NUREG/CR–6263
MTR 94W0000114
Vol. 1

High Integrity Software for Nuclear Power Plants

Candidate Guidelines, Technical Basis and Research Needs

Executive Summary

Manuscript Completed: June 1995
Date Published: June 1995

Prepared by
S. Seth, W. Bail, D. Cleaves, H. Cohen,
D. Hybertson, C. Schaefer, G. Stark, A. Ta,
B. Ulery

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

Prepared for
Division of Systems Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555–0001
NRC Job Code L2610

DISCLAIMER

ABSTRACT

The work documented in this report was performed in support of the U.S. Nuclear Regulatory Commission to examine the technical basis for candidate guidelines that could be considered in reviewing and evaluating high integrity computer software used in the safety systems of nuclear power plants. The framework for the work consisted of the following software development and assurance activities: requirements specification; design; coding; verification and validation, including static analysis and dynamic testing; safety analysis; operation and maintenance; configuration management; quality assurance; and planning and management. Each activity (framework element) was subdivided into technical areas (framework subelements). The report describes the development of approximately 200 candidate guidelines that span the entire range of software life-cycle activities; the assessment of the technical basis for those candidate guidelines; and the identification, categorization and prioritization of research needs for improving the technical basis. The report has two volumes: Volume 1, Executive Summary, includes an overview of the framework and of each framework element, the complete set of candidate guidelines, the results of the assessment of the technical basis for each candidate guideline, and a discussion of research needs that support the regulatory function; Volume 2 is the main report.

CONTENTS

SECTION		PAGE
Abstract		iii
NRC Summary		vii
Acknowledgments		ix
1	Introduction	ES-1
2	System Context and Framework	ES-2
3	Development of Candidate Guidelines	ES-6
4	Framework Elements and Candidate Guidelines	ES-9
5	Assessment of Technical Basis	ES-16
6	Identification, Categorization and Prioritization of Research Needs	ES-19
7	Discussion of Research Needed	ES-21
8	High-Priority Research Needs Supporting the Regulatory Function	ES-23
References		RE-1

FIGURES

FIGURE		PAGE
ES-1	Framework for System Development	ES-3
ES-2	Framework for Software Development and Assurance	ES-5
ES-3	Framework Elements and Subelements for Developing Candidate Guidelines	ES-7
ES-4	Development of Candidate Guidelines (Example: Software Design)	ES-8

TABLES

TABLE		PAGE
ES-1	Candidate Guidelines and Assessment of Technical Basis	ES-29
ES-2	Research Needs Supporting the Regulatory Function	ES-57
ES-3	Ranking of High-Priority Research Needs Supporting the Regulatory Function	ES-67

NRC SUMMARY

The objective for this project was to provide assistance to the NRC for the development of a technical basis for regulatory positions related to the use of high-integrity software in nuclear power plants. The assistance included the identification of research issues that could enhance a technical basis.

This report contains a comprehensive discussion of the present state of software engineering processes and design attributes in the form of candidate guidelines for the elements of the software life cycle and assurance activities. The candidate guidelines are considered by the contractor to be good practices that are important in the development of high integrity software for nuclear power plants. Most of the design attributes can be found in current software industry standards.

It is emphasized here that the application of the candidate guidelines to regulation and the determination of the need for the research identified in this report require further assessments by the NRC. The assessments will include consideration of the contribution to safety, the degree to which the candidate guidelines have an acceptable technical basis, and the cost-effectiveness of each guideline and research issue.

The NRC's current regulatory position on issues associated with the application of digital computer technology to instrumentation and control (I&C) systems that are important to safety and the NRC's staff actions to resolve these issues are presented in (1) SECY-91-292, *Digital Computer Systems for Advanced Light Water Reactors*; (2) SECY-93-087, *Policy, Technical and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor (ALWR) Designs*, Section II Q, "Defense Against Common-Mode Failures in Digital Instrumentation and Control Systems," April 1993, as clarified by the Staff Requirements Memorandum (SRM) on SECY-93-087, July 1993; (3) the draft *Operating Reactors Digital Retrofits Digital System Review Procedure* and the draft Branch Technical Position, *Digital Instrumentation and Control Systems in Advanced Plants* (presented at the Digital Systems Reliability and Safety Workshop, sponsored by the NRC and the National Institute of Standards and Technology, 13–14 September 1993); and (4) Generic Letter 95-02, *Use of NUMARC/EPRI Report TR-102348, 'Guideline on Licensing Digital Upgrades,' in Determining the Acceptability of Performing Analog-to-Digital Replacements Under 10 CFR 50.59*, April 1995. These documents present the current NRC position on basic safety issues, together with the NRC policy statements with regard to the resolution of these issues. It should be noted that a number of Final Safety Evaluation Reports (FSERs) have been issued approving retrofit and advanced applications of digital I&C safety systems.

ACKNOWLEDGMENTS

The work presented in this report involved the analysis, integration, and synthesis of a vast amount of technical information on a broad spectrum of topics related to high integrity software. The guidance provided by NRC staff during the course of this work was invaluable. The authors especially acknowledge the efforts of the following NRC staff members: Robert Brill (NRC Project Officer), Leo Beltracchi, Frank Coffman, John Gallagher, Joe Joyce, Joel Kramer, Jim Stewart, and Jerry Wermiel.

The report also benefited significantly from the peer review provided by a panel consisting of the following experts: Mel Barnes, Taz Daughtrey, Michael DeWalt, Ray Ets, Roger Fujii, Carl Gilbert, Richard Hamlet, Herbert Hecht, Gordon Hughes, Paul Joannou, Ted Keller, Kathryn Kemp, Schlomo Koch, John Matras, John McDermid, Joseph Naser, Ronald Reeves, and Dolores Wallace. Their affiliations are provided in Volume 2, Appendix B. The authors express sincere gratitude to the panel members for the perspectives and the comments they offered. Comments provided by William Agresti, Charles Howell, and Gary Vecellio, who contributed to MITRE's internal peer review, are also gratefully acknowledged. Many MITRE personnel contributed to the success of the Experts' Peer Review Meeting on High Integrity Software for Nuclear Plants, which was held at MITRE, 24–26 May 1994. Special thanks go to William Agresti, who served as the moderator for the meeting, and Brenda Fox, who coordinated all the meeting arrangements.

The preparation of this report, from the working drafts of individual sections to the final integrated product, required various types of technical support activities, which included developing and maintaining computer databases, coordinating the technical information needs of the project team members, and editing. The authors feel especially indebted for the painstaking and conscientious efforts of Tammy Ryan, Brenda Fox, and Sheila McHale, who bore the brunt of this work. Thanks are also due to Susan Gerrard and Dee Krzebetkowski for their support of the project. Finally, Rona Brière of Brière Associates deserves praise and many thanks for her sound advice and valuable suggestions during the final editing of this document.

EXECUTIVE SUMMARY

1. Introduction

Instrumentation and control (I&C) systems based on the use of digital computer technology offer several advantages, such as higher availability due to automatic self-test and diagnostics, greater flexibility, and increased data availability. While digital systems have replaced selected analog systems at a number of existing nuclear power plants, future reactor designs are expected to make extensive use of advanced applications of digital computer technology. To ensure that safety is not compromised, a proper implementation of this technology in nuclear power plants must address several potentially safety-significant concerns, including potential common-mode failures due to the use of common software in redundant channels [USNRC GL 95-02]. The NRC's current regulatory positions on issues associated with the application of digital computer technology to I&C systems that are important to safety, and the NRC staff actions taken to resolve these issues, are presented in documents referenced in the "NRC Summary" at the beginning of this volume.

The NRC has also undertaken several activities towards developing additional regulatory guidance for reviewing and approving digital computer systems to be used in the safety systems of nuclear power plants. The work documented in this report is an important part of that overall initiative. Specifically, it is in support of the NRC's efforts to examine the technical basis for guidelines that could be considered in reviewing and evaluating the safety system software. Based on that examination, candidate guidelines with an adequate technical basis could be considered by the NRC in developing regulatory guidance, such as Regulatory Guides or revisions to the Standard Review Plan. Candidate guidelines with an inadequate technical basis could be considered by the NRC in developing research projects.

The NRC's draft Branch Technical Position for advanced plants [USNRC-BTP DRAFT], which requires an orderly and systematic process for the development of software through its various life-cycle stages, provided a starting point for this work. The approach taken consisted of the following steps:

1. Develop candidate guidelines applicable to each software life-cycle stage.

2. Examine and describe the technical basis for each candidate guideline, where it exists.

3. Identify research needs where the technical basis for a given candidate guideline is insufficient or lacking.

Several previous studies have indicated considerable variability in the scope and depth of guidance available in the existing standards on developing high integrity software [EPRI, 1992; IAEA, 1993; NIST NUREG/CR-5930]. The approach outlined above provided a methodology for identifying those candidate guidelines whose use would involve considerable subjectivity because of the lack of sufficiently detailed methods, measures, or criteria; and for suggesting research that could minimize the uncertainty and vagueness in the available guidance for implementing the candidate guidelines considered.

A panel of outside experts with considerable knowledge of software safety and reliability relative to different government and industry programs provided a peer review of this report in its preliminary draft stage. A meeting of the expert panel was held as part of this process to discuss the peer review comments. Discussions at the meeting were aimed at obtaining the best judgment of, rather than consensus among, the experts. Written comments provided prior to the meeting, as well as meeting discussions, were used in revising the report.

It is emphasized here that the application of the candidate guidelines to the regulatory process and the determination of the need for and scope of research identified in this report require further assessment by the NRC. This assessment will include consideration of the contribution to safety and the cost-effectiveness of each guideline, and equally important, its potential impact on the incorporation of advances in digital computer technology. The regulatory process is not intended to be a deterrent to the application of technology advances, provided that the safety of the final product is maintained or enhanced.

The remainder of this volume summarizes the major aspects of the work: the framework for and development of the candidate guidelines (Sections 2 and 3); overview of the framework elements (Section 4); the assessment of the technical basis (Section 5); and the discussion of research needs (Sections 6, 7 and 8). Detailed descriptions of the technical basis for the candidate guidelines are provided in the main report (Volume 2).

Table ES-1 lists all the candidate guidelines along with the results of assessment of the technical basis for each guideline and the research needs identified to address any gaps in its technical basis. The importance of research needs that directly support the regulatory function is summarized in Table ES-2. For convenience, the tables are located at the end of this volume.

2. System Context and Framework

System-Software Interface

Since software is an integral part of I&C systems based on digital computer technology, requirements for the software must be allocated by and derived from the functional and safety requirements of the system. The system framework depicted in Figure ES-1, which is adapted from [Fujii, 1993] and [NIST, 1993], shows that a proper system design identifies software components, hardware components (including computer hardware), and human operators, and allocates requirements and constraints to each component. Examples of the types of requirements allocated to the various components of an I&C system that should be considered in an integrated manner are (1) fault detection, diagnostic, and recovery capabilities; (2) responses to design basis events, including computer-unique failure modes; (3) operator-machine interface requirements; (4) timing, response time, throughput, and performance requirements; and (5) functional diversity or defense-in-depth, as required.

Several significant studies on the sources, nature, and distribution of software defects underscore the importance of specifying a complete, clear, and correct set of requirements for the software.

Figure ES-1. Framework for System Development

For example, [Basili and Perricone, 1984], [Davis, 1990], and [Jones, 1991] provide evidence that approximately half of software defects can be traced to errors made during the requirements stage. Errors in requirements specifications are also difficult to detect in subsequent testing and are labor-intensive to correct [Boehm, 1975; Grady, 1992; OECD HALDEN, 1993]. Therefore, preventing the insertion of errors in the early requirements specification stages of software development not only minimizes risk, but also provides a cost-effective approach for developing high integrity software.

For advanced applications of digital I&C systems in nuclear power plants, detailed system-level standards are not available that would encompass the system-software interface, and provide guidance on issues related to system decomposition and interrelationships among the system components—hardware, software, and human operators. This is a vital need relative to the necessary systems analysis that should be performed for such applications. However, it was explicitly assumed in developing the candidate guidelines and associated technical basis, as documented in this report, that the systems analysis has been properly conducted, and the system-software interface completely defined.

Software Development and Assurance

The software development component of the overall system development, which is depicted separately in Figure ES-2, provided the framework for developing the candidate guidelines. It incorporates the main ideas from [USNRC-BTP DRAFT], [LLNL NUREG/CR-6101], and [NIST, 1993], which in turn are based on various standards, such as [IEEE1012] and [ASME-NQA-2a].

Software life-cycle activities, which constitute the elements of the framework, are depicted in Figure ES-2 as relating to either development or assurance. The illustration of software development activities in the figure does not imply an endorsement of any particular model. In particular, the timing of the activities shown may vary and overlap. The software assurance activities are closely coupled with the development activities and are performed throughout the life cycle. They provide oversight of, and support to, software development so that the generated products meet the safety requirements, are of the necessary quality, and are within cost and schedule. Also shown in Figure ES-2 are NRC audit activities that serve to verify that the software development process and the associated documentation are adequate.

The development and organization of the candidate guidelines as documented herein maps directly to the elements of the framework of Figure ES-2. However, depending on the technical content of a given activity element, and in order to avoid overlapping discussion, some elements have been combined or regrouped. For example, the candidate guidelines for the four successive levels of software testing and for verification and validation (V&V), all of which are depicted in the framework as separate life-cycle activities, are presented herein within the following three elements: V&V—Static, V&V—Dynamic (Testing), and V&V—General Considerations. This regrouping allows focus on the two separate technical aspects of V&V—static analyses and dynamic testing—that are important to both software development and software assurance. Similarly, software planning and software project management are combined as the element Software Planning and Management. The planning for each life-cycle activity is also discussed in the individual element corresponding to that activity.

Figure ES-2. Framework for Software Development and Assurance

Figure ES-3 shows the organization of the candidate guidelines. Further details on the scope of each framework element and of the subelements defined within each element are provided in Section 4 below.

