. Commentary for each sub-topic within a template category is provided as a separate item or paragraph.

The words "Not addressed" are given whenever a document contains no material on a principal topic. In some cases, the topic may be outside the scope of a document and, hence, not applicable. It is, therefore, not necessarily undesirable that a document fails to address a certain subject.

Acronyms used in this section appear in Tables 1-1 and 1-2.

B.1. ANSI/IEEE-ANS-7-4.3.2-1982: Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations (1982) [ANS7432]

B.1.1. Levels of Criticality/Assurance

Not mentioned except that while the tools used for verification do not require verification, the SQA measures on these tools must be established relative to their importance to the verification process.

B.1.2. Lifecycle Phases

Hardware requirements, software requirements, hardware-software integration requirements for the computer system.

At a minimum a development plan, design, and implementation phase are included in the software development process.

B.1.3. Documentation

The hardware-software integration requirements document includes the general QA plan for hardware/software integration and integration test procedures with acceptance criteria. Hardware requirements that impact software must be documented; several categories of these requirements are listed.

The software requirements document specifies items for description including security requirements.

The SDP includes a list of items to be defined, including organization, methodology for achieving software attributes, assurance for auditability and testing, and SQA provisions and program.

A design document is required. Minimum detail is given on this document other than it requires traceability.

Implementation procedures are to be documented.

B.1.4. Required Software Functionality Against Hazards

The Foreword[5] states that functional and design criteria are fully covered in ANS41 and IEEE603.

The Foreword states that the joint working group writing this standard has not specifically required self-checking. The SDP must include requirements for achieving error tolerance.

[5]The Foreword is NOT a part of this standard.

B.1.5. Software Engineering Practices

The SDP must identify requirements for achieving modularity.

The Foreword states that functional and design criteria are fully covered by ANS41 and IEEE603.

B.1.6. Assurance Activities

B.1.6.1. Software Verification and Validation (SV&V)

Minimum requirements for the verification plan are described for organization, review and audit, and software test and analysis.

The hardware/software integration testing results shall be described in a report.

The computer system validation testing, including static and dynamic simulation, formal test plan, and evaluation of results by those not involved in design or implementation, shall be specified in a test report.

B.1.6.2. Software Quality Assurance (SQA)

SQA is addressed in the SDP.

B.1.6.3. Software Configuration Management (SCM)

Not addressed.

B.1.6.4. Software Hazard Analysis

Not addressed.

B.1.7. Project Planning and Management

Not addressed.

B.1.8. Procurement Concerns

The Foreword addresses the need for independent verification as does Section 7.1.

B.1.9. Presentation

Since this standard is superseded by ASMENQA2, which incorporates the quality requirements, it is inappropriate to comment on its presentation at this time.

B.1.10. Supplemental Information

The Foreword claims that conforming to this standard does not guarantee adequacy of a system, however, not meeting the criteria of this standard ensures inadequacy. It also states that the following topics are planned for future work: quantitative software standards, computer security, self-testing, distributed computer systems, techniques for independent validation, firmware, and "simplification" objectives.

B.1.11. General Comments

This standard is currently under revision. The draft of the new ANS 7-4.3.2 is significantly different. The quality requirements of the current ANS 7-4.3.2 are captured by ASMENQA2.

Response to the First Set of Principal Questions:

- While this standard presents general system-level requirements, this review is concerned with the assurance activities for software, and how those activities relate to the system. This standard does address some issues of the hardware/ software integration. Documentation for the hardware must identify all requirements that impact software. The document does not require that a hazard analysis or criticality analysis be performed relative to those software requirements. Specific requirements for SCM, software integration, and types of software testing are lacking or are too general to be useful.

Response to the Second Set of Principal Questions:

- The requirements of this standard are stated so that an auditor could verify for at least minimum compliance, that is, details on the activities are not usually required.

B.2. Guideline for the Categorization of Software in Ontario Hydro's Nuclear Facilities With Respect to Nuclear Safety [CATEGORY]

B.2.1. Levels of Criticality/Assurance

The purpose of this guideline is to provide the rationale for determining levels of criticality of software systems used in nuclear power plants. Four levels of classification are defined, depending on impact of software failure. It is intended that other standards will be developed to associate software engineering practices to levels of assurance. At the highest level, a draft Standard for Software Engineering of Safety Critical Software exists.

B.2.2. Lifecycle Phases

Addresses categorization at initiation of project.

B.2.3. Documentation

Not addressed.

B.2.4. Required Software Functionality Against Hazards

Not addressed.

B.2.5. Software Engineering Practices

Design features are listed (both hardware and software) for lessening impact of failure.

B.2.6. Assurance Activities

B.2.6.1. Software Verification and Validation (SV&V)

Not addressed.

B.2.6.2. Software Quality Assurance (SQA)

Not addressed.

B.2.6.3. Software Configuration Management (SCM)

Not addressed.

B.2.6.4. Software Hazard Analysis

Criticality Assessment:

The four categories are:

1. Software is critical to nuclear safety (safety critical software).

2. Software has significant effect on nuclear safety, but effect of failure can be mitigated by special safety system action.

3. Software has some effect on nuclear safety, but failure does not prevent system from meeting its design intent, and failure does not require mitigating system action.

4. Software whose failure has no effect on nuclear safety.

Software failure impact type:

Three categories of failure impact are defined, according to specific loss of functionality.

Methodology:

Special safety systems are identified (e.g., shutdown systems, emergency coolant systems) and usually have a reliability requirement or unavailability target. First the safety significance of the system is identified and then the type of software failure impact is identified. Tables help to determine the final category.

B.2.7. Project Planning and Management

Not addressed.

B.2.8. Procurement Concerns

Not addressed.

B.2.9. Presentation

The methodology is clearly stated and examples are provided.

B.2.10. Supplemental Information

This guideline is a draft that will eventually become part of a family of software engineering documents for Ontario Hydro.

B.2.11. General Comments

NRC should encourage the use of a categorization standard similar to this document. Use of a categorization would support

- prevention of tailoring out requirements of other standards by stating no tailoring allowed for the highest category

- identification of whether there is separation of required software functionality against hazards (protection) and assurance requirements (e.g., completeness, consistency) in a standard

Response to the First Set of Principal Questions:

- This guideline's scope is restricted to identifying a methodology for criticality assessment and assessment of impact of failure. It does this well, in a manner that would assure appropriate assignment of levels of criticality within a system.

Response to the Second Set of Principal Questions:

- An auditor should be able to verify that the methodology has been implemented according to the guideline.

B.3. DLP880: Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations [DLP880]

B.3.1. Levels of Criticality/Assurance

Assumes that the software for which this standard is used is at the highest level of criticality.

B.3.2. Lifecycle Phases

Development includes system requirements, computer system design, software requirements, software behavior, software decomposition, module interface design, module internal design, and program design.

Maintenance and modification: Formal modification control procedure is specified in detail for modification request and executing a modification. Includes SV&V. Maintenance occurs as modification for the same reasons, with the addition of responding to an anomaly report.

Software verification is performed after each lifecycle phase.

B.3.3. Documentation

The management and integral processes documentation are not specified.

Requirements for the technical documentation for the development processes are in mathematical notation. Format and content are specified for the computer systems requirements document, system design document, software requirements document, software function specification, software module guide, module interface specifications, internal design documents, and program function specifications and displays.

B.3.4. Required Software Functionality Against Hazards

The programming language and its translator should not prevent error-limiting constructs; translation-time checking; nor, run-time type and array bound check, and parameter checking.

B.3.5. Software Engineering Practices

The principles applied in developing the requirements of the standard include

- use of formal methods
- top down design
- information hiding, modularity

The computer programming language used should have a thoroughly tested translator. If not, additional verification should justify the correctness of the translation.

"The languages should be completely and unambiguously defined, otherwise the use of the language shall be restricted to completely and unambiguously defined features." (p. 12)

The standard "strongly" prefers high-level languages over machine dependent ones.

"Automatic testing aids should be available." (p. 12)

The standard recommends the use of automatic tools.

B.3.6. Assurance Activities

B.3.6.1. Software Verification and Validation (SV&V)

A detailed software verification plan is required. The plan describes the activities to be conducted after each phase that demonstrate the requirements specification is met. Software verification addresses testing, independence, documentation and review of results.

A software test specification and test report are required as part of the software verification plan and detailed content is specified.

An integrated system verification test report is specified.

Computer system validation is required, and specific types of functions and properties are specified. A computer system validation report is required. Results must be maintained in an auditable form.

Identifies criteria to be checked for, in each phase document. These are listed, but not defined.

B.3.6.2. Software Quality Assurance (SQA)

An SQAP is required. Inspections are cited as are "dispute resolution mechanisms."

B.3.6.3. Software Configuration Management (SCM)

Documentation must be under configuration control.

B.3.6.4. Software Hazard Analysis

Not addressed.

B.3.7. Project Planning and Management

Indirectly addressed through general principles that govern the software project. These include principles on tasks of phases, SQA, error reporting and correction, and configuration control. There is no requirement for a project plan identifying how these principles will be implemented.

B.3.8. Procurement Concerns

Under the Verification section recommendations are made regarding independence and type of personnel skills needed to review the products of each phase.

B.3.9. Presentation

The content specification for documentation exceeds the "what" and becomes a "how." One example is that the trace assertion specifications for module interfaces may be considered a methodology for performing traceability. There may be other methods equally acceptable.

B.3.10. Supplemental Information

This document was prepared by Dr. David Parnas under the sponsorship of the Atomic Energy Control Board of Canada. The version reviewed was issued in March 1991; comments on it were due in September. NIST has not yet received information on the document's current status.

B.3.11. General Comments

Response to the First Set of Principal Questions:

- This document provides rigorous requirements for software at the highest level of criticality, however, there is no requirement for hazard analysis.

Response to the Second Set of Principal Questions:

- This document provides a rigorous way of showing that a safety system conforms to its requirements, and the requirements of the standard are sufficiently precise to show conformance to the document.

- Auditors will need to be trained in the use of formal specification languages to be able to audit for compliance with the standard to understand:

    - the requirements they must audit for
    - how well those requirements have been implemented

The following are the review comments that the authors of this report sent to Ontario Hydro:

"The draft provides a rigorous way of showing that a safety system conforms to its requirements, and the requirements of the standard are sufficiently precise that it is possible for a developer to show conformance to the standard. It should be noted that this rigorous way may also be considered a methodology specification.

"There should be a requirement for some type of hazard assessment, i.e., a description of the hazards to be protected against. This should be described in terms of the monitored and controlled variables. This requirement could be placed in section B.1 which states the general principles to govern all work in a software project.

"It is difficult to claim that a system is safe without some definition of safety.

"The glossary should contain a general definition of safety.

"Suggestions for formalizing the definition of safety to nuclear power stations are the following:

> The scope of this document identifies specific applications that fall under the requirements of this proposed standard. The requirements document for any of the systems to which this standard applies must identify the safety principles the system must meet.
>
> There should be a "safety policy" describing how the hazards identified in the hazard assessment are countered by the system. The policy should be stated in terms of the monitored and controlled variables. A formal statement of the safety policy should then be defined, with convincing argument that the informal to formal policy mapping is correct. If S is a predicate defining the safety policy, then, using the terminology of the proposed standard, it should be possible to show that REQ(m') => S.

"As noted in Section 9, the use of traces should be considered only as an example of an acceptable method; other rigorous techniques should be acceptable. Sections 9 and 10 therefore need to be moved to an appendix, since they specify a particular method. They should be replaced with paragraphs that specify what is required, but do not prescribe a particular method."

B.4. Dependability of Critical Computer Systems 2 - Chapter 1: Guidelines to Design Computer Systems for Safety [EWICS2-1]

B.4.1. Levels of Criticality/Assurance

Suggests associating design constraints with level of criticality (separation of control, protection functions, redundancy, diversity).

B.4.2. Lifecycle Phases

System design: Chapter addresses elements that must be considered in the design of the system; principles should be practices; response mechanisms to be used against failures; principles for man-machine interfaces.

B.4.3. Documentation

Not addressed.

B.4.4. Required Software Functionality Against Hazards

- o self-testing of critical functions
- o memory, storage integrity checking
- o redundant hardware components
- o redundant software components
- o fault tolerance, automated error recovery
- o diversity (e.g., product diversity for hardware, functional diversity for software)
- o safety from single failure
- o safe state when "external" problems happen
- o safety/security checking functions are non-bypassable
- o parameter checking
- o checks on sequence of operations

(NOTE: Required on-line software checks *could* include several of above; also required integrity checks.)

B.4.5. Software Engineering Practices

System design: Chapter concentrates on the system but does address some software engineering practices (in more detail than template provides for). Requires proven modules and monitoring tools.

B.4.6. Assurance Activities

B.4.6.1. Software Verification and Validation (SV&V)

Requires a system validation plan for demonstrating safety enhancements of the design, and to evaluate design for potential hazards. Should be by independent team, during production (licensing evaluation later).

B.4.6.2. Software Quality Assurance (SQA)

Detailed for system design. The QA section requires a plan and an IV&V assessment of design during the design process.

B.4.6.3. Software Configuration Management (SCM)

Detailed for design.

B.4.6.4. Software Hazard Analysis

A criticality analysis and risk analysis are included. Techniques are specified.

B.4.7. Project Planning and Management

Detailed.

B.4.8. Procurement Concerns

IV&V assessment.

B.4.9. Presentation

Chapter 1 is intended as a guideline and uses the word "should" instead of "shall." Material in each section addresses the title of the section. This does cause some redundancy due to repeating engineering techniques or required software functionality against hazards that are similar (e.g., in Section 2.4 some check mechanisms are repeated). The language is clear and concise; there should be no misunderstanding of intent.

B.4.10. Supplemental Information

None.

B.4.11. General Comments

Security is addressed weakly in Chapter 1, only as a login precaution.

Response to the First Set of Principal Questions:

- The design phase is thoroughly covered.

Response to the Second Set of Principal Questions:

- Conformance generally means *all* steps completed, but the techniques to support the steps may be considerably tailored.

- For conformance, NRC auditors (or vendors) need to demonstrate that each and every step has supporting procedures and that the procedures have been implemented. Note that each step may consist of a selection of techniques identified in the chapter. NRC should decide if *any* selection is acceptable (one, some, or all) and if not, then develop a uniform checklist from which all NRC auditors operate.

B.5. Dependability of Critical Computer Systems 2 - Chapter 2: Guidelines for the Assessment of the Safety and Reliability of Critical Computer Systems [EWICS2-2]

B.5.1. Levels of Criticality/Assurance

The chapter includes two examples of criticality levels from other documents. These two (nuclear and civil aviation) are quite different and are included as examples only.

A proposal from the nuclear industry ("Licensing issues Associated with the use of Computers in Nuclear Power Plants" Euratom Report by R. Bloomfield and W. Ehrenberger) gives seven levels of criticality. An interesting aspect of this breakdown is that levels are defined in terms of both the effect of a failure and the manner in which the protection is implemented. For example, systems that act for the protection of human life and the environment through automatic actions are placed in the highest criticality level, while systems that act for the protection of life and environment through influencing operator actions are placed a level below that of the first case.

Another chart displays criticality levels from various civil aviation standards, such as RTCA178A. The levels in all of these standards are determined by the effect on aircraft and occupants of failure conditions.

The chapter does not associate particular practices or assurance techniques with the levels from these standards. There is, however, a reference to a separate volume [EWICS3] that describes a variety of techniques for the assessment and analysis of safety and the conditions necessary for their use. The techniques mentioned include criticality analysis, failure modes and effects analysis (FMEA), FTA, event tree analysis, Markov modeling, preliminary system hazard analysis, sneak circuit analysis, software reliability modeling, and system hazard and operability studies.

B.5.2. Lifecycle Phases

The guideline touches on all lifecycle phases, using the following four simplified phases: project proposal; reification (conversion into concrete item) and construction; acceptance and commissioning; and O&M. The author advocates using the same models or methods in all phases, with greater detail in each successive phase, and with the same personnel involved. Very little guidance beyond this is given with respect to lifecycle phases.

B.5.3. Documentation

Although the chapter is written from an advisory, rather than mandatory, standpoint, the discussion of assessment implies that all of the documents from the template will be reviewed.

Much more specific guidance is given as to the documentation of the assessment process itself. The following are required as evidence in the evaluation and assessment:

- assessment plan - details the phases, deliverables, personnel, management, and review criteria used in assessment
- quality system - describes the quality system of the *assessing* organization, which must be externally approved by a recognized body to a recognized standard such as ISO9000
- external quality audit - a copy of the current approved certificate
- internal quality audit - copies of the latest internal audit

Specific products from the assessment process are also required:

- safety criteria and attributes - details of these characteristics, with an explanation of how they have been derived from the system level requirements
- safety case for the target system - contains three parts quality and experience - the QA audit and relevant operating experience (the QA audit is at the system level but includes analysis of software
- analysis - selection of analysis techniques, with justification for the choices, and analysis results
- risk management - details of the risk management

B.5.4. Required Software Functionality Against Hazards

This chapter gives guidance for assessment procedures, so specific functionality for the application system is not addressed.

B.5.5. Software Engineering Practices

Particular aspects of design and construction are not discussed, as the chapter concentrates on assessment of completed software.

The chapter does not address the rigor of the specifications except to note that mathematical models and formal specifications "may" be used. A note about design analysis says that "If rigorous software development methods are used, such as the Vienna Development Method (VDM), then the arguments for the correctness of the refinement can be analyzed."