3. Development of Candidate Guidelines

For each software development or assurance activity (framework element), candidate guidelines were grouped based on their subject matter into technical areas (framework subelements). For example, one of the subelements of the element Software Design is Interface Integrity. Certain recent standards that cover a broad range of software development activities and that are generally recognized in the nuclear industry were the primary sources for deriving candidate guidelines within each framework subelement. These sources, termed the "baseline" in this report, were the following: *Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations* [IEEE7-4.3.2]; the *Advanced Light Water Reactor Utility Requirements Document* [URD]; and standards such as [IEC880], [ASME-NQA-2a: Part 2.7], and various Institute of Electrical and Electronics Engineers (IEEE) standards explicitly referenced within requirements or guidelines contained in [IEEE7-4.3.2] and the [URD]. The NRC's draft regulatory guidance documents [USNRC-BTP DRAFT] and [USNRC-RETROFITS DRAFT] are included in the baseline as well.

Pertinent statements that describe software attributes relevant to safety were extracted from the baseline to develop the candidate guidelines. Guidance in the standards relative to specific methods or approaches (e.g., object-oriented versus structured analysis in the element on Software Design) or which had little relationship to safety was not extracted. Although statements from the baseline sources range from requirements to nonbinding guidance, they were all considered as components for developing the set of candidate guidelines. For uniformity, all candidate guidelines use "should" in stating the desired software attributes, although "shall" may have been used in the corresponding statement in the baseline source.

Because standards have different objectives, applications, and vintage, it is common to observe overlap, mismatch, or gaps in the scope, nature, completeness, and specificity of guidance on a given topic. Therefore, for each framework subelement, based on a consideration of the essential attributes of the guidance contained in the baseline, any gaps for which additional guidelines might be considered were identified. The suggested additional guidelines to fill the observed gaps were derived from defense, aviation, aerospace, and nuclear industry standards, including some developed in other countries, which are not encompassed by the baseline, e.g., [DOD-STD-2167A], [RTCA DO-178B], and [MOD55]; from pertinent results of work sponsored by the NRC; and from the researchers' software engineering experience with other government programs.

The process described above, which is also depicted in Figure ES-4, was used to develop a set of candidate guidelines for each framework subelement that summarizes the software safety

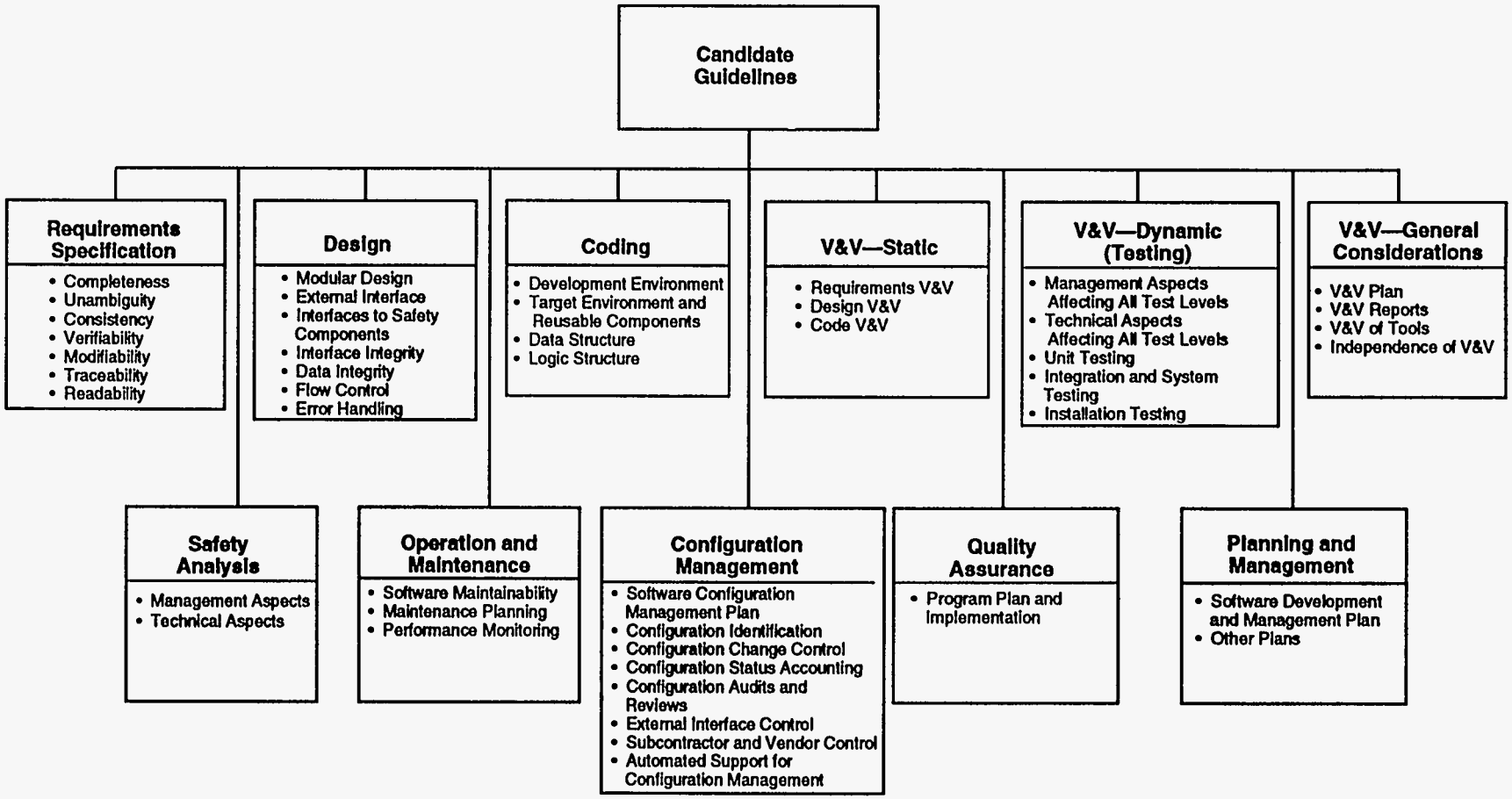

Figure ES-3. Framework Elements and Subelements for Developing Candidate Guidelines

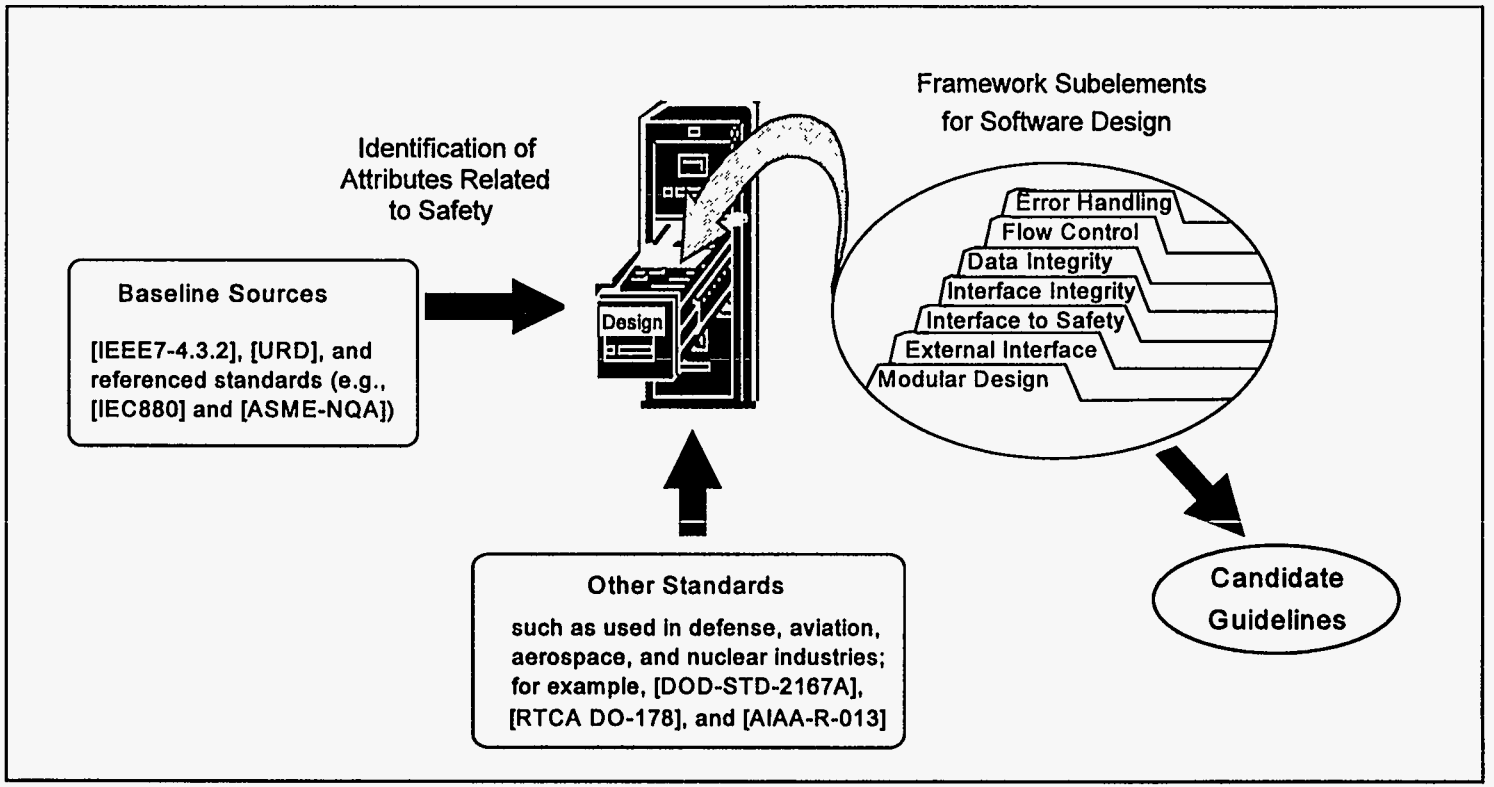

Figure ES-4. Development of Candidate Guidelines (Example: Software Design)

attributes derived from the baseline and any suggested additional guidelines. The candidate guidelines should not be considered as substitutes for the standards from which they were derived. The source standards provide additional details related to the attributes selected for high integrity software.

4. Framework Elements and Candidate Guidelines

The following is a brief overview of each element of the framework for software development and assurance. It explains the safety significance of each element, and indicates the scope of the subelements into which it is subdivided and for which candidate guidelines were developed. The candidate guidelines for all the framework subelements are presented in Table ES-1 at the end of this volume.

Software Requirements Specification

The software requirements specification (SRS) is the highest-level software specification of a software component of a system. Its scope is defined by the collection of system requirements that are allocated by the system design to that software component. Any software requirement that is a prerequisite to the system's meeting a system safety requirement is a software safety requirement. The SRS therefore has a vital role in the overall argument that the software is "safe," because a precise specification of the software requirements is necessary to provide assurance that the software will serve its intended purpose.

The candidate guidelines relating to this framework element were grouped into subelements corresponding to seven attributes or properties of the information that should be identified in the SRS. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Completeness**—Ensure that all necessary requirements are included.
- **Unambiguity**—Ensure that requirements are interpreted the same way by all readers.
- **Consistency**—Ensure that requirements do not conflict with each other.
- **Verifiability**—Determine that a practical method exists to verify that each requirement is satisfied.
- **Modifiability**—Ensure that requirements are easy to modify correctly.
- **Traceability**—Determine that software requirements trace to the system requirements/design, and the software design can be traced to the software requirements.
- **Readability**—Ensure that readers can easily read and understand all requirements.

Software Design

The design of a software component is the foundation for implementing the requirements and the constraints specified in the SRS. Those areas which could compromise the integrity and the robustness of the software design were identified as subelements for this framework element. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Modular Design**—Enhance the quality of software by dividing it into manageable, more understandable sets of interrelated components with clearly defined interfaces. Modular design contributes to safety by minimizing the ripple effect of design changes, and the propagation of abnormal conditions.

- **External Interface**—Ensure that specified external events are addressed up front so that other processing, specifically safety processing, will continue without interruption.

- **Interfaces to Safety Components**—Focus on separating safety and nonsafety software components to ensure that processing of the latter does not interfere with that of the former under any circumstances (e.g., by ensuring that only the necessary information is communicated through defined parameters and by performing periodic self-tests to confirm that safety software in protected memory is not corrupted).

- **Interface Integrity**—Ensure the correctness of the interfaces between software components so that errors, such as invalid protocol and data, do not occur.

- **Data Integrity**—Ensure that the software operates in defined states by minimizing the occurrence of errors that could transition the software to an undefined state or allow an unintended function to be performed.

- **Flow Control**—Ensure the correct and continual processing of safety components.

- **Error Handling**—Ensure that the software is robust and able to recover from an error by following a well-defined strategy.

Software Coding

Software coding is implementation of the SRS in accordance with the design for a software component. Subelements for this framework element were defined by focusing on both external and internal aspects of developing software that could affect the correctness of the source code, and thereby its predictable execution. The external aspects consist of the development environment, and the target environment and reusable components; the internal aspects that affect the source code are data structure and logic structure. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Development Environment**—Provide for the analysis of characteristics of the software development environment that might introduce errors resulting in a safety hazard. The environment typically involves the use of automated support tools to reduce the need for human effort and the associated risk of introducing human errors.

- **Target Environment and Reusable Components**—Ensure that the interface of a new software component with other existing software environments (e.g., operating system reusable software components) is sufficiently defined that software integrity is not compromised by behavior undocumented in the interface information.

- **Data Structure**—Address the representation of data using the available constructs to ensure high-quality implementation of the overall design.

- **Logic Structure**—Provide for a structured approach to coding the logic to ensure its correctness.

Software V&V—Static

Static software V&V is a subset of V&V that covers static methods in two broad categories: reviews or inspections and analytic techniques. Three subelements are defined for this framework element: requirements, design, and code V&V. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Requirements V&V**—Demonstrate that the specified software requirements and software safety constraints satisfy the requirements and safety constraints allocated to the software in the system design. The candidate guidelines for requirements V&V have a basis in the candidate guidelines for SRS, with the quality attributes most central to the goal of requirements V&V being completeness, unambiguity, and consistency.