The question of critical component isolation is not addressed. Specific aspects of software development, such as modularity, programming languages and practices, are not addressed.

B.5.6. Assurance Activities

Assurance activities such as V&V, QA, and CM for the computer system are not specifically addressed, except to note that these activities should be considered in the assessment. QA of the assessment organization itself is addressed, with reference to ISO9000.

B.5.6.1. Software Verification and Validation (SV&V)

See Section B.5.6 above.

B.5.6.2. Software Quality Assurance (SQA)

See Section B.5.6 above.

B.5.6.3. Software Configuration Management (SCM)

See Section B.5.6 above.

B.5.6.4. Software Hazard Analysis

Not addressed.

B.5.7. Project Planning and Management

The chapter is essentially a guide on how to plan and conduct an assessment. It references other EWICS books, and recommends ISO9000. The requirements/recommendations on the assessment process are sufficiently detailed that an organization could add details for a particular application area and come out with a sound assessment process.

B.5.8. Procurement Concerns

Independent evaluation of the assessment organization is specified, with reference to ISO9000.

B.5.9. Presentation

This is strictly a guidance document, rather than a standard. As such, it intermingles terms "should," "shall," and "must," but this is acceptable in a guideline. The organization is effective and the material is useful, but the guidance is more generic and high-level than is appropriate for a standard.

B.5.10. Supplemental Information

None.

B.5.11. General Comments

Response to the First Set of Principal Questions:

- The chapter is adequate as high-level advice on how to do assessments. To be useful for particular types of software, it should be supplemented with more specific guidance designed specifically for the application area.

Response to the Second Set of Principal Questions:

- The chapter is intended to be generic so that it may be tailored for a specific application. Therefore, this set of questions does not apply.

B.6. Dependability of Critical Computer Systems 2 - Chapter 3: A Questionnaire for System Safety and Reliability Assessment [EWICS2-3]

B.6.1. Levels of Criticality/Assurance

This chapter provides an interesting table of factors that contribute to complexity in systems. Seventeen factors are given, with five levels of complexity for each factor. For example, if the factor "frequency of change of role" is "never," the system is given a low complexity rating for that factor; if the frequency is "continually," the system is rated as 5 on a five point scale for complexity. The "scope of safety considerations" factor ranges from a low-level of "risk to components or data only" to a high-level of "life of external personnel." An estimate of the complexity of the system can be computed by rating the system on each factor.

Unfortunately, there is no attempt to associate the complexity levels with specific practices or assurance techniques.

B.6.2. Lifecycle Phases

All lifecycle phases are covered by the questionnaire. The questionnaire is divided according to the following lifecycle phases: project planning and management, system requirements specification, design, coding and construction, integration of hardware and software, V&V (hardware, software, system), qualification, and O&M.

B.6.3. Documentation

Documents are specified for each lifecycle phase. The developer is not required to follow a specific format. Contents of the documents are not specified directly, but the questionnaire indicates factors that must be considered and documented at each lifecycle phase.

The questionnaire checks for conformance to various standards such as IEEE1012.

Some traceability is implicit in the questionnaire, but the chain of evidence from requirements through code is not treated in sufficient detail. For example, a section entitled "Translation of functional requirements" handles requirements to design specification consistently with the following question: "Are the functional requirements clearly and accurately translated into component specifications?" No further question is given regarding the level of rigor used in the requirements-to-specification correspondence.

B.6.4. Required Software Functionality Against Hazards

The questionnaire includes a comprehensive set of functions for fault detection and handling, including checks for automated error recovery, self testing, redundant components, clock-drift checking, and many others. No major category of critical functions is ignored.

B.6.5. Software Engineering Practices

The questionnaire gives preference to formal specifications over informal ones, and also checks for modularity and isolation of critical components from other parts of the system. High-level languages are preferred. Significant attention is given to coding practices that are hard to assess or likely to lead to errors. The coding practices section is among the most comprehensive of the standards examined for this report.

B.6.6. Assurance Activities

B.6.6.1. Software Verification and Validation (SV&V)

A very comprehensive questionnaire is provided for reviewing V&V of both software and hardware, covering all phases of the lifecycle. The questionnaire checks for conformance to various standards such as IEEE1012.

Most of the questionnaire questions are general, high-level considerations. For example, design to code correspondence is covered by the question "What techniques has the verifier used to ensure that the code reflects the intent of the design?" No specific technique is given, and no specific level of detail is required.

Requirements to design correspondence is handled with a similar question: "What techniques has the verifier used to ensure that the design reflects the requirements of the engineering specification?" Specific verification techniques are not given and design verification is not categorized by the rigor of the specification and verification techniques. There is, for example, no enumeration of assurance techniques such as static analysis, informal design verification against informal requirements, informal verification against formal specifications, formal verification against formal specifications.

Software testing is covered in the section on SV&V. The questionnaire is quite detailed, and checks for specific types of testing, such as path coverage, statement coverage, linear code sequence and jump coverage. This categorization is much more detailed than the requirements and design correspondence questionnaire items.

B.6.6.2. Software Quality Assurance (SQA)

SQA is covered in EWICS2-4.

B.6.6.3. Software Configuration Management (SCM)

Not addressed.

B.6.6.4. Software Hazard Analysis

Hazard analysis is touched on only briefly, essentially in a yes-no manner. For example, one question asks "Was the functional specification of the target system analyzed to see whether it includes any intrinsic hazards?" Criticality analysis is also covered by a binary question: "Does the functional specification classify system functions according to their safety criticality?" FTA and FMEA are noted. These are presumed to be at the system level. There does not appear to be a *software* hazard or safety analysis requirement.

The chapter does contain some very useful questions regarding safety. One aspect of safety overlooked in some safety standards is a requirement for a set of criteria that define what it means for the system to be safe. This questionnaire includes a check that such criteria have been developed. Safe system states and the procedures that move the system into these safe states are also checked for. A particularly valuable aspect of the questionnaire is the inclusion of a series of questions dealing with human interfaces and ergonomics. User training, different classes of user, physical constraints on the user, help facilities and their relationship to user experience are among the human factors covered.

B.6.7. Project Planning and Management

Not addressed.

B.6.8. Procurement Concerns

A separate section of the questionnaire deals with qualification of the system at installation time. Questions in this section cover aspects such as interaction with the licensing authority, acceptance testing, and dealing with discrepancies. Formal contractor capability assessment is not addressed, although the question "Is company QA sufficient?" reminds the evaluator that contractor capability should be considered.

B.6.9. Presentation

The chapter is thorough and well organized.

B.6.10. Supplemental Information

None.

B.6.11. General Comments

Response to the First Set of Principal Questions:

- This questionnaire serves its intended purpose well. It can be used to fill out a framework for evaluators to use, but it should not be used in its entirety, since many questions may not apply to a particular application or organization. It is necessary to incorporate material from other sources as well. The chapter is not sufficiently complete to fill out such a framework using only items from the questionnaire. Software hazard analysis questions need to be supplemented using guidance specific to software hazard analysis methods. Requirements-to-design and design-to-code correspondence are also dealt with in a cursory manner. Specific guidance for evaluators is needed in these areas as well.

Response to the Second Set of Principal Questions:

- This chapter is intended to be generic so that it may be tailored for a specific application. Once the

tailoring is completed for a particular application, the questions are specific enough for an auditor to determine if they have been satisfied.

B.7. Dependability of Critical Computer Systems 2 - Chapter 4: A Guideline on Software Quality Assurance and Measures [EWICS2-4]

B.7.1. Levels of Criticality/Assurance

Not addressed.

B.7.2. Lifecycle Phases

Not addressed.

B.7.3. Documentation

Not addressed.

B.7.4. Required Software Functionality Against Hazards

Not addressed.

B.7.5. Software Engineering Practices

See Section B.7.6.2 of this report for quality attributes.

B.7.6. Assurance Activities

SCM, SV&V, and testing are beyond the scope of this chapter.

B.7.6.1. Software Verification and Validation (SV&V)

See Section B.7.6 above.

B.7.6.2. Software Quality Assurance (SQA)

This brief chapter gives general guidance on SQA and measures to be used in estimating quality. The guidance is high-level and generic; no specific measures are defined. The focus is on helping the reader specify in measurable terms attributes to be considered in SQA. Examples of quality attributes include performance, usability, availability, reliability, maintainability, integrity, and adaptability. A systematic method for specifying estimation and measurement techniques for these attributes is given. Among the items to be considered in these specifications are the scale of measurement; the measurement (or estimation) technique; worst-case specification; planned value for an attribute; record case specification; past-level specification; and current-value specification.

No standard is required, although there is an extensive list of applicable standards that could be used by organizations involved in defining SQA procedures.

B.7.6.3. Software Configuration Management (SCM)

See Section B.7.6 above.

B.7.6.4. Software Hazard Analysis

Not addressed.

B.7.7. Project Planning and Management

Organizational and managerial aspects are touched on briefly in a subsection on Acceptance Criteria.