- **Design V&V**—Check that a design is complete: that all levels of the design between the SRS and the source code satisfy the software requirements and, in particular, the software safety constraints given in the SRS. Successful design V&V thus increases confidence that the source code will correctly implement the software requirements, assuming that the source code correctly implements the design.

- **Code V&V**—Demonstrate that the source code correctly implements the software design, and the object code is a correct translation of the source code. These represent the last two links in the chain of argument from the SRS to the object code to increase confidence that software will fulfill its safety function, will not introduce hazards, and will otherwise behave as specified.

Software V&V—Dynamic (Testing)

Dynamic software testing is a subset of both software development and V&V. It involves executing the software to determine whether actual results match expected results, or to estimate the reliability of the software. Testing has multiple goals, including (1) exposing failures unique to execution (e.g., timing, iteration of logic and data, introduction of unknown paths caused by program interrupts); (2) confirming the understanding of the software achieved through static analysis techniques; and (3) demonstrating compliance with functional, performance, and interface requirements.

In making an argument that a software component is safe, it is important to demonstrate that the system behaves as expected under both normal and abnormal conditions. Testing, therefore, has an important role in the overall argument that software is safe. The strategy adopted in choosing test cases and executing tests has an effect on the types of failures that are observed and the faults that are discovered and corrected in the software. Thus, it is important that the testing strategy provide evidence of the software being "stressed," using a thorough mix of testing techniques during the test process.

Dynamic testing consists of several levels, starting with unit or component testing, and ending with installation tests conducted in the field. The subelements for this framework element include two on the general management and technical aspects of testing, and three on the various levels of testing. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Management Aspects Affecting All Test Levels**—Ensure that test planning and management take into account the tradeoffs among resources, staffing, and schedule, and provide adequate confidence in the software.

- **Technical Aspects Affecting All Test Levels**—Provide for an evaluation of test plans and specifications to demonstrate compliance with the SRS and the software design.

- **Unit Testing**—Ensure that individual units behave as specified in the design document, that all of the module interfaces behave as expected, and that the extreme cases of internal control and data structures do not result in unintended behavior.

- **Integration and System Testing**—Determine how units interact as larger composite items. Once all of the major design items have been integrated with each other and with any hardware subsystems, the purpose is to detect inconsistencies in the hardware-software interaction by exercising and testing the complete system against the SRS, using pre-defined success criteria.

- **Installation Testing**—Ensure that the software system can be installed, properly configured, and executed in the customer's execution environment. The testing includes not only the integrated system, but also the user documentation that will accompany it.

Software V&V—General Considerations

Software V&V activities take place throughout the software life cycle, and provide confidence that the safety system and development requirements have been implemented and can handle abnormal conditions. Software V&V ensures that the software adequately and correctly performs all intended functions, and that the products of any given cycle meet the requirements of the previous cycle(s). V&V ranges in rigor from informal reviews to formal proofs, and is an important part of the safety assurance argument for high integrity software.

The static and dynamic testing portions of V&V have already been addressed explicitly. The subelements for this framework element deal with general V&V considerations. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **V&V Plan**—Organize the V&V activities, provide visibility of these activities to the customer and to management, and provide direction to V&V personnel. The V&V plan plays an important role in the success of a V&V program by establishing control of the V&V activities.

- **V&V Reports**—Document the evidence to be presented to an auditor. Confidence in the safety of a software system depends on how thoroughly the V&V activities have demonstrated that the source code, software design, and software requirements have satisfied the system safety requirements allocated to the software. The safety case for software is the set of reasoned arguments that the software correctly implements all safety requirements and constraints allocated to the software in the system design.

- **V&V of Tools**—Provide assurance that support tools, such as those used in code generation, do not cause a defect in the operational system, and other tools (e.g., data flow analysis tool) satisfy their intended purpose of catching defects in the operational code.

- **Independence of V&V**—Provide for the technical, managerial, and financial independence required for an effective V&V effort.

Software Safety Analysis

Software safety analysis demonstrates that the system is safe when the software operates both as intended and in the presence of abnormal conditions and events (ACEs). ACEs include events external to the safety system, as well as conditions internal to the computer hardware or software (e.g., initialization status or buffer overflow), which have the potential for defeating the safety function. Software safety analysis activities include the development and implementation of a software safety plan that provides for thorough software safety analyses during the entire development process.

It is critical that software safety be viewed in the context of its associated hardware, environment, and operators, and that all software interfaces with these components be addressed. Therefore, a system-level analysis identifying the hazardous system states and the functions—including safety-related actions—to be performed by the software is a prerequisite. The safety importance of this framework element is intrinsic to system safety assurance.

Two subelements were defined for this framework element to address the major aspects of the software safety analysis activity. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Management Aspects**—Ensure the development and implementation of a proper software safety plan, including documentation of the results of safety analyses.

- **Technical Aspects**—Determine that software safety requirements analysis, software safety design analysis, software safety code analysis, software safety test analysis, and software safety change analysis are thoroughly performed to ensure that ACEs, including computer-unique failure modes, do not compromise system safety.

Software Operation and Maintenance

Software operation focuses on delivering system utility, including system operation and operational support services. Software maintenance refers to those activities required to keep a software system operational and responsive to users' needs by modifying it to correct faults, improving its performance or other attributes, or adapting it to a specified environment.

Changes to the software are inevitable and should be anticipated to occur during the 40- to 60-year expected lifetime of nuclear power plants. Many factors distinguish maintenance from new software development, for example, the transition to and the continuity of maintenance support, the need to monitor and change the operational system without disrupting it, the need to respond quickly to problem reports, and limited V&V of minor upgrades. These factors must be accounted for to ensure that safety is not compromised.

Three subelements were defined for this framework element. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Software Maintainability**—Facilitate modifications to a software system or component to correct faults, improve its performance or other attributes, or adapt it to a different environment, as well as facilitate tasks such as performance monitoring and change impact analysis.

- **Maintenance Planning**—Address activities that are critical to successful operational software support, especially those activities related to providing the continuity of software engineering support and V&V.

- **Performance Monitoring**—Focus specifically on proactive measures to assess the software and to maintain confidence in the operational system.

Software Configuration Management

Software configuration management (SCM) is a process that applies to the entire software life cycle, and contributes to software safety by ensuring the integrity of the executable code with respect to the components that constitute a particular version of the software. It provides the procedures necessary to ensure that (1) possible impacts of software modifications are evaluated before changes are made, (2) various software system products are examined for consistency after changes have been made, and (3) software is tested according to established standards after changes have been made.

Eight subelements were identified for this framework element. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **SCM Plan**—Address the management organization and the procedures, activities, schedules, and resources necessary to perform SCM for the software system.
- **Configuration Identification**—Require appropriate identification of configuration items so that they can be properly referenced, traced, and controlled.
- **Configuration Change Control**—Provide the controls necessary for managing and controlling the change process.
- **Configuration Status Accounting**—Provide for data management and tracking of the status of items under configuration control.
- **Configuration Audits and Reviews**—Provide a check on the completeness of a software product, and ensure that developers have satisfied external obligations.
- **External Interface Control**—Provide for the management of the interfaces with procedures, organizations, or items outside the scope of SCM proper, to prevent problems from arising in software system development.
- **Subcontractor and Vendor Control**—Ensure that SCM procedures apply to purchased and subcontractor-developed software, so that the safety and integrity of the entire software system can be guaranteed.
- **Automated Support for Configuration Management**—Provide for the use of automated tools since SCM on large systems is laborious and error-prone if performed manually.

Software Quality Assurance

The development of software that has the highest functional reliability requires the establishment of an effective software quality assurance (SQA) program and the verification, such as by checking, auditing, and inspection, that all activities affecting safety functions have been correctly performed. The criteria established in [10 CFR Part 50: Appendix B] provide the regulatory basis and the framework necessary to implement SQA program plans and procedures.

The candidate guidelines for this framework element are given under the subelement **Program Plan and Implementation**. The anticipated benefit of adherence to these guidelines is the further delineation of those aspects of an SQA program which should be an integral part of the development and management of a high integrity software system through its entire life cycle.

Software Planning and Management

Planning and management activities begin before software development starts, are conducted throughout the software development cycle, and continue through software operation and maintenance. The planning process focuses on the identification and scheduling of activities and resources required to complete a future project successfully. The management process is concerned with following those plans, controlling plan evolution, monitoring project activity and progress, and imposing risk mitigation actions.

The development of software that is important to safety demands significantly greater management and technical attention than that required for less stringently constrained software products. The reason for this increased attention is the critical need for high integrity software to perform its safety functions correctly. Therefore, extreme care must be taken during the development of the software to minimize the insertion of defects and to maximize the detection and removal of defects before it is placed into service. A critical part of this care must be guaranteed through well-planned development processes, which are monitored and controlled by effective management practices.

Two framework subelements provide the candidate guidelines on software planning and management. These subelements are listed below, together with the anticipated benefits of adherence to the candidate guidelines for each:

- **Software Development and Management Plan**—Provide for the overall planning and control of all software life-cycle activities.
- **Other Plans**—Provide for supporting project planning documentation (e.g., software safety plan, software test plan, and software maintenance plan).

5. Assessment of Technical Basis

The following five criteria stated in [Beltracchi, 1994] were considered in describing and assessing the adequacy of the technical basis for each candidate guideline:

- Criterion 1—The topic has been clearly coupled to safe operations.
- Criterion 2—The scope of the topic is clearly defined.
- Criterion 3—A substantive body of knowledge exists, and the preponderance of the evidence supports a technical conclusion.
- Criterion 4—A repeatable method to correlate relevant characteristics with performance exists.
- Criterion 5—A threshold for acceptance can be established.

In applying the above criteria to assess the technical basis for guidelines on computer software, it is necessary to recognize the unique character of software systems and the maturity level of the software industry. The following paragraphs summarize the general nature of the technical basis that is available for requirements on software and the manner in which the five criteria were applied to assess the technical basis for the candidate guidelines in this report.

Nature of Available Technical Basis

In the case of a software system, first principles are not available to provide the kind of technical basis that can be obtained for hardware. A software component is an abstract construct of interlocking concepts involving data sets, relationships among data items, and algorithms. Like all digitally encoded information, software results in discontinuous behavior, and possibly drastic consequences, with the smallest possible perturbation—the change of a single bit. Even sampling arbitrarily "close" to a fault may not reveal the cause for an observable failure. "There is no meaningful metric in which small changes and small effects go hand in hand, and there never will be" [Dijkstra, 1989]. Furthermore, many problems with developing software arise from its inherent complexity and the nonlinear increase in complexity with size. "From the complexity comes the difficulty of enumerating, much less understanding, all the possible states of the program, and from that comes the unreliability" [Brooks, 1987]. Based on a great deal of research completed on software-based systems in normal environments, it appears that deterministic evaluation of such complex systems is currently an intractable problem [Littlewood and Stringini, 1992; Butler and Finelli, 1993; Bennett, 1991; Zucconi, 1991; Lavine, 1990; Leveson et al., 1991; SAND93-2210].

In lieu of first principles, the technical basis might rely on empirical evidence that certain requirements will be effective in managing the complexity inherent in software systems, and in minimizing the introduction of defects into the software during its design and coding. Empirical correlations between specific software design characteristics and software reliability could help define the necessary software characteristics and thereby provide the technical basis for requiring those characteristics. However, the maturity of the software industry in relation to the high degree of complexity in software systems is such that empirical evidence of this nature is not well documented or established in a quantified manner. The majority of judgments on the quality of software are based on good practice, peer review, and expert opinion within the software engineering community [Fenton et al., 1994].

In the absence of first principles or a substantive body of empirical evidence to support requirements on software development, the safety and reliability of the overall system—comprising hardware, software, and user—can be, and generally are, assured by applying the defense-in-depth-safety principle, requiring an adequate level of diversity and redundancy in system design (including, where necessary, a combination of analog and digital systems), and emphasizing high quality in implementing system safety objectives [IEEE7-4.3.2:D; NUREG-0493; USNRC-BTP DRAFT].

As documented in this report, the technical basis for each guideline generally relies on standards that reflect the best practices in the software industry. In terms of its simplicity and reliance on best industry standards, the technical basis for many guidelines for software development is similar to

that for the quality assurance of safety-related hardware in nuclear plants [10 CFR Part 50: Appendix B] or the General Design Criteria; e.g., "The protection system shall be separate from control system . . ." [10 CFR Part 50: Appendix A: Criterion 24]. In general, the relationship to safety does not need much elaboration, as in the following example: "The software design should include self-supervision of control flow and data." Such guidelines for software development and the above examples of regulations for nuclear plant safety for which the technical basis is readily apparent may be contrasted with other regulations and guidance that are founded on extensive experimentation, testing, and analyses. An example of the latter is [10 CFR Part 50: Appendix K], which provides the requirements for evaluating the capability of emergency core cooling systems in nuclear power plants.

Finally, in examining the technical basis, all candidate guidelines in this report were considered to be relevant to the development of high integrity software, without regard to the relative degree to which individual guidelines contribute to safety. For example, the safety rationale for guidelines describing the attributes of completeness of SRS might differ from that for guidelines specifying the protection of critical data, or those providing for control of software configuration items. The technical basis (including the relationship to safety) for the candidate guidelines was not examined from the standpoint of their relative values and impacts.

Method and Results of Assessment

The set of five criteria previously mentioned provides an objective methodology for examining the technical basis. However, given the nature of the available technical basis as discussed in the preceding paragraphs, evaluation of the technical information supporting each candidate guideline against each criterion involved considerable subjectivity.

Since the candidate guidelines for a given framework subelement are closely coupled, the technical basis was developed and described for the group of guidelines as a whole. This information, which is provided in the main report (Volume 2), includes the following for each subelement: scope and relation to safety; excerpts from the baseline sources as supporting evidence; discussion of the baseline; any suggested additions to the baseline and their rationale; potential methods for meeting the guidelines; and assessment of the technical basis, including references to research needs identified to fill gaps in the technical basis.