B.7.8. Procurement Concerns

Not addressed.

B.7.9. Presentation

The presentation is generally clear, and helpful examples are provided.

B.7.10. Supplemental Information

None.

B.7.11. General Comments

The chapter is adequate as a general guide on how to put together a SQA manual. It is lacking in specifics, however. The most effective way to use this guide is to supplement it with more specific information on SQA, such as might be found in IEEE standards.

Response to the Principal Questions:

- The principal questions do not apply to this chapter.

B.8. Dependability of Critical Computer Systems 2 - Chapter 5: Guidelines on the Maintenance and Modification of Safety-Related Computer Systems [EWICS2-5]

B.8.1. Levels of Criticality/Assurance

Not addressed.

B.8.2. Lifecycle Phases

This chapter is a guideline specifically for the operations and maintenance phase. The guide specifies detailed procedures for system maintenance, including corrective (fixing errors), adaptive (modifications to fit new circumstances/environments), and preventive maintenance. Both hardware and software components are addressed.

B.8.3. Documentation

Not addressed.

B.8.4. Required Software Functionality Against Hazards

Not addressed.

B.8.5. Software Engineering Practices

Not addressed.

B.8.6. Assurance Activities

B.8.6.1. Software Verification and Validation (SV&V)

Not addressed.

B.8.6.2. Software Quality Assurance (SQA)

Not addressed.

B.8.6.3. Software Configuration Management (SCM)

Not addressed.

B.8.6.4. Software Hazard Analysis

Not addressed.

B.8.7. Project Planning and Management

Not addressed.

B.8.8. Procurement Concerns

Not addressed.

B.8.9. Presentation

This chapter is quite clear regarding documentation and organizational aspects of maintenance. A comprehensive set of data flow diagrams showing processes and documents adds to the presentation.

B.8.10. Supplemental Information

None.

B.8.11. General Comments

This guide should be viewed as a framework that can be adapted to a particular organization. Its strengths are in detailing the kinds of documentation and management needed for effective system maintenance.

Response to the First Set of Principal Questions:

- Although this chapter is designated as applicable to safety-critical systems, it contains nothing that is unique to the needs of such systems. That is, it could be considered a generic maintenance guide. The guidance is extensive, and is appropriate, but it consists essentially of following the standard lifecycle phases for software maintenance as for initial development. Such guidance is consistent with that advocated by most experts on software maintenance. Maintenance should be treated as continued development, so the same lifecycle phases should be followed. The principle difference in maintenance is that special consideration must be given with respect to the dependencies between system functions. Modification of one component can often have unexpected consequences for components that depend on the one modified. This guide provides a process for management to use to help identify and track the impact of changes on the system.

- It is short on details of how maintenance procedures should be carried out, but this provides flexibility that can be used to make the guideline applicable to a variety of organizations. For example, regression testing, or re-testing after modification, is required at the unit, subsystem and system levels, but the type of testing is not specified.

Response to the Second Set of Principal Questions:

- The chapter includes a conformance clause that details what is necessary for an organization to claim conformance to it. Conformance requires showing that required documents are produced, and that the organization includes the necessary management structures and operational groups.

B.9. IEC 880: Software for Computers in the Safety Systems of Nuclear Power Stations [IEC880]

B.9.1. Levels of Criticality/Assurance

No levels of criticality are specified in the body of the standard.

In Appendix B, requirements and recommendations for design and programming practices are identified by priority or importance (1,2,3). Limited direction is provided as to "priority or importance of system" in Section 5.3.2. While Appendix B attempts to specify elements of design and code, as well as techniques to be used in design and code, the table has many weaknesses in its presentation (see Section B.9.9 of this report).

B.9.2. Lifecycle Phases

This standard states that it addresses all phases of the lifecycle. In Section 2 the phases are defined as typically being requirements, design, implementation, test, installation and check-out, and O&M. However, in Appendix F (List of Documents Needed) the phases are system requirements, software requirements, software design, coding, hardware/software integration, computer system validation, commissioning and exploit/maintenance. These inconsistencies become significant when verification is discussed in the body of the document (see Section B.9.5.1 of this report).

For each phase of the lifecycle the standard identifies elementary tasks and generation of documents. It requires every product checked at termination of each phase, and a formal review at termination of each phase. SQA is to be run in parallel with the phases.

This standard has a chapter on maintenance; comments on this chapter appear under Supplemental Information (see Section B.9.10 of this report).

B.9.3. Documentation

Since this is an international standard, its statement in Section 5.4.4 regarding applicable standards should allow for standards from accredited organizations or those approved in the agreement to use this standard, rather than only national standards.

Twenty documents (planning specifications, reports) are required by the standard or its appendices. A definition for the software performance specification should appear in the glossary; this is the first occurrence found of this document name in over 70 related standards and guidelines. It is important that the scope of each required document be explained in a standard, and possibly listed in the glossary.

This standard does not include: an installation document, a maintenance manual, project planning documentation, and source code.

Appendix A contains "acceptable content" for the software requirements specification (SRS) but the requirements of this document are very confusing as to when they are documentation requirements, requirements for functions the system must contain, or requirements for software engineering practices that must be used. While the appendix requires presence of readability, testability, maintainability, and feasibility, no criteria are provided for the attributes.

B.9.4. Required Software Functionality Against Hazards

Functionality is specified in requirements and throughout the document. These include: continuous self-supervision; no single failure can block function to prevent release of radioactivity; automatic actions in case of failure; and, periodic testing.

Other functionality is indirectly required by requiring that programming languages not prevent functions for error limit constructs, translation-time checking, run-time type and array bound check, and parameter checking.

Appendix A addresses basic functions that must be included, or things that must be identified. For example, emergency shutdown (A2.1.2) is required in the appendix. But Section A2.1 contains requirements for the computer system specification documentation. The functional requirements should be in Section 4.1. Rules for their description may be contained in A2.1.

Section 4.8 describes some safety functions the system must meet.

While the standard should not specify the rules, it should require that a nuclear safety system provide for the following:

- ability to execute a corrective action of wrong command issued or a command misinterpreted or executed out of sequence (e.g., operator may use programmed function keys and may hit key for "GO" when the key for "STOP" was intended; some function, perhaps a verification required by operator that proper key was hit, should allow a correction, especially when consequence of wrong function execution is hazardous)
- functionality for fail-safe operation like bringing the system into a safe configuration in case of an anomaly, e.g., power down, remote shutdown
- rules for when the operator may (can) override software
- use of warning devices relative to states of other safety systems
- diagnostic maintenance features should be defined
- exception alerts for reporting on out of bound parameters
- procedures for system console failure

- protection for access to the memory region of critical functions
- protection against unauthorized or inadvertent access to code
- function to verify that outputs of critical algorithms are all right (parity or other checks); actually this feature may have been intended by the requirement for "safe output"

Minimum attention is paid to functional requirements for computer security concerning protection of data, access to the system, and inadvertent change to the software during development, maintenance and operation. Once again, the SCMP should provide requirements for how SCM will provide protection for computer security functions.

B.9.5. Software Engineering Practices

This standard provides guidance on many software engineering practices in the following places: phase guidance, documentation guidance, and Appendices B and D. One weakness is that computer security is minimally addressed. Documents should include software engineering practices that ensure computer security (e.g., modularity).

With one exception, this standard does not provide for the assurance of any support software that may be needed for the safety system software, nor does it provide for assurance of any automated tools used to develop or test the system. The statement "Hardware and software tools used for this validation need no special verification." (p. 33) is the exception. However, the authors of this report disagree with this statement. Software tools for testing are becoming more sophisticated and may be more complex than the safety system they will be used to test. When their results are to be trusted, then they must have a similar level of assurance as the system they will be used on.

The standard attempts to specify features the design should have, but these features are intermingled throughout the software requirements in such a manner as to confuse the user (see Section B.9.9 of this report).

A formal specification language may be used, but the requirements must also be presented in (normal) language. Today, standards for critical systems are requiring formal specifications for the critical components. An additional requirement for a normal language representation is acceptable.

In several places in the standard, reference is made to isolation of critical components and to modularity.

There is a requirement that "The final source program should be readable from start to end." This may have been intended to mean that a higher order language should be used. Appendix B gives rules on use of language features, such as branches and loops, subroutines, nested structures. It also has a section on assembler code, implying assembler languages are permissible, which could be in contradiction with the readability requirement.

Appendix D provides recommendations for languages, translators, and linkers. There is a recommendation for level 2 importance that high-level languages should be used. Many recommendations are well-stated, but others are poorly posed. Examples of poor statements include:

- "Readability of produced code is more important than writeability during programming."

- "Produced programs should be easy to maintain."

In several places in the standard reference to quality attributes is made. But no criteria are provided, nor is there is a reference to criteria in other documents. Examples include:

- "Source readable." (This is an ambiguous statement.)
- "Good documentation." (This is an ambiguous statement.)
- "Produced programs should be easy to maintain." (No definition of maintainability.)

This is an example of a somewhat more clearly stated requirement in the document:

- "Simple and easy to understand structure (avoid tricks, recursive structures, and unnecessary code compaction)."

While the standard does not appear directly to control practices, its language requirements do address controls. For example, "Language must not prevent: error-limiting constructs; translation-time type checking; run-time type and array bound check, and parameter checking." This statement is unclear but appears to be saying that the converse (use of a language that prevents these) is a discouraged practice. The reason is not because something is hard to analyze or verify during development, but because during operation the goal is to check these things, which cannot be done if the language prevents them.

Another ambiguous statement, "Problem oriented language is `strongly preferred' over machine oriented language," seems to contradict Appendix B's statement permitting assembly language.

Some additional software engineering practices for design and code may be:

- no unused executable code
- no unreferenced variables
- rules on memory overlays
- rules on flags

B.9.6. Assurance Activities

B.9.6.1. Software Verification and Validation (SV&V)

The standard states that software verification ends each lifecycle phase, but the wording is not sufficiently strong to require software verification activities of requirements specifications, design, and code. Mention should have been made of traceability, correctness, completeness, and consistency, at a minimum.

Every verification step or critical review results in a report.

The standard requires that software functional requirements and software design are verified for adequacy. Even if defined, adequacy is not a sufficient attribute to provide for software verification of requirements and design (adequacy is not defined in the standard).

Compliance of coded software to the design specification is addressed by the Verification section (which addresses software verification).

The software verification plan addresses software verification strategies (guidance for software testing is provided in Appendix E; code inspection and use of mathematical proofs are supplementary techniques), selection of test equipment, execution of software verification, documentation of software verification activities, and evaluation of software verification results.

The software verification plan may be executed by an independent agent. There is no requirement for independence and the developer may develop the Software Verification Plan.[6] The developer may prepare the test plans, and IV&V may perform the tests. However, an independent group executing test cases prepared by the developer does not necessarily provide the benefits of complete IV&V, in which situation the independent group develops the test strategy and cases. Testing falls under the Verification section, and has an appendix describing its features. However, the types of testing should be separated. Logic testing is described somewhere in the middle of testing recommendations dealing with system test strategies.

A confusing statement is that the design verification subsection addresses "the respect of quality requirements."

No mention is made in the design verification report of verifying that *all* requirements have been included, hence the report does not ask for a trace to show completeness either. Items are to conform to design standards but those were not clearly identified in Section 5.

Test specifications should be required during the requirements phase with final elements produced during design. Also, no mention of independence appears.

[6]There are separate definitions for verification and validation, yet for all practical purposes the verification section appears to cover both in terms of the template in Appendix A. The use of the word verification is another example of the misunderstanding regarding the terms "*system* verification" and "*software* verification and validation."

In this standard, code verification begins with module testing. In general SV&V of code includes at least a code review, if not inspections or other static analyses for the most safety-critical requirements. There is emphasis on showing no unintended functions at the module level. This emphasis does not reappear at software integration where it is especially important.

Full SV&V activities are not addressed.

The software test specification and the software test report are required testing documents. But, this standard addresses test in general. There are several types of testing that may need different test requirements. Part of the problem is that the IEC880 is a system standard in which software is a part. But software testing itself is a major activity that requires rigorous test planning for all types of testing: module (component), integration of software, integration of hardware/software, system, and acceptance test. Test documentation should include the test plan (e.g., management, resources, objectives, schedule), test cases and procedures, and test reporting and error resolution procedures and monitoring. Error resolution is addressed in Section 7.6 for the system but it should be addressed during software-specific phases.

Appendix E provides some tutorial information on testing and attempts to include some information on static analyses. It does not list all possible test strategies or conditions for which to test (e.g., stress testing). It would be further improved by adding some error analyses applied to test results. Appendix E treats some static analyses as testing. The appendix could use a significant amount of rework to make it useful.

Periodic testing is required. "The general principles of IEC Publication 671: Periodic Tests and Monitoring of the Protection System of Nuclear Reactors, are applicable." (p. 41).

Scope and objectives of test are defined, such as:

- A test program which includes functional tests, instruments, checks and verification of proper calibration and response time test.

- Verification of the basic functional capabilities of the software (all basic safety functions, detect failures not revealed by self-checking provisions, major non-safety related functions).

B.9.6.2. Software Quality Assurance (SQA)

SQA is required, but no standard is proposed. Section 3.1 does require review, documentation and reporting on analyses at the end of each lifecycle phase. There is a systematic check of each product after every phase (implication: review?). Generation of appropriate documents (required documents listed in Appendix F) is to occur during each phase. Reviews and audits are not fully specified.

B.9.6.3. Software Configuration Management (SCM)

CM is required only at the system level. SCM should be required for the software. The change control procedures required during maintenance should be part of SCM during software development. This lack of SCM could be critical for safety microprocessors in nuclear power plants. While the software in these controllers may not be large, there can be changes made during development and after the software is in operation. Since CM is specified at the system level only, software changes and version control may likely be neglected under this standard. IEEE828 provides basic requirements for SCM during development. Perhaps a revision of IEC880 could cite this standard.

B.9.6.4. Software Hazard Analysis

Software hazard analysis is neither mentioned nor required.

The authors of this standard may assume that by expressing some safety functions, they have covered all possible hazards. This may not be true, and software hazard analysis can also locate potential hazards for specific designs. Another assumption may be that a system hazard analysis has already been performed which is the reason the standard is invoked. However, results of a system hazard analysis need to be analyzed relative to the software implications, and results of that analysis need to be taken into account during development activities.

B.9.7. Project Planning and Management

Not addressed.

B.9.8. Procurement Concerns

The software verification plan must be written such that an independent agent may execute the tests. This is not a complete definition of independence, for independence may also mean that the agent develops the test plan.

B.9.9. Presentation

It took many readings of this standard to understand its organization and especially its requirements. It was difficult to sort out. Even though this document contains a great deal of important information, it was hard to determine if it addresses appropriate software engineering practices, functionality issues, and assurance practices.

This standard should have an opening section describing the sections of the standard and especially what parts of the standard are binding. Repetitive information appears in several places. It becomes difficult for an auditor to sort out the items to be audited.

The following tables and charts are difficult to understand without rewriting and rearranging them:

- The table on page 21 is not clearly described relative to its relationship to the recommendations of Appendix B.

- The tables in Appendix B can be simplified; the criteria are the most important part and do not get the "visual" emphasis. The middle sections of the chart are primarily white space and detract from the criteria.

- Appendix B gets operating practices (of the developer) mixed up with activities and with design practices. Appendix B lists criteria for meeting objectives of practices for the design and code of the system. Each element is assigned a number (1, 2 or 3) for its importance. While the total list of practices is reasonable (see Section 2.4 of this report), a developer needs to know how to select them, and an auditor needs some direction for knowing if the developer has selected and properly implemented the "right" practices. And, the language of most of the descriptions needs to be made more concise and specific.

- Table E is laid out in two different formats, which becomes confusing. Again, different types of test topics are intermingled.

The standard intermingles different kinds of requirements or recommendations which will make it difficult for both developer and auditor to ensure that these are met. Sections 4.1 - 4.7 describe the types of information that must be included in the requirements, but 4.8 describes safety functions that the system must meet. Section 4.9 presents more information about how to present the requirements. Section 4.8, because it is a functional requirement, not a development or SQA procedure, should be a section highlighted by itself. In Section 5, functionality features, design practice, and quality features are intermingled. For example, the fact that system design shall include self-supervision is mentioned in the same subsection that good documentation is required. These are different types of requirements, and would be verified by entirely different mechanisms. Both Appendix A and B have similar problems.

Functional requirements are mixed in with design practices which are things to consider, and so on. Requirements for functions of the system and general procedural requirements should be separated into sections. In the sections on software requirements, features to be built into the design, or principles to be invoked by the designers, are listed. Any auditor will need to sort out the requirements of the standard before the vendor product can be checked for compliance. Unless NRC does this for all auditors, it is reasonable to assume that auditors will interpret and sort out the requirements differently.

This is more of a guidance document than a rigorous standard; conformance cannot be shown. With some work, however, "standard" items could be separated from recommendations, and for all of these, categories need to be established. This document is confusing because the structure of the document is not clear.

A few examples of weak wording include the following:

- "Good documentation" is not an acceptable requirement; a standard or content should have been specified.

- The statement "the program should be written to allow easy testing" (p. 19) does not provide an auditor with any measure. Rather, the intended recommendation that every requirement in the specification must be stated in quantitative terms so its implementation may be measured `pass' or `fail' should be stated explicitly.

- "The respect of quality requirements." (p. 25) is a meaningless sentence.

- What does "the computer shall not mislead the operator" (p. 47) mean?