Because the attributes of candidate guidelines for a given subelement were selected from baseline and other sources based on their relationship to safety and relevance to the scope of the topic covered in the subelement, the first two of the five criteria, namely those on relation to safety and definition of scope, were not used again in assessing the technical basis. The technical bàsis for each candidate guideline was evaluated with respect to the remaining three criteria on the strength of the evidence, the availability of a method for satisfying the guidelines, and the availability of a threshold of acceptance for determining conformance to the guidelines.

The supporting body of knowledge or evidence for the candidate guidelines was judged to be satisfactory, questionable, or unsatisfactory based on the following:

- Satisfactory

  (a) One baseline source (as defined above) or two other sources support(s) the candidate guideline.

  (b) There is no conflicting standard or issue with respect to (a).

- Questionable

  Criterion (a) is partially satisfied, or Criterion (b) is not satisfied.

- Unsatisfactory

  Criterion (a) is not satisfied.

Similarly, the availability of methods for meeting the candidate guidelines was judged to be satisfactory or unsatisfactory depending on whether or not a method is available, and judged as questionable if there is a significant issue that affects the effectiveness or adequacy of the method identified.

The assessment of the availability of a threshold for acceptance was based on whether different evaluators using an available set of evaluation criteria would come to the same answer (yes or no) concerning the conformance of the software to a given guideline. In several cases, implementing a candidate guideline involves the use of a standard (e.g., a standard for the development of a formal set of software test plans and specifications). In the cases where such a standard does not provide its own acceptance criteria, or if subjective judgment is also required to assess the quality of items required by the standard, the availability of a threshold was judged to be partial or questionable.

The results of the assessment of the technical basis for each candidate guideline are provided in Table ES-1 at the end of this volume. Of the 201 candidate guidelines, only 84 meet all the technical basis assessment criteria. The primary reason for this is the lack of a satisfactory threshold of acceptance (Criterion 5) for more than half of the candidate guidelines. Most of the candidate guidelines (188) have adequate supporting evidence (Criterion 3), although the method of implementation (Criterion 4) is inadequate for about a quarter of the guidelines.

6. Identification, Categorization and Prioritization of Research Needs

The lack of an adequate technical basis for a candidate guideline, including the presence of issues related to available implementation or review methods within a framework subelement, indicated a research need. A total of 61 research needs were identified to address the gaps in the technical basis. The words "research need" are used here to indicate any type of further work, such as a survey, which may or may not be characteristically labeled as "research." In identifying research

needs, the focus was on addressing generic issues, rather than specific methodologies. In several instances, the evaluation of conformance to a candidate guideline could involve considerable subjectivity on the part of an evaluator, even though appropriate standards for review exist. In these instances, where further research to establish a threshold was considered impractical, no research need was identified, although the threshold for acceptance (technical basis criterion 5) was judged to be questionable. All of the research needs identified are referenced in Table ES-1 according to the candidate guideline whose technical basis they are intended to support. The table also shows the guidelines whose threshold of acceptance was judged to be inadequate, but for which research was considered unnecessary. The main report (Volume 2) contains a discussion of the objective and the rationale for each research need. Research needs are organized according to the framework element they support and are discussed at the end of the corresponding section in Volume 2.

Since candidate guidelines and associated implementation approaches or issues are of interest to software developers and researchers, as well as to reviewers or auditors, there are obvious overlaps with respect to which organization might be interested in, or perform the necessary research identified to fill the gaps in the technical basis: the NRC, the nuclear industry, the software industry, or other research and regulatory organizations. However, as part of an overall assessment, those research needs that would directly support the NRC's regulatory function of performing reviews and audits were segregated from those that are more related to developing methods and tools for the industry to use in producing safe software. In addition, many of the research needs were judged as providing substantial support for both groups. Therefore, the identified research needs were grouped into three categories:

- Category A: Research needs that directly support the regulatory function
- Category B: Research needs that provide substantial support both to the regulatory function and to industry
- Category C: Research needs that support primarily industry, i.e., support the engineering of safe software

The number of research needs in each category is as follows: Category A—13; Category B—24; and Category C—24. The 37 research needs that support the regulatory function (Categories A and B), along with the rationale for their importance to that function, are presented in Table ES-2 at the end of this volume.

All of the research needs were then prioritized based on a subjective evaluation that considered the following specific perspectives: (1) relationship to safety; (2) degree of leverage provided by the research, based on the level of maturity of the software industry relative to the life-cycle activity (framework element) where the gap in the technical basis exists; and (3) type of research product, i.e., methods, criteria, measures, empirical data, or tools for implementing a guideline. As noted before, the prioritization did not involve a cost-benefit analysis.

With respect to the first perspective, although all research needs identified are related to safety, they could still be distinguished by how direct that relationship is. For example, research related to software reliability was judged to contribute more directly to safety than that related to software

maintainability. In using the second perspective, the framework elements of Software Requirements Specification and Software Safety Analysis were assessed as the least mature, and therefore, research related to these framework elements could be expected to provide the greatest leverage. The difficulty and importance of specifying valid software requirements was discussed in Section 2. The next level of maturity and relative leverage was reflected in the framework elements of Software Design and Software V&V—both static and dynamic. The remaining framework elements were judged to have a level of industry experience and standardization such that research related to those framework elements would provide relatively the least leverage. The third perspective used in prioritizing the research needs was the type of research needed to address the gap in the technical basis. Defining measures and criteria for evaluation was given the highest priority since it addresses the lack of useful and valid measures of software properties and other acceptance criteria, and also supports primarily the regulatory function. On the other hand, research addressing the automation of support (e.g., tools to generate test cases from requirements specifications) was assigned low priority.

All of the research needs were rated individually as high-, medium-, or low-priority according to each of the three perspectives. A composite priority rating of high/medium/low was then given based on the three individual ratings, as well as the consideration of any overlap with ongoing NRC research. While the details on the rating process are given in the main report (Volume 2), the composite ratings of Category A and B research needs are included in Table ES-2. Twelve of those research needs are indicated as high-priority; these are discussed in Section 8.

7. Discussion of Research Needed

Traditional methods and practices in software engineering generally are not based on standard engineering approaches and principles, such as the use of proven components and architectures and an empirical foundation of measurement and experimentation (e.g., [Fenton et al., 1994]). Thus, many of the gaps in technical basis identified in this report are related to two broad areas of concern that reflect the general lack of maturity of software engineering and that pose problems in developing high integrity software. These two areas are (1) lack of standardized software solutions for specific applications within the domain that software supports, and (2) lack of an empirical foundation based on measurement, analysis, and experimentation. The progress in these areas is slow, partly because of the inherent differences between the software product and the physical products of more traditional engineering fields. The digital nature of software precludes the traditional benefits of approximation and continuity, as discussed in Section 5. A brief discussion of the two areas of concern and the need to focus on research related to both process and product is presented below.

Standardization

Because software serves many disciplines, the practice of software engineering has tended to emphasize general methods that could be applied to the development of custom software for any application or discipline. However, the software industry has recently been developing a "software version" of the principles of proven technology and standardization used in other engineering fields. The software engineering discipline has now matured to the point where the

notion that every new software system is unique (i.e., has unique requirements and therefore a unique custom-built architecture and design) is giving way to the understanding that many software systems have much in common, and that this commonality can be exploited [Hooper and Chester, 1991]. Stability and standardization in the *system* requirements and architectures in an application area lead to stability in the requirements of the *software* systems in that application area [DOD, 1992]. There is potential for codifying and standardizing all aspects of software systems, including requirements, architectures, design, code, and the mapping between system and software, all of which become proven or standardized when that potential is realized in the application area. The manner in which these proven aspects of software support the regulatory function is the same as that for traditional engineering: as the software becomes accepted, standardized, and proven, it will be codified and used for developing regulatory acceptance criteria. In the case of high integrity software, the proven or standardized aspects will strengthen the basis for acceptance, which in the short term must rely on (1) a demonstration of an appropriate development process, which is only an indirect indicator of the quality and safety of a software product, and (2) an attempt to demonstrate clearly the quality and safety of newly developed custom software, which is expected to remain a difficult task for some time.

Several of the research needs set forth in this report would facilitate the standardization of various aspects of software systems that support nuclear power plant safety systems, in addition to addressing the specific gaps in technical basis. The general approach is called *domain-specific software engineering*, or *domain engineering*. It is viewed as an important theme cutting across the research needs because it benefits both industry and the regulatory function. Section 14.3.1 of the main report (Volume 2) provides further information on the domain engineering approach and discusses its applicability to the domain of nuclear power plant safety systems. An important aspect of this approach is that it addresses the interface between the system and the software. It provides greater benefit if standardization exists in the application area, and it encourages the development of more detailed guidance on system-software interface requirements. Domain engineering involves focusing on the specific software-supported application domain (in this case, nuclear power plant safety systems) and providing the following information, based on experience: (1) models of the behavior of software-supported systems in the application domain and their interfaces with other systems and with humans; (2) standard requirements specifications that cover the set of software systems in the domain; (3) one or more standard software architectures and lower-level designs that map to the standard specifications; and (4) code that implements the design and satisfies the requirements [Prieto-Diaz and Arango, 1991; Arango, 1994]. An important part of this information is the mapping between the system and the software, i.e., which system requirements and designs relate to which software requirements and designs, again based on experience—a knowledge of what works.

In defining the application domain that applies in this case, namely, the nuclear power plant safety systems, it is useful to distinguish two types of functions performed by the software. The first type is concerned with performing the plant safety functions themselves. These safety functions are identified at the system level and flow down through the software requirements, design, and code. The second type is concerned with maintaining the integrity of the system and software elements that perform the

primary safety functions, such as error checking and fault tolerance. The integrity functions are partially identified at the system level but are significantly expanded within the software itself. The domain engineering perspective on the research needs identified in the present work addresses the integrity aspects of both types of software functions: plant safety functions, which predominantly are specific to the nuclear power plant safety systems; and integrity functions, which might be common to other application domains, e.g., mission-critical aviation and space systems, or to software systems in general.

It is especially noted here that the availability of detailed system-level standards for defining the system-software interface, which currently do not exist, will provide considerable impetus to the domain engineering of software for nuclear plant I&C systems. This is in addition to the advantages they will offer towards developing complete and accurate software requirements specifications, the importance of which was discussed in Section 2.

Measurement and Experimentation

Another area of significant gaps is software measurement. Despite the amount of effort expended to date in the area of software metrics, there is little empirical foundation for software measurement and experimentation. Most of the "evidence" provided in the software literature is not based on the results of controlled experiments, but is instead either anecdotal evidence or "advocacy research" [Fenton et al., 1994]. Even "common-sense" claims, such as the notion that the use of formal notation leads to higher-quality specifications, are not necessarily supported by the experimental evidence. (In fact, [Naur, 1993] found evidence that the use of a formal notation can lead to more defects.) Given the lack of useful measures and experimental evidence, together with the significance of demonstrating the safety of software, the identification and prioritization of research needs as documented herein emphasized experimentation to facilitate progress in this difficult area.

Product- and Process-Related Research

Because of the difficulty of assessing the quality of software products, the currently accepted approach is to supplement the product assessment with an assessment of the process used to develop the software. The assumption is that certain "disciplined" processes tend to increase the quality and safety of the software that is produced. This approach is expected to continue playing a major role in software development and assurance until the two areas of concern discussed above are remedied. Therefore, the identified research needs span the life cycle and include both product- and process-related research.

8. High-Priority Research Needs Supporting the Regulatory Function

The twelve research needs in Categories A and B selected as high-priority were ranked subjectively based on overall considerations. The resulting priority sequence is shown in Table ES-3, and the rationale for the ranking of each research need is discussed below. A brief discussion of the importance of these research needs is also part of Table ES-2. Of the twelve research needs, eight are related to domain-specific software engineering; these are identified as "Domain" following their titles in Table ES-3.

1. Develop regulatory review criteria based on domain analysis of nuclear power plant software systems [R3-1/R6-1]

This is considered to be the most important research need for several reasons. First, it addresses the problem of lack of understanding of the interplay between software and the system context in which it is defined. Second, it captures common or standard proven requirements and criteria for evaluating requirements (not just the process of developing requirements); this should contribute to resolving for nuclear power plant safety software the largest problem area in software engineering in general, namely, specifying, understanding, and verifying software requirements. This research is the most substantial part—the anchor—of the domain-specific software engineering effort, whose importance has been described earlier in Section 7. Once the nuclear power plant software community understands what the requirements should be, those requirements can be used for new systems and incorporated into standard review criteria, rather than created anew for each system.

2. Identify common software-related ACEs or hazards based on domain-specific experience [R9-2]

This research need is perhaps the one most closely tied to safety among all the research needs identified in this report. Articulating and understanding the set of ACEs or hazards that can occur in nuclear power plant systems and software would help establish ways to avoid or mitigate them in current and future software systems. This research need is also an important part of the domain-specific software engineering effort, although it is somewhat more specific than the previous research need. For these reasons, it is nearly as important as the previous research need, and the two should be worked together.

3. Identify proven software architectures or designs that satisfy system safety principles (including diversity and redundancy) in software systems [R4-3]

Although the area of design in general is relatively mature in software engineering, the application of system safety principles to software design presents a new challenge because software cannot simply apply the methods and implementation approaches in the same way that they are understood and applied at the system level. The application of these principles to software is problematic because of the fundamental differences between software and the physical systems on which the safety principles and methods matured. For example, redundancy works for physical systems whose components wear out differentially, but does not work analogously for software because all copies of the software will have the same faults. Even attempts to achieve independence, and thereby avoid common-mode failures, through software design diversity (in the form of N-version programming) have been less than successful because the resulting versions do not appear to be mutually independent.

Therefore, the importance of this research is that it would identify proven architectures and designs based on experiential evidence. Increasing experience in domain-specific software engineering has resulted in greater recognition of the importance of a good software architecture to the overall quality of and confidence in software systems. Software architectures and designs that have proven themselves to be successful in meeting safety principles could be used in the

regulatory review to evaluate whether the software design for a new system is adequate in this respect. This research is related to research need No. 1 in that architectures and designs identified would correspond to the requirements and domain models identified as part of research need No. 1. This research is ranked high because of the proven technology and the broad scope involved, but is ranked lower than the above two research needs because they offer more direct leverage in terms of supporting requirements and safety analysis.

4. Determine definition and measurement of software reliability [R3-4/R6-4]

This research need is important in several respects. First, it defines a measure that can be used in the regulatory function. There is currently no adequate way to measure the ultra-high levels of software reliability required for nuclear power plants. The immaturity of software with regard to lack of measurement, as discussed earlier in this section, makes regulating software much more difficult. This research is intended to establish such a measure; the resulting criteria could be used in the regulatory review to evaluate software reliability requirements and assess whether they have been met. Second, reliability is very important because it is directly tied to safety. Third, this research need is related to requirements specifications, which, as indicated above, may be the biggest problem area for software development, as well as for V&V, because it would enable more precise reliability values to be specified as well as verified. It therefore addresses an important gap in the current ability to determine the reliability of a software system, or of a nuclear power plant safety system that includes software. This research goal would be difficult to achieve quickly, but it is important. It is ranked lower than the previous three research needs because it is concerned with the measure of a single attribute and does not contribute directly to the proven technology argument as they do.

5. Develop adaptable software operational profiles for each generic class of safety systems and a measure of their fit to specific systems [R7-14]

This research is important because it would establish a realistic profile or model of the pattern of data input to the software system. This is needed as the basis for statistical testing, and would also support the evaluation of reliability. This research need is important because of its relation to safety and to the domain engineering effort, but is ranked lower than the previous research needs because it is considered to offer slightly lower leverage than the requirements, safety, reliability, and architecture issues that constitute those research needs.

6. Determine common notation for or translation between system-level engineering and software safety requirements [R3-2/R6-2]

This research would provide a common proven mapping between the notation of the system requirements and that of the software requirements, and would therefore provide further system-software links. This is an important issue, but the reason this research is ranked lower than the previous ones is the narrower scope of the issue involved. This research supports the domain analysis effort under research need No. 1 and should be worked in conjunction with that effort.

7. Identify common safety software performance requirements based on domain-specific experience [R3-10]

This research serves a role analogous to that of research need No. 2 (common ACEs or hazards); however, in this part of the domain-specific software engineering effort, the common artifacts produced would be safety software performance requirements. These requirements would help ensure that timing constraints are met. Although this is an important area, this research need is ranked lower than those above because its scope is more narrow.

8. Conduct empirical study of software failures in high integrity systems [R7-2/R8-1/R9-4]

An understanding of software failure patterns in the system context could provide valuable feedback that would help resolve many of the errors and minimize failures in future systems. The resulting codified information could be incorporated into standard regulatory review criteria. This research is important because of its general relation to safety and to the proven technology argument of domain engineering. However, its leverage as test-related research is rated as somewhat lower than that of most of the previous research needs, and its relation to safety is perhaps slightly less direct as well.

9. Identify software architectures that use self-monitoring functions and other approaches to fault tolerance and still satisfy performance requirements [R4-6]

This research supports research need No. 3, but focuses more on the tradeoff between self-monitoring and fault tolerance versus performance and timing requirements. This is important, but the narrower focus of the research, along with its somewhat lower leverage as design-related research, lowers its ranking below that of the previous research needs.

10. Define criteria for acceptable design of error handling and performance margin [R4-4]

This research would provide criteria needed to adequately perform the regulatory review of software design with regard to meeting safety requirements. The general rationale for its ranking is similar to that for the previous research need; however, this research addresses somewhat lower-level design than the architectural issues addressed in the previous one, and it does not contribute directly to domain engineering. These two differences are responsible for the slightly lower ranking of this research.

11. Define criteria for acceptable design and use of software interrupts [R4-2]

This research also provides criteria needed to adequately perform the regulatory review of software design with regard to meeting safety requirements. The general rationale for its ranking is similar to that for the previous two research needs; however, this research is focused on the more narrowly defined issue of the design and use of interrupts, and that results in the slightly lower ranking of this research.

12. Identify criteria for evaluating adequacy of path coverage in software testing [R7-6]

This research is important because it would provide criteria for evaluating path coverage during software testing, which is related to safety. It is impossible to cover all paths in a software system, so it is important to determine a threshold and achievable types of coverage, and at the same time reduce to an acceptable level the risk involved in not testing some of the paths. Although testing in general is partially mature and understood in software engineering, criteria for path coverage are inadequate. Regulatory review could use these criteria in evaluating specified tests to determine whether they meet the threshold and cover the required types of paths. This research need is ranked last among the high-priority research needs because of a combination of the following: its leverage as test-related research is somewhat lower than that of most of the previous needs, its relation to safety is slightly less direct, and it does not directly support the proven technology argument of the domain engineering effort.

Table ES-1. Candidate Guidelines and Assessment of Technical Basis

Criteria for Assessment of Technical Basis

1. Relation to Safety
2. Definition of Scope
3. Body of Knowledge and Evidence
4. Existence of a Method
5. Threshold for Acceptance

Since Criteria 1 and 2 were used in selecting and developing the candidate guidelines, they were not used again in assessing the technical basis. Section 5 provides a discussion of the assessment of the technical basis, and defines the assessment criteria. The assessment results are shown in the table as Y (Satisfactory), Q (Questionable), or N (Not Satisfactory).

Identification Numbers of Candidate Guidelines and Research Needs. The first number corresponds to the number of the section in the main report in which the candidate guidelines and the research needs for each element are presented, while the second is sequential within the section. For example, [G3-5] and [R3-2] are the fifth candidate guideline and the second research need, respectively, in Section 3, on Software Requirments Specification in Volume 2. A research need applicable to more than one framework element has multiple identification numbers, and is discussed in the section of the main report identified by the first number. For example, [R3-1/R6-1] is discussed in Section 3 of Volume 2.

Acronyms are defined at the end of the table.

No.	Candidate Guideline	Assessment of Technical Basis: Criterion			Research Need
		3	4	5	
3. Software Requirements Specification					
Software Requirements Specification: Completeness					
G3-1	For each software input, the SRS should specify the following: • Data format requirements • Precision and accuracy requirements • How often the input must be read	Y	Y	Y	
G3-2	For each software output, the SRS should specify the following: • Data format requirements • Precision and accuracy requirements • How often the output must be updated	Y	Y	Y	
G3-3	The SRS should specify each abnormal hardware condition or event that the software must detect. Abnormal hardware conditions and events include failure of input or output device, processor, and memory.	Y	Y	Y	
G3-4	The SRS should specify each abnormal software condition or event that the software must detect. Abnormal software conditions and events include failure of software services at the operating-system level and failure of logic in application-level software (only general types of such failures can be identified prior to software design).	Y	Q	Q	R3-1/R6-1 R3-2/R6-2 R3-3/R6-3
G3-5	The SRS should specify the time-dependent input-to-output relation (i.e., the required functions) that the software component must implement. That is, the SRS should specify allowed sequences of outputs given any combination of sequences of inputs and abnormal conditions/events. "Sequence" here means an ordered list of <time, value> (or <time, event>) pairs, including startup of processing, response to interrupts, response for no input, and response to variations in sampling of data. The specification should cover all possible values (valid or invalid) of inputs.	Y	Q	Q	R3-1/R6-1 R3-2/R6-2 R3-10

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G3-6	The time-dependent input-to-output relation specified by the SRS should not permit any behavior that would violate any requirement (including safety requirements) that the system design allocates to the software component or any requirement that is derived from general system requirements.	Y	Q	Q	R3-1/R6-1 R3-2/R6-2 R3-10
G3-7	The SRS should specify the maximum percentage of available resources (processor, dynamic memory, mass storage, network capacity) that the software may use and should specify timing requirements, including speed, recovery time, and response time.	Y	Y	Y	
G3-8	The SRS should specify reliability requirements for the software.	Y	Q	Q	R3-4/R6-4
G3-9	Software maintainability requirements should be analyzed and specified prior to system design. These requirements should address coding standards and design constraints, as well as requirements for the supporting tools and documentation.	Y	Q	N	R3-5/R6-5
G3-10	The SRS should specify necessary design constraints (conformance to standards, hardware constraints).	Y	Y	N	R3-1/R6-1 R3-2/R6-2
Software Requirements Specification: Unambiguity					
G3-11	The SRS should be unambiguous. Each requirement should have only one interpretation.	Y	Y	Q	R3-6/R6-6
Software Requirements Specification: Consistency					
G3-12	The SRS should be consistent. The SRS should not contain requirements that contradict or conflict with each other.	Y	Q	N	R3-7/R6-7
Software Requirements Specification: Verifiability					
G3-13	Every requirement in the SRS should be verifiable. A method or process should exist (inspection, demonstration, analysis, or testing) that can show that an implementation fulfills the requirement.	Y	Y	Q	R3-4/R6-4 R3-5/R6-5 R3-8/R6-8 R3-9

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion			Research Need
		3	4	5	
Software Requirements Specification: Modifiability					
G3-14	The SRS should be written and structured so that modifications can be demonstrated to be correct.	Y	Q	N	R3-5/R6-5 R3-7/R6-7
G3-15	The SRS should not contain redundant requirements. The same requirement should not appear in more than one place in the SRS.	Y	Q	Q	R3-7/R6-7
Software Requirements Specification: Traceability					
G3-16	It should be possible to trace (1) between software requirements and system requirements/design, and (2) between software requirements and software design/code.	Y	Y	Y	
G3-17	The SRS should explicitly identify all those requirements that are relevant to the software's performing its safety function.	Y	Y	Y	
G3-18	The SRS should justify the inclusion of any software design or implementation detail.	Y	Y	Q	None Needed
Software Requirements Specification: Readability					
G3-19	Requirements in the SRS should be readable.	Y	Q	N	R3-6/R6-6

4. Software Design

Software Design: Modular Design					
G4-1	The software design should be divided into components using the information-hiding principle to produce components with interfaces that remove the design's implementation detail (or volatile design features).	Y	Y	Y	
G4-2	The critical data structures and related operations in the software design should be encapsulated within components.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G4-3	The degree of modularity for a component as a minimum should be based on the following: • Ease of understanding, taking into account both functional and structural complexity • Minimal ripple effect of changes • Cohesiveness of components	Y	Q	Q	R4-1
Software Design: External Interface					
G4-4	The software design should encapsulate external interface components that are separated from the application program in order to achieve the following: • Prevent ACEs from propagating to other components. • Ensure protocol compatibility between the application software and the external interface. • Ensure uninterrupted safety processing.	Y	Y	Y	
G4-5	The software design should encapsulate the processing of asynchronous or synchronous events from an external interface into a component.	Y	Y	Y	
G4-6	The software design should use interrupts only to simplify the system.	Y	Y	Q	R4-2
G4-7	The validation and error processing for an external interface should be thoroughly documented in the design specification.	Q	Y	Q	R4-2
G4-8	The software design should thoroughly document each usage of interrupt.	Y	Y	Q	R4-2
G4-8	(a) The software design should define processing duration and maximum number of occurrence constraints for external event processing (e.g., interrupts).	Y	Y	Y	
G4-8	(b) The software design should define the maximum duration for which an interrupt is disabled.	Y	Y	Y	
Software Design: Interfaces to Safety Components					
G4-9	Safety components should be identified to increase the focus on support for prevention of unnecessary interaction from nonsafety components.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion			Research Need
		3	4	5	
G4-10	The interfaces between safety and nonsafety components should be carefully designed to prevent unnecessary interactions with nonsafety component processing.	Y	Y	Q	R4-6
G4-10	(a) The software design should ensure that only the necessary information is communicated between safety and nonsafety components.	Y	Y	Q	R4-1 R4-6
G4-10	(b) The software design should use self-checking to ensure the accuracy of data produced for use in safety components.	N	Y	Q	R4-6
G4-10	(c) The software design should ensure that all communication between safety and nonsafety components is through their defined parameters.	Y	Y	Y	
G4-11	Safety software should reside in protected memory and periodically perform self-test to confirm that it has not been corrupted. The time interval between checks should be determined based on the safety importance of the function being performed.	Y	Y	Q	R4-6
Software Design: Interface Integrity					
G4-12	The software should perform parameter validation by checking the range of data used in communications to ensure that its value is within the defined domain.	Y	Y	Y	
G4-13	The software should perform protocol validation by doing the following: • Checking whether the requested action satisfies the conditions defined in the assertions stated for the interface. • Verifying the permission for the called component before performing the action requested.	Y	Y	Y	
G4-14	(a) The software should perform a plausibility check on the data being communicated to ensure that its value is accurate.	Y	Y	Y	
G4-14	(b) Communications messages should contain sufficient information to check for the completeness, correctness, correct ordering, and timeliness of the message data (e.g., sumchecks, sequence number timestamp).	N	Y	Q	R5-5/R4-7

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software Design: Data Integrity					
G4-15	The software design should explicitly identify the critical data and its usage context. Critical data is any data needed to maintain plant power operation, maintain plant safety, permit plant maneuvering, or establish operating limits and margin.	Y	Y	Y	
G4-16	The software should protect and monitor data storage.	Y	Y	Q	R4-3 R4-5 R4-6
G4-16	(a) The software design for data storage should address diversity and redundancy.	Y	Y	Q	R4-3
G4-16	(b) The software design should balance symmetric operation types (e.g., create-delete and open-close) to prevent corruption of data.	Y	Y	Y	
G4-17	The software should use a defensive technique for data storage.	Y	Y	Q	R4-3 R4-6
G4-17	(a) Each variable should be assigned a default value when it becomes available.	Y	Y	Y	
G4-17	(b) The use of techniques that allow critical data overwriting (e.g., circular structure) should be justified.	Y	Y	Q	R4-6
G4-17	(c) The software should define attributes for time tag, quality tag, and identification tag to be used for important data.	Y	Y	Y	
G4-18	The software design should include the consideration of security aspects as part of designing for data integrity.	Y	Y	Q	R4-3
G4-19	The software should implement access control for data storage.	Y	Y	Y	
G4-19	(a) The software design specification should specify the sequence of accesses for the set of critical data and shared data.	Y	Y	Y	
G4-19	(b) The software design should ensure that only authorized access to data is allowed.	Y	Y	Q	R4-3