- The statement "requirements shall be easy to understand" (p. 53) should be quantified with reference to a readability tool or measure, or some other standard could be referenced, e.g, IEEE830.

- Appendix B: "If possible, complete and correct error recovery techniques should be used" (p.77). This is an ambiguous statement.

B.9.10. Supplemental Information

This standard addresses the system's hardware/software integration, computer system validation, maintenance and modification (explained in some detail but configuration control is explained poorly).

For changes during maintenance, procedures are defined for requesting a change and executing a change, including SV&V. The weakness is that the document tries to treat both maintenance change and change during development under the maintenance section of the document. Rules for change control during development should be a part of the standard's requirements for development; it is too easy to misunderstand the intent of these procedures. As stated, neither developer nor auditor may readily determine when a development interim product comes under control. The standard should insist that a project establish baselines, authority to approve change, status accounting and other SCM activities in an SCMP for the development and also for maintenance after delivery. Note that clause 9.1.3 would be easier to implement if traceability and an analysis report had been required during development.

B.9.11. General Comments

Usually, the body of a standard contains the mandatory requirements. In IEC880, the appendices appear to contain requirements also, and parts of the body appear to be recommendations. The body and the appendices are not always consistent (e.g., lifecycle phases).

Levels of assurance are only hinted at, and "requirements" appear in the form of "should," "shall," "may" priority. NRC auditors need to develop a consistent list of requirements from which they examine software products to provide assurance of NRC systems.

Response to the First Set of Principal Questions:

- IEC880 contains minimal requirements for documentation content. It contains detailed requirements for the functionality a nuclear safety system should contain but it weakly addresses requirements for computer security. While it identifies software engineering practices for the software design and code, it does not require either formal specifications for the software requirements or rigorous static software verification analysis on the requirements, design, and code. Its requirements regarding test activities and error analysis are minimal. It considers system CM but not specifically *software* CM. SQA activities levied on the software developers are not rigorous. Furthermore, quality attributes have not been defined. From an engineering perspective, this standard requires many software functions that are critical to safety systems but does not necessarily provide sufficient requirements for the assurance of the system.

- Since unintended functions are a serious concern in nuclear safety systems, it is clear that insufficient attention is paid to requirements for documentation of the traceability analysis in both the forward and backward direction from the system requirements, to the software requirements, to the software design to the code.

- Because this standard does attempt to address both functional requirements and software engineering practices, it has more to offer than most standards. Yet, this positive attribute becomes a negative one because of the omissions and weaknesses of the standard. The concern is that a developer may have overconfidence in this standard and not consider other practices.

Response to the Second Set of Principal Questions:

- This was a difficult document to review. Because of the document's style (intermingling functional requirements, documentation requirements, system requirements, software engineering practices) and because many requirements were ill-defined, any two auditors will probably examine different features of the same system.

Recommendation:

- The document could be reorganized. Separating types of requirements in the same section would be a major help.

- Example: Appendix A includes many requirements on the design. In several sections, functions are specified that the software requirements must include. Then, requirements which include design principles or functions the design must include, are given. This appendix would be much easier to use, for both a developer and an auditor, if all principles or functions to be considered by the designers were presented in one section of this appendix. Example: Design practices and functions are contained in A2.2 of this standard, the second set of requirements and in A2.8, Section A2.8.2.

B.10. 45A/WG-A3(Secretary)42: (3rd Draft) Software for Computers Important to Safety for Nuclear Power Plants as a Supplement to IEC Publication 880 [IECSUPP]

B.10.1. Levels of Criticality/Assurance

In Section 3.1, each potential software malfunction must be classified according to its safety importance, i.e., the severity of its consequences.

Level of diversity depends on reliability requirements and their ability to be demonstrated. Levels are referred to but no direction is given on how to determine them.

B.10.2. Lifecycle Phases

Requirements, design, implementation and maintenance. The section on maintenance addresses these phases: specification, design, coding, and validation.

B.10.3. Documentation

For requirements, either formal notation or informal methods with written definitions of requirements, algebraic formulations, data tables, logic diagrams. Suitable documentation in the national language is needed when using formal methods.

A maintainability plan is required and described (supplemented during design phase).

Project management documentation is mentioned.

B.10.4. Required Software Functionality Against Hazards

The publication[7] does specify some functional goals to achieve when there is failure caused by other system functions:

- reversion to alternate success paths
- fault tolerance
- graceful degradation
- fail safe reaction

[7] At the time of review, this document was in draft form (intended to be a supplement to the *standard* IEC880). It is assumed it will become a standard. However, for this report, it will be referred to as a publication.

The publication specifies some functions in response to timing, error detection and recovery, common cause failures (diversity):

- integrity checks (not required)
- rules on bypassing of safety functions by safety personnel
- redundancy
- diagnostics
- error detection and recovery
- diversity (not required), e.g.,
    - functional diversity in order to sense operational occurrences from different parameters
    - design or manufacturing diversity
    - equipment diversity

There is a full section on security analysis that specifies security objectives, and maps the security test plan to the full system lifecycle (see Section B.10.4 of this report).

Some functions are discussed for maintainability.

B.10.5. Software Engineering Practices

Publication recommends use of formal methods for the highest requirement of safety importance.

Publication also recommends use of prototyping, but cautions it should be used only to answer questions, not to build the system.

As a software engineering practice, the publication recommends use of CASE tools and states categories of tools, their purposes, and features. A later draft will discuss qualification of those tools. Guidelines for use of support tools are included.

Practices for language are reasonably thorough.

Suggestions are given for maintainability; in particular the customer will provide the maintainability specifications.

An entire section addresses computer security issues. The publication recommends a separate set of plans to address computer security. The draft standard for software safety plans [P1228] provides requirements for plans to address software safety needs; it is recommended that the concept be adapted for computer security.

B.10.6. Assurance Activities

B.10.6.1. Software Verification and Validation (SV&V)

Additional SV&V testing is to be decided. Testing is presently only addressed under maintenance.

B.10.6.2. Software Quality Assurance (SQA)

An SQAP is mentioned. The SQAP must account for the use of pre-existing software, and must address procedures for use of support software.

B.10.6.3. Software Configuration Management (SCM)

Mentioned only for maintenance.

B.10.6.4. Software Hazard Analysis

Under Section 3.1 (Architecture) the very first requirement is to identify all possible incorrect responses of the software (this is essentially one type of software hazard analysis though it is not referred to as such).

B.10.7. Project Planning and Management

Project management documentation is addressed under "Improvements to Existing Text of IEC 880," paragraph 3. There are no other references to project planning and management.

B.10.8. Procurement Concerns

If the computer system has critical functions for the safety of the nuclear power plant, design and implementation should be assessed by a team separate from development.

B.10.9. Presentation

It is apparent that this publication is being prepared by different authors and that it has not been thoroughly edited. For example, SCM is cited in the maintenance section but not during development. Inconsistencies such as this will probably be edited, so it is useless to comment about the presentation at this time. Some recommendations for the final version are

- o be consistent on use of "should" and "shall"
- o clarify relationship of this publication to IEC880

B.10.10. Supplemental Information

None.

B.10.11. General Comments

This publication is intended as a supplement to IEC880. Because it is only a draft, with many sections missing, this review is performed without full knowledge of the intentions of this working group. It is apparent that the publication is being prepared by different authors and that the final publication will be significantly different. The material in this publication appears to be in support of carrying out requirements of IEC880, rather than complementing them.

Response to the Principal Questions:

- It is premature to address the principal questions.

B.11. NPR-STD-6300: Management of Scientific, Engineering and Plant Software [NPR6300]

B.11.1. Levels of Criticality/Assurance

Software shall be classified according to the grading of the hardware, function, or activity it supports or performs. SV&V is performed consistent with the complexity and criticality of the software.

B.11.2. Lifecycle Phases

The lifecycle model of ASMENQA2 shall be used.

B.11.3. Documentation

Documentation developed or used shall be in accordance with ASMENQA2 and industry practices.

Required documentation includes: Software Requirements Documentation; Software Design and Implementation Documentation; and, User Documentation. Specifics on these and other documents can be found in ASMENQA2.

Storage and maintenance of documents and records shall be in accordance with [50] and the NPR Records Management Plan for the life of the New Production Reactors.

B.11.4. Required Software Functionality Against Hazards

Security control is required to restrict access for all software and computer systems.

B.11.5. Software Engineering Practices

Not addressed.

B.11.6. Assurance Activities

B.11.6.1. Software Verification and Validation (SV&V)

An SVVP and an SV&V report (SVVR) are required and their content specified. The definition of both software verification and software validation are different from those in IEEE1012. This standard's definitions actually create problems with the recommendation that software verification activities should be done before software validation. The demonstration that software correctly performs its stated capabilities and function (the definition for software verification in NPR6300) requires testing, which is software validation.