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software Design: Flow Control					
G4-20	The degree of diversity and redundancy for high integrity software should be evaluated to determine what constitutes adequate support for diversity and redundancy at the system level.	Y	Y	Q	R4-3
G4-21	Software design should include self-supervision of control flow and data.	Y	Y	Q	R4-5 R4-6
G4-21	(a) The software should use timers to monitor safety processing for possible occurrence of blocking.	Y	Y	Y	
G4-21	(b) The software should ensure that self-checking processing will not prevent timely system response in any circumstance.	Y	Y	Q	R4-5 R4-6
G4-21	(c) The software should use plausibility checks for safety-related calculations.	Y	Y	Y	
G4-21	(d) The software should have on-line diagnostics that include both plausibility checks and periodic testing of hardware status.	Y	Y	Q	R4-5 R4-6
Software Design: Error Handling					
G4-22	(a) The software design should define processing for internal ACEs not specified in the SRS.	Q	Y	Q	R4-4
G4-22	(b) The processing of internal ACEs should be identified as derived requirements and be included in the safety analysis for the software so that potential effects at the system level can be identified.	Q	Y	Q	R4-4
G4-23	The software design should have a global error-handling strategy to maintain consistency in the processing of errors.	Y	Y	Y	
G4-23	(a) The software design should define a limit on ACE propagation, and guidelines for ACE handler design.	Y	Y	Y	
G4-23	(b) The identification for ACEs should use a descriptive name to convey its usage context.	Y	Q	Y	R4-4

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G4-23	(c) The software design should define a central facility for reporting ACEs and capturing their contextual information.	Q	Y	Y	None Needed
G4-23	(d) The software design should produce well-defined outputs for ACE processing.	Y	Y	Q	R4-4
G4-23	(e) The software design should ensure that any status information returned from the called service is evaluated before the processing continues.	Y	Y	Y	
G4-24	ACEs associated with a component should be identified.	Y	Y	Q	R4-4
G4-25	The occurrence of an ACE should not interrupt safety processing. Protection against critical failures should be provided through strategies for fault-tolerant design and quality control.	Y	Y	Q	R4-4
G4-26	A performance margin should be provided. The computer system should be designed with a sufficient performance margin to perform as designed under conditions of maximum stress. Those conditions include data scan, data communication, data processing, algorithm processing, analytical computation, control request servicing, display processing, operator request processing, and data storage and retrieval, at a minimum. Thus the computer system should be designed with reasonable expansion capability that would permit an owner to add other functions in the future.	Y	Y	Q	R4-4 R4-6

5. Software Coding

	Software Coding: Development Environment				
G5-1	(a) The development environment should consist of proven tools with vendor support.	Y	Y	Q	R5-1 R5-5/R4-7
G5-1	(b) Besides satisfying the development requirements, development tools should comply with a standard or an industry benchmark that requires a certification process, if such a standard or benchmark exists.	Y	Y	Q	R5-1 R5-5/R4-7
G5-2	The tools in the development environment should be tested for the intended application area before use.	Y	Y	Q	R5-1 R5-5/R4-7
G5-2	(a) Evaluations of development tools should be based on safety requirements identified in safety analysis.	Y	Q	Y	R5-3 R5-4

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G5-2	(b) Each tool should be analyzed for unique features that might directly or indirectly introduce errors into the implementation (i.e., during usage or transfer of data).	Y	Y	Q	R5-1
G5-3	The development environment should consist of a minimum number of different tools.	Y	Y	Q	R5-1
G5-4	The development environment should be maintained under configuration management.	Y	Y	Y	
G5-5	High-level programming languages should be used to implement the software design. Otherwise, the justification for the language selection should be provided.	Y	Y	Y	
G5-6	The use of assembly language to implement the software design should be limited to special cases, such as optimization.	Y	Y	Q	None Needed
G5-7	There should be detailed programming guidelines for each language used in the implementation.	Y	Y	Y	
Software Coding: Target Environment and Reusable Components					
G5-8	(a) The target environment and reusable products should be tested for the intended application before use.	Y	Q	Q	R5-2 R5-3 R5-4
G5-8	(b) The target environment and reusable product evaluations should be based on safety requirements identified in safety analysis.	Y	Q	Y	R5-2 R5-3 R5-4
G5-9	The target environment and reusable products should be selected from proven products with vendor support.	Y	Q	Q	R5-3 R5-4
G5-9	(a) There should be a plan describing the strategy and the time period for which vendor support is obtained for the target environment and reusable products.	Y	Q	Q	R5-3 R5-4
G5-9	(b) The target environment and reusable products used in implementing the software design should comply with industry interface standards, when they exist (e.g., POSIX [IEEE1003.1]).	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G5-9	(c) If a target environment or a reusable product was not developed in accordance with a process standard required for its development, then the development organization should analyze and test the software to ensure that the product complies with product standards and provide the necessary documentation.	Q	Y	Q	R5-3 R5-4
G5-9	(d) For a target environment and reusable products that are COTS, the development organization should obtain maintenance guarantees from the vendor.	Q	Y	Q	R5-3 R5-4
G5-10	The target environment or reusable products should be maintained "as delivered" under configuration management.	Y	Y	Y	
G5-11	Dependency on the target environment's services (e.g., operating system services) should be minimized.	Y	Y	Q	R5-3 R5-4
Software Coding: Data Structure					
G5-12	Data structures should be evaluated to ensure the correct usage.	Y	Y	Q	R5-5/R4-7
G5-13	Common variables should be grouped together.	Y	Y	Y	
G5-14	Installation-specific data should not be hardcoded.	Y	Y	Y	
G5-15	Array structures should be accessed using simple addressing mechanisms.	Y	Y	Q	R5-5/R4-7
G5-16	When addressing an array, its bounds should be checked.	Y	Y	Y	
G5-17	Each data type and subtype should have a defined range.	Y	Y	Y	
G5-18	The nesting of data record structures should be kept to a minimum and presented clearly.	Y	Y	Y	
Software Coding: Logic Structure					
G5-19	The logic structure of a component should contain only one entry point and one exit point.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G5-20	The nesting of logic structures should be kept to a minimum and presented clearly.	Y	Y	Y	
G5-21	(a) Multiple complex operations should not be attempted in a single statement.	Y	Y	Y	
G5-21	(b) Parentheses should be used for multiple arithmetic operations in a statement to show the precedence order of the operations, even if not required by the compiler's rules for evaluation of arithmetic expressions.	Y	Y	Y	
G5-22	Branch and loop structure should be handled cautiously.	Y	Y	Q	R5-6
G5-22	(a) Backward branches should be avoided; loop statements should be used instead.	Y	Y	Y	
G5-22	(b) No branch into loop structure should occur.	Y	Y	Y	
G5-22	(c) No branch out of a loop should occur unless it leads to the end of the loop.	Y	Y	Y	
G5-22	(d) "Go to" statements should be avoided.	Y	Y	Y	
G5-22	(e) Each case-controlled statement should specify all case conditions, and a default branch should be reserved for failure handling.	Y	Y	Y	
G5-22	(f) Each loop structure should be constrained with a maximum limit or a termination condition.	Y	Y	Y	

6. Software V&V—Static

Software V&V—Static: Requirements V&V					
G6-1	V&V of the software requirements should demonstrate that the requirements are complete. It should address functional, performance, timing and sizing, database, security, interface, and software quality attribute requirements; identification of safety requirements; ACEs and the responses to them; and special operating conditions.	Y	Q	Q	R3-1/R6-1 R3-2/R6-2 R3-3/R6-3 R3-4/R6-4 R3-5/R6-5 R3-6/R6-6 R3-7/R6-7 R3-8/R6-8

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G6-2	V&V of the software requirements should demonstrate that the requirements are unambiguous, consistent, verifiable, modifiable, traceable, and readable. These requirement quality attributes are defined by the candidate guidelines on SRS.	Y	Q	N	R3-1/R6-1 R3-2/R6-2 R3-3/R6-3 R3-4/R6-4 R3-5/R6-5 R3-6/R6-6 R3-7/R6-7 R3-8/R6-8
Software V&V—Static: Design V&V					
G6-3	V&V of the software design should demonstrate that the design correctly implements the software requirements. V&V should demonstrate that the software design is complete in the following ways: • The highest level of software design satisfies the software requirements and the software safety constraints. • Each level in the software design satisfies the requirements and safety constraints of the next-highest level of the software design.	Y	Q	N	R6-9
G6-4	V&V of the software design should demonstrate that the design is consistent.	Y	Q	Q	R6-9
G6-5	V&V of the software design should demonstrate that the design is traceable to the software requirements: • Each part of the design can be traced to one or more software requirements. • There is adequate justification for any functionality in the software design that is not specified in the software requirements. • Parts of the design that are relied upon in any way to implement software safety constraints are readily distinguished from parts that are not, and there is evidence that this partitioning of the design is correct.	Y	Y	Q	R6-9
G6-6	V&V of the software design should demonstrate that the software design is testable.	Y	Q	Q	R6-10

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software V&V—Static: Code V&V					
G6-7	V&V of the code should demonstrate that the code correctly implements the software design. V&V should demonstrate that the source code satisfies the requirements and safety constraints in the lowest level of the software design. As a special consideration, V&V should identify sources of ACEs introduced in coding and demonstrate that these cannot prevent the software from satisfying the safety constraints.	Y	Q	N	R6-11 R6-12 R6-13
G6-8	V&V should demonstrate that there is adequate two-way traceability between the source code and the design specifications, and that no unintended functionality has been added.	Y	Y	Q	R6-11 R6-12 R6-13
G6-9	V&V should demonstrate that the source code and object code are compatible with constraints on timing and resource utilization (processor, network, memory, mass storage).	Y	Q	Q	R6-11 R6-12 R6-13
G6-10	V&V should demonstrate that the object code is a correct translation of the source, that is, that the object code implements the behavior specified in the source code.	Y	Q	Q	R6-12

7. Software V&V—Dynamic (Testing)

Software V&V—Dynamic (Testing): Management Aspects Affecting All Test Levels					
G7-1	A formal set of test plans and specifications should be developed in accordance with [IEEE829].	Y	Y	Y	
G7-2	Software test reporting should be in accordance with [IEEE829]. These reports include the test-item transmittal report, test log, test-incident report, and test summary report.	Y	Y	Y	
G7-3	The team that conducts the testing should define, collect, and analyze metrics related to the progress, effectiveness, and content of each test phase.	Y	Q	Q	R7-8 R7-9

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software V&V—Dynamic (Testing): Technical Aspects Affecting All Test Levels					
G7-4	The software test specifications for safety functions should specify response to failure under test, including suspension of testing, analysis, changes, configuration management of changes, resumption of testing, and rules for retesting.	Y	Y	Q	R7-1 R7-9 R7-20
G7-5	The expected results for each test case should be defined so as to prevent misinterpretation of results.	Y	Q	Y	R7-1
G7-6	The software test specifications should be traceable to the SRS and the software design.	Y	Y	Q	R7-1
G7-7	The test plans and specifications should be reviewed for completeness, consistency, and feasibility.	Y	Q	N	R7-1 R7-9 R7-20
G7-8	Implementation of the software test plan should be repeatable and should include regression test cases.	Y	Q	Q	R7-1 R7-9 R7-20
G7-9	At a minimum, functional, structural, and random testing should be performed on all software and data that is resident on the computer at run time (i.e., applications software, network software, interfaces, operating systems, and diagnostics).	Y	Y	Q	R7-1
G7-10	Testing of previously developed or purchased software should be conducted in accordance with [IEEE1228: 4.3.11], *Previously Developed or Purchased Software*. Activities include testing the previously developed or purchased software both *independently of* and *with* the project's software.	Y	Q	Q	R7-1 R7-3 R7-4/R9-5
G7-11	The design, development, testing, and documentation of software tools and techniques developed in support of developing and testing the safety system software should entail the same rigor and level of detail as the safety system software.	Y	Q	N	R7-1 R7-3 R7-4/R9-5

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software V&V—Dynamic (Testing): Unit Testing					
G7-12	Each unit should be tested on the target platform *or* with actual external interface devices if they are available to verify that the unit complies with the specified requirements and implements the design correctly.	Y	N	N	R7-5 R7-18
G7-13	The unit test effort should use both code execution and source code review and analysis.	Y	Y	Y	
G7-14	Unit test specifications should be developed according to [IEEE1008], *IEEE Standard for Software Unit Testing.*	Y	Y	Q	R7-6 R7-9
G7-15	Test cases should be developed that execute all functions designed into the code. The coverage should be monitored, and paths not executed should be identified. The risk associated with the nonexecuted paths should be evaluated and tested. Unit test coverage should be based on criteria in [IEC880: Appendix E].	Y	Q	Q	R7-6 R7-9 R7-19/R9-7 R7-20
Software V&V—Dynamic (Testing): Integration and System Testing					
G7-16	The integration and system-level tests should be developed and the test results evaluated by individuals who are familiar with the system specification and who did not participate in the development of the system.	Y	Y	Q	R8-4/R7-21
G7-17	The software tests should provide 100 percent system interface coverage during integration test.	Y	Q	Y	R7-12
G7-18	Test cases should include execution of extreme and boundary values, exception handling, long run times, utilization of shared resources, workloads with periods of high demand and extreme stress, and special timing conditions.	Y	Q	Q	R7-2/ R8-1/R9-4 R7-4/R9-5 R7-7/R9-6 R7-19/R9-7
G7-19	Test cases should stress the system's performance to determine under what conditions tasks fail to meet their deadlines and whether a fixed subset of critical tasks will always meet their deadlines under transient overload conditions. The response time to urgent events, the schedulability, the performance margin, and the stability of the system should be assessed.	Y	Q	Q	R7-2/ R8-1/R9-4 R7-4/R9-5 R7-7/R9-6 R7-19/R9-7

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G7-20	A simulator with failure injection capability that uses the complete software system in its target processors should be used as an environment for system testing.	Y	Y	Q	R7-20
G7-21	Software reliability growth should be measured during the software system test. A description of the model(s) used should be supplied. The description should include a list of assumptions made in constructing the model and all information necessary to enable verification.	Y	Q	Q	R7-10 R7-11 R7-13 R7-14 R7-15 R7-16
G7-22	The reliability of the software should be demonstrated using a reliability demonstration test. A description of the operational profile used should be supplied. The description should include a list of assumptions made in constructing the profile and all information necessary to enable verification.	Y	Y	Q	R7-10 R7-11 R7-13 R7-14 R7-15 R7-16
G7-23	Testing should include system users, i.e., operators and maintenance personnel, to evaluate interfaces, system performance, and system response.	Y	Y	Y	
Software V&V—Dynamic (Testing): Installation Testing					
G7-24	Installation tests should be executed independently by the plant operating staff, to the extent practical in the plant simulator and eventually in the plant itself.	Y	Y	Q	None Needed
G7-25	Installation tests should include tests for the presence of all necessary files, as well as the contents of the files; hardware devices; and all software components (e.g., underlying executive version, firmware in displays, and anything to which the software could be sensitive).	Q	Q	Q	R7-4/R9-5 R7-17
G7-26	Installation tests should test response, calibration, functional operation, and interaction with other systems. Interfaces that were simulated during integration or factory acceptance testing should be exercised.	Y	Y	Q	R7-17
G7-27	Installation tests should include those operations that are judged to be difficult or inconvenient for the operating staff.	Q	N	N	R7-4/R9-5 R7-17

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G7-28	Installation tests should include conditions in which periodic in-service tests are carried out and new releases are installed. This includes regression tests, as well as support for periodic surveillance testing of nuclear plant safety systems described in [IEEE338].	Y	Q	N	R7-4/R9-5 R7-17
G7-29	All installation data, including test results, should be under configuration control.	Y	Y	Y	

8. Software V&V—General Considerations

Software V&V—General Considerations: V&V Plan					
G8-1	A software V &V plan should be developed. This plan should meet the requirements of [IEEE1012], including in particular the minimum requirements for critical software in [IEEE1012] and [IEC880].	Y	Y	Q	None Needed
Software V&V—General Considerations: V&V Reports					
G8-2	The results of software requirements V&V, design V&V, and code V&V should be documented in accordance with [IEEE1012].	Y	Y	Y	
G8-3	The V&V reports should be part of the development of the safety case, i.e., should give the reasoning that would lead one to believe that the software correctly implements all safety-related requirements and constraints. This reasoning should address the adequacy of the evidence provided by each V&V effort that is reported on.	Q	Q	N	R7-2/ R8-1/R9-4
G8-4	All data from the development process, including V&V reports of failures, anomalies, and corrective actions, should be available for inspection.	Y	Y	Q	R7-2/ R8-1/R9-4
Software V&V—General Considerations: V&V of Tools					
G8-5	Automatically generated source code that becomes part of the operational system should be subject to the same degree of V&V as manually produced code.	N	Q	Y	R8-2 R8-3

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G8-6	A tool that is used in a V&V argument should be subject to a degree of V&V that is appropriate to (a) the criticality of the tool in the V&V argument and (b) the criticality of the argument in the overall safety argument.	Q	Q	Q	R8-2 R8-3
Software V&V—General Considerations: Independence of V&V					
G8-7	Software V&V should be performed by individuals other than those who designed the software.	Y	Y	Y	
G8-8	The V&V process should be audited by an independent organization.	Y	Y	Q	R8-4/R7-21

9. Software Safety Analysis

Software Safety Analysis: Management Aspects					
G9-1	Prior to initiating the development of safety-critical software, a comprehensive software safety plan should be developed in accordance with [IEEE1228].	Y	Y	Q	R9-1
G9-2	There should be a single person responsible for the software safety program, and this individual should have sufficient organizational independence and authority from the development organization to ensure that the program is conducted properly.	Y	Y	Y	
G9-3	Documentation of the results of safety analyses required for safety-critical systems may be independent or integrated with other project documents. The set of information required from safety analyses should include the following: • Results of software safety requirements analysis • Results of software safety design analysis • Results of software safety code analysis • Results of software safety test analysis • Results of software safety change analysis	Y	Y	Q	R9-3
G9-4	Each completed set of information used in or resulting from a safety analysis activity should be maintained under configuration control to ensure that changes are controlled, and only current data are distributed.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software Safety Analysis: Technical Aspects					
G9-5	Software requirements essential to accomplishing the safety function should be analyzed to identify ACEs that could prevent the software (if implemented as given in the SRS) from meeting all software safety requirements.	Y	Q	Q	R9-2 R9-3
G9-6	The design of software essential to accomplishing the safety function should be analyzed to identify ACEs that could prevent the software (if implemented as given in the design) from meeting all software safety requirements given in the SRS.	Y	Q	Q	R9-2 R9-3
G9-7	The safety-critical code should be analyzed to identify ACEs that, when executed, could prevent it from meeting all software safety requirements addressed in the software design.	Y	Q	Q	R9-2 R9-3
G9-8	Test cases should be derived from the ACE analysis to include execution of rare conditions (abnormal events, extreme and boundary values, exceptions, long run times, etc.), utilization of shared resources, workloads with periods of high demand and extreme stress, and special timing conditions.	Y	Q	Q	R7-2/ R8-1/R9-4 R7-4/R9-5 R7-7/R9-6 R7-19/R9-7 R9-2 R9-3
G9-9	Software safety analysis should be performed on all changes to specifications, requirements, design, code, systems, equipment, test plans, descriptions, procedures, cases, and testing, unless it can be shown to be unnecessary because of the nature of the change. The starting point of the change analysis should be the highest level within the system that is affected by the proposed change.	Y	Q	Q	R9-2 R9-3
G9-10	Software change analysis should show that the change does not create a hazard, does not impact on a previously resolved hazard, does not make a currently existing hazard more severe, and does not adversely affect any high integrity computer software component. Software change analysis should verify that all affected documentation accurately reflects all safety-related changes that have been made in the software.	Y	Q	Q	R9-2 R9-3 R10-3/R9-8
G9-11	The safety framework for a high integrity software system should be examined for adverse conditions arising from the overlaps between safety issues and security issues.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
10. Software Operation and Maintenance					
Software Operation and Maintenance: Software Maintainability					
G10-1	Software should be designed to simplify and reduce the amount and difficulty of the maintenance required over the lifetime of the software. It should be designed for maintenance in accordance with accepted human factors engineering principles.	Y	Q	N	R10-1 R10-2
G10-2	A baseline, not only of the software, but also of the documentation and support tools (e.g., CASE tools and test cases), should be maintained as a basis for continuing support. The baseline should include all items necessary to maintain the system. The consistency and integrity of this baseline should be maintained.	Y	Y	Q	R10-2
Software Operation and Maintenance: Maintenance Planning					
G10-3	Maintenance planning should emphasize continuity of activities from initial development through deployment to operational support to meet the same criteria and rigor as applicable to the initial development.	Y	Y	N	None Needed
G10-4	Maintenance plans should identify all resources used or generated during software development that will be needed to provide continuing support during the operation and maintenance phase. They should also describe any procedures required for transitioning these items to the support organization, including the training required to familiarize new staff with the software system and support procedures.	Y	Y	Q	R10-4
G10-5	Maintenance planning should address the full range of software engineering activities. It should, at a minimum, address the full scope of [IEEE1219].	Y	Y	Y	
G10-6	Change management procedures should explicitly address the operational need to be responsive to maintenance requests. Such procedures must address how information from the following three viewpoints is recorded and used for planning: operational need, engineering alternatives, and programmatic constraints. Separate procedures should be required for scheduled builds and emergency maintenance.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G10-7	Once computer hardware, software, or firmware has been procured as a commercial grade item, and accepted through a commercial grade dedication process, this commercial dedication should be maintained as follows: • Changes to the computer hardware, software, or firmware commercially dedicated should be traceable through formal documentation. • Changes to the computer hardware, software, or firmware commercially dedicated should be evaluated in accordance with the process which formed the basis for the original acceptance. • A written evaluation should be performed by the commercial dedicator to provide adequate confidence that all changes to the computer hardware, software, or firmware commercially dedicated have been performed in accordance with the approved design and the V&V process. This evaluation should include consideration of the potential impact of computer hardware revisions on software or firmware. • Any changes by the original product designer to the approved design or V&V process should be evaluated in accordance with the design or V&V steps required by standard [IEEE7-4.3.2]. • Commercial grade dedication of computer hardware, software, or firmware for a specific safety system application cannot be construed as an approval for use of the commercially dedicated item for all safety system applications.	Y	Y	Y	
G10-8	V&V of modifications to software should be performed in accordance with [IEEE1012]. It should provide the assurance that the modification was correctly designed and properly incorporated, and that no other software or system functions are adversely impacted by the modification.	Y	Q	Q	R10-3/R9-8 R10-5
Software Operation and Maintenance: Performance Monitoring					
G10-9	A plan should be developed for a systematic and documented performance monitoring program. Performance monitoring requirements should be analyzed and specified prior to the initial system design. The necessary supporting tools and procedures should be considered during system design. Software monitoring should be performed at specified internal software boundaries, not limited to external observations of system performance.	Y	Y	Q	R10-6
G10-10	Performance of software should be monitored. Utilization of processing resources should be monitored and reallocated as necessary to satisfy the performance requirements. Known failure modes, observed failure rates, and any anomalous behavior should be addressed within the context of the safety analysis report. Software anomalies should be routinely collected and reported.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
11. Software Configuration Management					
Software Configuration Management: SCM Plan					
G11-1	An SCM plan should be established in accordance with [IEEE828]. This standard should apply to the entire software life cycle.	Y	Y	Y	
G11-2	SCM should meet the guidelines specified in [IEEE1042], [ASME-NQA-2a: Part 2.7, 5], and [ASME-NQA-1].	Y	Y	Y	
Software Configuration Management: Configuration Identification					
G11-3	SCM configuration identification should be defined and performed in accordance with [IEEE828] and [IEEE1042].	Y	Y	Y	
G11-4	The software configuration items and support or test software items to which SCM will be applied should be identified. The software configuration items should include both commercially purchased and custom-developed software.	Y	Y	Y	
G11-5	A configuration baseline should be defined at the completion of each major phase of the software development. Approved changes created subsequent to the baseline should be added to the baseline. The baseline should define the most recently approved software configuration. A labeling system for configuration items should be implemented that does the following: • Uniquely identifies each configuration item. • Identifies changes to configuration items by revision. • Provides the ability to uniquely identify each configuration of the revised software available for use.	Y	Y	Y	
G11-6	A document transfer package that identifies the configuration-controlled items required to be maintained throughout the software life cycle should be developed. It should contain the information required for the plant owner to maintain the software at the same level established by the software developer.	Y	Y	Q	None Needed

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software Configuration Management: Configuration Change Control					
G11-7	Configuration control procedures should be defined and performed in accordance with [IEEE828] and [IEEE1042].	Y	Y	Y	
Software Configuration Management: Configuration Status Accounting					
G11-8	Configuration status accounting should be performed in accordance with [IEEE828] and [IEEE1042].	Y	Y	Y	
Software Configuration Management: Configuration Audits and Reviews					
G11-9	Configuration audits and reviews should be performed in accordance with [IEEE828] and [IEEE1042].	Y	Y	Y	
Software Configuration Management: External Interface Control					
G11-10	SCM interfaces to other (non-SCM-related) activities, organizations, or items should be defined in accordance with [IEEE828] and [IEEE1042].	Y	Y	Y	
Software Configuration Management: Subcontractor and Vendor Control					
G11-11	Software configuration items should include commercially purchased software and subcontractor-developed software. SCM on such software should be performed in accordance with [IEEE828]and [IEEE1042].	Y	Y	Y	
G11-12	After the code has been tested and verified on the purchasing organization's system, the software should be placed under configuration management. From this point forward, the code should be handled and treated as software developed by the organization, and the software life cycle is implemented.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G11-13	The software engineer should maintain records of all commercially purchased software, including the version numbers of the software used to perform calculations and the dates they were run. In addition, the purchasing organization should have a systematic means of informing all past code users of updates, of bugs that have been identified and fixed, and of planned changes to the software.	Y	Y	Y	
Software Configuration Management: Automated Support for Configuration Management					
G11-14	There should be automated support for SCM, particularly for change control, defect recording, and status accounting. Such automated support should be selected and applied in accordance with [IEEE1042].	Y	Y	Y	

12. Software Quality Assurance

Software Quality Assurance: Program Plan and Implementation					
G12-1	An SQA program should be established in accordance with the 18 criteria identified in [10 CFR Part 50: Appendix B], *Quality Assurance Criteria for Nuclear Power Plants and Fuel Reprocessing Plants.*	Y	Y	Y	
G12-2	An SQA plan should be in existence for each new software project at the start of the software life cycle, or for procured software when it enters the purchaser's organization.	Y	Y	Y	
G12-3	The SQA plan should address all quality assurance procedures required during all phases of the software life cycle.	Y	Y	Y	
G12-4	The tasks of quality assurance should be run generally in parallel with the other tasks of the life cycle.	Y	Y	Y	

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion			Research Need
		3	4	5	
G12-5	A set of management policies, goals, and objectives is necessary to guide the implementation and application of the SQA program. Upper levels of management should recognize that SQA is a vital part of the software development process, and that software development, implementation, operation, and maintenance are similar to other engineering processes subject to quality assurance requirements. This recognition by upper management should be translated into a commitment through policies that set software quality goals; establish SQA functions; and authorize the resources in terms of people, money, and equipment necessary to perform the tasks.	Y	Y	Q	None Needed
G12-6	SQA personnel should possess technical experience in software development, specification, design, and testing. Senior technical staff are preferable to administrative project management staff. SQA personnel should have technical currency.	Y	Y	Q	None Needed
G12-7	The SQA organization should have a charter, with each element of the organization defined and its responsibility outlined. The elements responsible for SQA should be independent from those responsible for software development.	Y	Y	Y	
G12-8	The interfaces between the SQA organization and the software development organization should be carefully defined.	Y	Y	Y	
G12-9	The organizational elements responsible for each SQA task should be identified.	Y	Y	Y	
G12-10	The SQA program should provide in-depth training in the elements of software engineering and SQA for all personnel performing activities affecting quality. This includes training in software design and development techniques, as well as SQA procedures.	Y	Y	Q	None Needed