Software verification:

- shall be performed in parallel with the software development process, as much as possible
- design reviews and other design control measures shall be applied

Software validation:

- shall be separate from software verification as much as possible
- testing is the primary method; validation activities such as test plans and test cases are integrated into each step of the software development process

SV&V documentation includes information regarding the hardware and software configuration, and is organized such that traceability to software requirements and design is possible.

SV&V covers all phases of the lifecycle.

B.11.6.2. Software Quality Assurance (SQA)

ASMENQA1 and ASMENQA2 are considered a part of this standard.

An SQAP (in accordance with ASMENQA2) is required and its minimum requirements are listed.

B.11.6.3. Software Configuration Management (SCM)

The NPR0006 and NPR6301 are considered a part of this standard.

SCM establishes and controls interfaces, identifies and controls all software changes, and includes: configuration identification; configuration control, configuration status accounting; configuration verification, auditing, and records retention (as identified in NPR0006 and this standard).

B.11.6.4. Software Hazard Analysis

Not addressed.

B.11.7. Project Planning and Management

Not addressed.

B.11.8. Procurement Concerns

Whenever practical, SV&V shall be performed by individuals other than those who designed or developed the software or documentation.

B.11.9. Presentation

While the requirements may be general, the presentation is well-organized and statements are not ambiguous.

B.11.10. Supplemental Information

Requirements for the development of new software, existing software, and procured software and services are addressed.

NPR6300 has requirements for problem reporting and corrective action, which go beyond change control and support today's movement toward process improvement. For example, trend analysis on anomalies is required. NPR6300 requires security protection. It also defines a policy for data rights (includes software). Guidelines are provided for reuse of existing software.

B.11.11. General Comments

This standard addresses several issues that other documents in this study do not (see Section B.11.10 of this report) and provides detail on some assurance activities (e.g., SV&V, SCM).

Response to the First Set of Principal Questions:

- This standard has no requirements for software engineering practices, project management, software hazard analysis, and required software functionality against hazards.

Response to the Second Set of Principal Questions:

- The requirements are general, but at least an auditor can determine the basic activities that should be performed.

B.12. P1228: (DRAFT) Standard for Software Safety Plans (IEEE Working Group) [P1228]

B.12.1. Levels of Criticality/Assurance

Criticality analysis is discussed, and is suggested as one means of performing a software safety requirements analysis.

B.12.2. Lifecycle Phases

The SSP describes safety-related activities for each stage of the software lifecycle.

IEEE1074 specifies which processes must be included in the development and maintenance of critical software.

B.12.3. Documentation

Documentation required for all software is identified in IEEE Standards listed in the references, and additional requirements are included for safety-critical software.

The documentation requirements of this standard may be included through the augmentation of documents. Software safety requirements must be specified, possibly as part of the SRS. Other documentation that may be augmented includes: software project management plan; software development standards, practices, and conventions; test documentation; SVVP (describes how SV&V will be performed); SVVR (reports results of SV&V); and software user documentation. software design documentation is not mentioned, but the IEEE1016 for design description is included in the references.

Since this is a planning standard, the omission of source code may not be significant. However, the omission of reference to installation, operations, and maintenance manuals for ensuring that safety requirements are addressed may be significant.

Detailed requirements are listed for records of the Software Safety Program activities.

B.12.4. Required Software Functionality Against Hazards

Not addressed.

B.12.5. Software Engineering Practices

Specific software engineering practices have not been cited. However, engineering practices essential to achieving system and software safety objectives must be identified for a project.

This entire standard is devoted to specifying the definition of one quality attribute--software safety. Criteria are not specified.

Hazard analyses including software hazard analysis and software safety analysis are referred to as engineering practices. These analyses are addressed under Section 5.4.

B.12.6. Assurance Activities

B.12.6.1. Software Verification and Validation (SV&V)

SV&V is not directly required. Instead, documentation for SV&V is required and may augment the project's SVVP, with emphasis on safety features (see Section B.12.3 of this report). IEEE7301 is required which requires SV&V.

Testing is required indirectly through IEEE7301 which requires SV&V and additional testing. Testing is also indirectly required through requirements for test documentation (see Section B.12.3 of this report) and analysis of test reports.

B.12.6.2. Software Quality Assurance (SQA)

SQA is required, with specific attention to SQA activities for software safety. IEEE7301 provides SQA requirements, and IEEE983 provides guidance for SQA.

B.12.6.3. Software Configuration Management (SCM)

SCM is required during all phases of the software lifecycle. It includes control of project documentation, source code, object code, data, development tools and environments (the latter includes both software and hardware), and test suites. IEEE828 defines the SCM requirements. IEEE1042 may be used for guidance.

B.12.6.4. Software Hazard Analysis

The standard assumes that a preliminary hazard analysis has been done on the entire system, and that the information is available to those performing additional software safety analysis.

The following safety-related analyses are required: software safety requirements; software safety design; software safety code; software safety test; and, software safety change. The actual techniques suggested for performing these analyses are often considered verification techniques (e.g., specification analysis, logic analysis, data analysis, interface analysis, constraint analysis, module functional analysis).

B.12.7. Project Planning and Management

This standard requires a management section of the SSP, and gives a list of management activities that must be described in the plan.

B.12.8. Procurement Concerns

This standard specifies that the SSP must identify provisions for ensuring that subcontractor software meets software safety program requirements.

An SSP written by this standard must identify a method for certifying that the software product was produced in accordance with the process specified in the SSP.

A single individual (as in MOD0055) is designated as responsible for the overall conduct of the Software Safety Program. Organizational independence is required.

The SSP must specify qualifications for the personnel performing the software safety activities specified in the SSP.

B.12.9. Presentation

This standard is a plan standard and uses the word "shall" to specify elements or activities an SSP must address. Intermingled with the "shall"s are recommendations on processes that may be used to fulfill required activities. There should not be confusion between requirements and considerations. However, reference to the IEEE family of software engineering standards does hide some implicit requirements and relationships. For example, IEEE7301 is the vehicle to indicate that SV&V will be used to fulfill some software safety activities. A user of P1228 will need to be well-versed in this IEEE family of standards.

Acronym Problems:

- ASHA is not in the acronym list.
- SCM refers to safety-critical modules but is *very* confusing since SCM often refers to software configuration management.

B.12.10. Supplemental Information

This standard contains detailed steps for the use of previously developed or purchased software.

B.12.11. General Comments

This standard is not consistent with its scope relative to system applicability. Sometimes it has references to hardware, or system, and other times it is restricted to software. Is this a software safety plan or a system safety plan? Some clarification is needed.

Response to the First Set of Principal Questions:

- While the standard does not specify any software functionality against hazards, it does require software safety-related analyses. The results of these analyses are to be used to verify that the software

requirements, design and code have functionality to protect against the hazards. The document assumes hazards are application specific.

- The intent of this standard is to focus attention on software safety issues during development. This standard serves this purpose well. It must be used with other standards for assurance. This approach to ensure that software safety issues are particularly addressed and monitored in project planning and assurance activities could be used for a basic standard in computer security planning as well. However, specific analyses and techniques for assuring software safety are presented under "Discussion" sections and are not requirements of this standard.

Response to Second Set of Principal Questions:

- An auditor could verify that the generic planning and documentation requirements for software safety issues have been performed. Because the safety techniques and analyses are included in the Discussion sections and are not requirements, auditors do not need to examine the technical activities of the project to show compliance with requirements of this standard. An auditor could verify that all requirements for software safety issues have been performed.

B.13. RTCA/DO-178A: Software Considerations in Airborne Systems and Equipment Consideration [RTCA178A]

B.13.1. Levels of Criticality/Assurance

Levels of criticality must be addressed in a "certification" plan. A minimal description with guidance on three levels is provided. The criticality level relates to the system function; the software level is related to the effort required to achieve certification of functions, systems, and equipment.

Requirements for assurance are summarized for functional criticality levels; assurance means the evidence a vendor must provide to demonstrate compliance with requisite development and software verification activities.

B.13.2. Lifecycle Phases

Software development, verification, assurance. In this guideline assurance is the set of activities upon product completion that comprise the evidence indicating that the product should be accepted.

B.13.3. Documentation

Documentation is under SCM control to support initial certification and post-modification re-certification.

Basic elements of the following documents are specified: software requirements document; design description document; programmer's manual; source listing; source code; executable object code; support/development system configuration; accomplishment summary; software design standards; system requirements; and, plan for software aspects of certification.

B.13.4. Required Software Functionality Against Hazards

Not addressed.

B.13.5. Software Engineering Practices

Constraining functions to clearly identifiable modules is suggested as one approach to design. No other software engineering practices are mentioned.

B.13.6. Assurance Activities

In this guideline, the "assurance" paragraphs identify the compliance activities for the tasks of the standard. These generally consist of documentation of the tasks and audits to assess completion of tasks.

B.13.6.1. Software Verification and Validation (SV&V)

Basic elements of the software verification plan, procedures and results document are specified.

Specified in conjunction with development. Matrix of activities to software levels is provided. Note that software requirements verification includes verifying adequacy and completeness of the software requirements and the system requirements.

Software verification of design includes generation and review of test criteria for the design. Software verification includes checking execution flow and timing; data flow/data corruption; and, hardware action affecting software functional partitioning and integrity.

Validation treated as testing for compliance with system requirements under operational conditions (e.g., simulation, actual environment).

Complete test documentation specified. Testing described for requirements testing, structure testing, module testing, module integration, hardware/software integration testing. Detailed objectives for test cases are identified. All test documentation goes under configuration control.

B.13.6.2. Software Quality Assurance (SQA)

Basic elements of the SQAP are specified. Effort may vary according to function criticality and equipment application. The SQAP must show how SQA and SCM will interface. SQA audits are performed to verify that procedures are adhered to. Software reviews are conducted at specified milestones. Software problem-reporting and corrective action procedures are necessary at levels 1 and 2. SQA identifies the documents subject to control.

B.13.6.3. Software Configuration Management (SCM)

Basic elements of the SCMP are specified. Effort may vary according to function criticality and equipment application. The SCMP must be coordinated with the SQAP. Detailed guidance is provided for identification, management of change, status accounting, media control, and configuration audits. Guidance is given for continued responsibility of SCM.

B.13.6.4. Software Hazard Analysis

System Criticality and Software Levels:

> Software levels (1, 2, and 3) are based on the system criticality categories. The software level varies with the criticality category and may vary within the criticality category (design/implementation techniques/considerations may make it possible to use a software level lower than the functional categorization). Otherwise, Level 1 is associated with the Critical category (the continued safe flight and landing of the aircraft would be prevented), Level 2 with Essential (the capability of the aircraft or the ability of the crew to cope with adverse operating conditions would be reduced), and Level 3 with Non-Essential (the aircraft capability or crew ability would not be significantly degraded).

B.13.7. Project Planning and Management

Not addressed.

B.13.8. Procurement Concerns

In this guideline, assurance relates directly to certification of the system by the customer. For each step of development and verification, the guideline identifies the records and activities that must be provided as evidence (e.g., results of verification activities, error tracking reports, audits).

B.13.9. Presentation

The presentation contains clear explanations concerning the use of this guideline. Recommendations for development and verification are clear and reasonable.

B.13.10. Supplemental Information

This is a guideline, under revision.

B.13.11. General Comments

This guideline is currently being revised. In the draft Version 6, the approach of using the word "process" rather than "phase" is prevalent, and appears to be based heavily on the terminology of IEEE1074. Some material is added on Project Planning and Management. More description is added for determining levels for software (now up to 5 levels) and additional advice on testing is provided. However, guidance appears to be lumped into either high-level or low-level.

Response to the First Set of Principal Questions:

- This guideline addresses at least minimally all the major categories identified in Appendix A except for required software functionality against hazards and project planning and management.

Response to the Second Set of Principal Questions:

- RTCA178A is intended as a guidance document, not as a standard for which compliance should be demonstrated.

- The developer is not obligated to perform all the activities identified in the guideline. In fact, the developer selects the activities that will be performed. While this guideline contains strong recommendations, the fact that the developer chooses the activities weakens the assurance results of using RTCA178A.

B.14. (DRAFT) Standard for Software Engineering of Safety Critical Software [SOFTENG]

B.14.1. Levels of Criticality/Assurance

This standard is for safety-critical software, the most stringent class of requirements. Other levels are not addressed.

B.14.2. Lifecycle Phases

This standard identifies software phases and processes relative to system phases.

The development process involves software requirements definition, software design, and code. The verification process involves requirements, design, code and testing. Outputs from the development process are verified as they are completed. O&M is not covered.

B.14.3. Documentation

This standard imposes requirements on the Design Input Documentation (DID), that is, the information the software engineering process must have from system engineering, hardware, and pre-developed software.

The SRS specifies requirements for each of the following attributes: completeness, correctness, consistency, verifiability, modifiability, traceability, understandability, and robustness.

The software design description (SDD) specifies requirements for each of the following attributes: completeness, correctness, predictability, robustness, consistency, structuredness, verifiability, modifiability, traceability, modularity, and understandability. Some of these may be practices and features of the design that should be described in the SDD, but as practices they should be required in Section 3 (see Section B.14.4 of this report).

The code documentation specifies requirements for each of the following attributes: completeness, correctness, predictability, robustness, consistency, structuredness, verifiability, modifiability, traceability, and understandability.

B.14.4. Required Software Functionality Against Hazards

Not addressed.

B.14.5. Software Engineering Practices

This standard states the quality objectives and specific definitions of attributes related to the objectives. Engineering principles (practices) that will enable achievement of those objectives must be selected.

Quality attributes are stated and defined; outputs of software engineering processes are described to meet these attributes.

The standard contains a list of fundamental principles including use of mathematical notation, information hiding, principles for test strategy, and activities over the full lifecycle. CM, hazard analysis, and verification are stated as principles (for the system and as applied to software).

Isolation of critical components is required in the SRS.

Software engineering practices and controlled practices are specified in the documentation requirements for design and code. For example, it is the design that must have the attribute "predictability through practices" such as minimizing the use of interrupts. This is not the proper location for these practices. These are principles and practices that should be identified in the body of the standard. Other similarly misplaced attributes are robustness, structuredness, modularity, and understandability.

B.14.6. Assurance Activities

B.14.6.1. Software Verification and Validation (SV&V)

The verification process includes requirements verification, design verification, code verification and testing, and shall be accomplished in accordance with the SDP and the standards and procedures handbook. Reviews and reports of reviews (format and content) which are specified are software requirements, software design, systematic design verification, code, systematic code verification, and code hazards analysis.

The relationship between the SRS and the DID is specified. Checking to ensure that the SRS meets its required attributes based on the DID is not required; this important verification activity could be omitted.

Objectives are specified for unit, subsystem, validation, and reliability testing. Each section specifies types of test cases and documentation descriptions. Test procedures (what the tests should cover) are defined in the appendix.

Test reports are required and content is specified.

B.14.6.2. Software Quality Assurance (SQA)

"The quality assurance program requirements not addressed in this standard are covered in the AECL CANDU and Ontario Hydro quality programs" (Foreword). SQA is NOT specified as a component of software engineering but typical SQA processes are included under verification.

B.14.6.3. Software Configuration Management (SCM)

SCM is specified as a component of software engineering.

SCM is addressed through the requirements of the SDP.

SCM covers baselines, change control library, and error analysis.

B.14.6.4. Software Hazard Analysis

System hazard analysis is required indirectly under rules for the DID that require identification of failure modes for the system.

Software hazard analysis is required for code, but under the verification process.

Section 3.6 lists types of potential hazards that should be identified in a hazard analysis report.

B.14.7. Project Planning and Management

SDP includes planning, milestone, and organization information.

Guidelines are specified for project management in the SDP description.

B.14.8. Procurement Concerns

All requirements of standard must be met (no tailoring).

Independence of developer and verification personnel specified.

Training must be planned and upgraded for the project.

B.14.9. Presentation

The presentation is a little confusing. For example, Section 3.1, Appendix B and Appendix D all need to be reviewed in order to identify the requirements for the SRS. The body of the standard is descriptive, and the requirements are contained in the appendices (reverse order of many standards). Requirements for documentation sometimes contain the software engineering practices that are to be embodied by the software, or the features that should be verified (e.g., code hazard analysis report lists potential hazards). The term "Design Input Documentation (DID)" should have been defined at its first reference. Although this standard is intended primarily for use in Canada, where DID is a familiar term, non-Canadians will also use it.

B.14.10. Supplemental Information

This is a draft standard prepared by Ontario Hydro and AECL CANDU. Comments were due in December, 1991. This standard will be the first of a series of standards where the requirements of each standard are intended for a specific category, per CATEGORY.

This standard does not specify other standards. Instead it specifies that standards for specific processes or outputs shall be written by the project according to this standard.

Changes to the software, any time during its life, must be verified to the same degree of rigor as the original development.

B.14.11. General Comments

This draft standard has identified quantifiable features for quality attributes. Few, if any, standards have done so.

Response to the First Set of Principal Questions:

- This draft standard does not require any specific software functionality for addressing any specific safety requirements. It requires information from other system components to be made available to software engineering. The fundamental principles are excellent. The weakness of the document may be that, except for the code hazards analysis, it does not emphasize that assurance activities should focus especially on the software features that impact safety. However, the document is written for safety-critical software and therefore all the requirements of the standard apply to all of the software.

Response to the Second Set of Principal Questions:

- The organization of the standard does not fit the template. For example, engineering requirements were actually hidden in documentation requirements. A mapping of engineering requirements that affect software requirements, design and code would be useful. Some other engineering requirements appear in the verification requirements as well. With a map, however, an auditor could check that these features are built into the system.