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Continued)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
G12-11	The SQA plan should identify the documentation to be prepared during the development, verification, use, and maintenance of the particular software system. It should identify the organizational elements responsible for the origination, V&V, maintenance, and control of the required documentation. It should also identify the specific reviews, audits, and associated criteria required for each document. It should identify the tools, techniques, and methodologies to be followed during the audits; checks and other functions that will ensure the integrity of the software products; required documentation; and the management structure and methodology to be employed.	Y	Y	Y	
G12-12	SQA audits should evaluate the adherence to and effectiveness of the prescribed procedures, standards, and conventions provided in SQA program documentation. The internal procedures, the project SQA plans, configuration management, and contractually required deliverables, from both the physical and functional perspectives, should be audited throughout the life cycle. The SQA audit consists of visual inspection of documents, and nondocument products, to determine whether they meet accepted standards and requirements.	Y	Y	Y	

13. Software Planning and Management

	Software Planning and Management: Software Development and Management Plan				
G13-1	Prior to initiating the development of safety-critical software, project management personnel should develop a comprehensive software development and management plan for the entire life cycle. This plan, as well as other supporting plans for the successive life-cycle phases, should be updated as necessary and maintained under configuration control.	Y	Y	Q	R13-1 R13-2

Table ES-1. Candidate Guidelines and Assessment of Technical Basis (Concluded)

No.	Candidate Guideline	Assessment of Technical Basis: Criterion 3	4	5	Research Need
Software Planning and Management: Other Plans					
G13-2	The following planning documentation should be developed: • Software safety plan • Software quality assurance plan • Software test plan • Software verification and validation plan • Software configuration management plan • Software integration plan • Software maintenance plan Each plan should be completed prior to the development phase in which that plan will be applied. These plans should conform to the requirements stated in the software development and management plan. If the cited document formats are not used, developers should provide a mapping of their documentation to the content requirements specified by these formats.	Y	Y	Q	R13-1

ACRONYMS:

ACEs	Abnormal Conditions and Events
CASE	Computer-Aided Software Engineering
COTS	Commercial off-the-Shelf
SCM	Software Configuration Management
SQA	Software Quality Assurance
SRS	Software Requirements Specification
V&V	Verification and Validation

Table ES-2. Research Needs Supporting the Regulatory Function

Number and Category[1]	Title	Importance to Regulatory Function	Overall Priority[2]
R3-1/ R6-1 (B)	Develop regulatory review criteria based on domain analysis of nuclear power plant software systems	Developing and clearly defining new software requirements that are consistent with system requirements is one of the most significant problem areas in software engineering. But experience with existing requirements for nuclear power plant safety software systems exists and can be exploited. This research would provide common models showing the relationship between the system and the software, common software requirements satisfying system requirements and designs, and common review criteria that could be used to evaluate the specific software requirements for a new system. This research potentially is part of the domain-specific software engineering activity.	H
R3-2/ R6-2 (B)	Determine common notation for or translation between system-level engineering and software safety requirements	The translation of safety system requirements and constraints to software requirements is not straightforward and not well understood. Part of the difficulty is the difference in notation between the system and software levels. This research would provide standard notation for representing both system and software requirements, or at least a standard mapping between system and software requirements, that could be used to evaluate the specific software requirements for a new system. This research potentially is part of the domain-specific software engineering activity.	H
R3-4/ R6-4 (A)	Determine definition and measurement of software reliability	There is currently no adequate way to measure the ultra-high levels of software reliability required for nuclear power plant safety systems. This research would establish a standard definition of reliability and a method of measuring reliability for I&C systems; the resulting criteria could be used in regulatory review to evaluate software reliability requirements and assess whether they have been met.	H
R3-5/ R6-5 (A)	Determine definition and measurement of software maintainability	Maintainability is currently measured indirectly at best. This research would establish a standard definition of maintainability and a more direct method of measuring maintainability for I&C systems; the resulting criteria could be used in regulatory review to evaluate software maintainability requirements and assess whether they have been met.	M

[1]The first number corresponds to the section number in the main report (Volume 2), while the second is sequential within that section. For example, R3-1 is the first research need in Section 3 of Volume 2. A research need applicable to more than one framewrok element has multiple identification numbers, and is discussed in the section of the main report identified by the first number. For example, [R3-1/R6-1] is discussed in Section 3 of Volume 2.

[2]H = High; M, = Medium, L = Low; and an asterisk (*) indicates that the priority was lowered by one priority level because the identified research need may overlap with ongoing NRC research.

Table ES-2. Research Needs Supporting the Regulatory Function (Continued)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R3-8/ R6-8 (B)	Develop tools to measure test coverage of specifications	Functional testing of a software component should be based on that component's software requirements specification. To produce evidence that a set of functional tests is adequate, one should be able to measure the degree to which a set of test cases covers the requirements in a software requirements specification. There is currently no automated way to do this. This research would provide methods of and tools for measuring how well a set of test cases covers a set of requirements. Regulatory review could use these results to evaluate the process; that is, when test cases are produced, they could be evaluated before the tests are executed to determine the extent to which they cover each of the requirements.	M
R3-10 (B)	Identify common safety software performance requirements based on domain-specific experience	The software industry has experience in translating system performance goals into software requirements specifications, but the commonalities among requirements of similar software systems for each of the various nuclear plant types have not been identified and codified. If this information were captured, it could be reused and could mature more rapidly into "proven" requirements. This research would identify common software requirements specifying time-dependent software behavior that accounts for hazards and abnormal events, based on domain-specific experience. These requirements could be used in regulatory review to assess whether system performance and timing constraints have been accurately represented in the software requirements. This research potentially is part of the domain-specific software engineering activity.	H
R4-1 (A)	Develop quantitative measures for evaluating modularity	Modularity is an established way to reduce risk in the design by separating complex problems into manageable parts, but there is currently no established measure of modularity. This research would develop quantitative criteria, based on accepted software engineering principles of information hiding, for evaluating modularity. These criteria could be used in the regulatory review of the design to determine whether there is adequate modularity to reduce the risk.	M
R4-2 (A)	Define criteria for acceptable design and use of software interrupts	System design determines where interrupts are needed in the software design, but software interrupts are known to be error-prone. This research would examine system design constraints to determine how to minimize the need for interrupts in software designs. The results could be used in regulatory review to determine whether the interrupt issue has been addressed properly and interrupts kept to a minimum.	H

Table ES-2. Research Needs Supporting the Regulatory Function (Continued)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R4-3 (B)	Identify proven software architectures or designs that satisfy system safety principles (including diversity and redundancy) in software systems	While system safety principles such as diversity and redundancy in system design are well understood, their application to software is not well understood, in part because it is not straightforward. However, the premise of this research is that safe software systems exist and can be identified as such. The architectures or designs of these systems would be analyzed to determine how they satisfy system safety principles. This research would provide standard architectures and designs proven to satisfy safety principles. These could be used to evaluate the specific software design in a new system. This research potentially is part of the domain-specific software engineering activity.	H
R4-4 (A)	Define criteria for acceptable design of error handling and performance margin	Adequate error handling is one of the most important areas of design and implementation of high integrity software systems because this part of the software is most closely tied to safety functions, particularly in terms of responding to abnormal events. Regulatory review of these systems should have a way to determine the adequacy of error handling. This research would provide a threshold that could be used to determine the acceptability of the error-handling design of a new system.	H
R4-5 (A)	Develop criteria for evaluating self-monitoring designs that use watchdog processing on a separate coprocessor	No criteria are available to evaluate self-monitoring designs that use watchdog processing on a separate coprocessor to perform flow control processing, including checking processing duration and correctness of processing flow. This research would provide more objective and precise criteria for evaluating the design of this type of system to ensure that the monitoring does not interfere with safety.	M*
R4-6 (B)	Identify software architectures that use self-monitoring functions and other approaches to fault tolerance and still satisfy performance requirements	There is some risk that the additional software required to support self-monitoring functions, as well as other approaches to fault tolerance, could increase the risk of failures, particularly if the monitoring functions are not adequately isolated from the primary functions. This research would assess these risks for high integrity software, and identify (from an experience basis) standard software architectures that include the self-monitoring and other fault-tolerant functions without violating the timing constraints. This includes addressing tradeoffs among approaches to self-monitoring, such as using the same processor for application and monitoring functions versus using a separate coprocessor for monitoring (as described under research need [R4-5]). This research potentially is part of the domain-specific software engineering activity.	H

Table ES-2. Research Needs Supporting the Regulatory Function (Continued)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R5-1 (A)	Define evaluation criteria for development environments for producing high integrity software	Part of the evaluation of the process of developing a high integrity software system involves evaluating the adequacy of the software environment used in the development. Even though many publications address evaluation of individual tools for high integrity software system development, there is no standard set of criteria for the overall tool environment in which the software is developed and tested. This research would provide criteria for evaluating the adequacy of the development environment, for example, whether the tools are sufficiently integrated.	M
R5-3 (B)	Define framework and criteria for certification of reusable components	There are safety advantages to using components that have been successfully used in other safety systems, rather than developing new components. However, there are also disadvantages, partly because the development organization typically has little knowledge of the context or pedigree of a component, which can affect its use and its safety in the new system. Evaluation criteria for reusable components are critical for certifying the quality and safety of the components, but these certification criteria are not clearly defined. This research would define an evaluation framework and develop criteria for certification of reusable components. These could be used in regulatory review to evaluate whether code developed in another, perhaps unknown, context is safe to use in a new high integrity system. This research potentially is part of the domain-specific software engineering activity.	L*
R5-4 (B)	Determine feasibility of and criteria for using COTS products in high integrity software systems	There are problems associated with using COTS products in high integrity software systems for which the proposed solutions are only partial. For example, maintenance of someone else's software is difficult, and modifying a COTS product may terminate its support from the vendor. This research would provide standard rules or guidelines for using COTS products in high integrity software systems, and identify tradeoffs from both the technical and organizational perspectives. These could be used to evaluate whether specific COTS products are safe to use in a new high integrity system. This research potentially is part of the domain-specific software engineering activity.	L*
R6-9 (A)	Define thresholds for evaluating design completeness, consistency, and traceability to requirements	Part of regulatory review is evaluating the design of safety functions. However, an auditor does not have clear criteria or thresholds for evaluating whether the design is adequate, and whether design guidelines have been met. This research would support that function by providing thresholds of completeness, consistency, and traceability to requirements that could be used as objective criteria for determining the acceptability of the design in these areas.	M

Table ES-2. Research Needs Supporting the Regulatory Function (Continued)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R6-10 (B)	Develop measure of design testability	It is difficult to evaluate whether the candidate guideline of producing a testable design has been met. This measure would provide more objective means of making that determination in the regulatory review.	M
R6-11 (B)	Identify reverse engineering support for code V&V	It is assumed that the regulatory function will include performing at least some key V&V activities. Reverse engineering could be used in the regulatory review as part of static code analysis.	M
R6-12 (A)	Survey automated support for code analysis	Under the assumption that regulatory review will include V&V activities, this research would provide tools covering many aspects of static code analysis.	L*
R6-13 (A)	Develop evaluation criteria for formal verification of source code	Formal verification of source code offers potential benefit for software quality and safety. However, there is a lack of standard techniques for assessing the adequacy of a formal verification effort. This research would develop evaluation criteria for regulatory review to determine whether a formal verification of source code in a high integrity software system has been performed satisfactorily.	L*
R7-1 (B)	Determine relative efficiency of test strategies and resources required for each	Testing experience reports are rare in the literature. Additional research is needed on the ability of various testing strategies to trigger failures, and the resources required as a function of execution time, test runs, and types of failures. This research would determine which testing strategy identifies the most failures in the software and what resources are required to implement each strategy. The regulatory review could use the results of this research to evaluate the adequacy of test strategies either before or after the tests are run.	M
R7-2/ R8-1/ R9-4 (B)	Conduct empirical study of software failures in high integrity systems	Almost all large software projects perform disposition of and log information about software failures, but there has been little collection and dissemination of this information on a larger scale. This research would provide an understanding of standard software failure patterns and a list of common errors or failures in existing software systems. These could be used to evaluate whether the software for a new system has been tested for these errors, and to demonstrate that they are not present. This research potentially is part of the domain-specific software engineering activity.	H

Table ES-2. Research Needs Supporting the Regulatory Function (Continued)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R7-3 (A)	Determine extent of testing required to certify existing software	Certification of existing software (see research need [R5-3]) can potentially be done through a combination of static analysis; review of pedigree, including process and usage; and testing. This research would explore what is required for certification through testing, including the extent of testing with the new system being developed. The results could be incorporated into regulatory guidelines for assessing whether the testing of existing software is adequate for certification in a new system.	L*
R7-6 (A)	Identify criteria for evaluating adequacy of path coverage in software testing	Since it is impossible to cover all paths in testing a software system, it is important to determine a threshold and types of coverage required that are possible to achieve, and at the same time reduce to an acceptable level the risk involved in not testing some of the paths. Although testing in general is partially mature and understood in software engineering, criteria for path coverage are inadequate. Regulatory review could use these criteria to evaluate specified tests to determine whether they meet the threshold and cover the necessary types of paths.	H
R7-8 (B)	Develop metrics for evaluating test effectiveness and test progress	Data on the productivity and efficiency of testers is available, but it is sparse. This research would provide metrics showing the benefit-to-cost ratio of additional testing, which would be good indicators of testing progress, and profiles of software test duration, which would enable comparison of a development project with industry convention or averages for safety applications. These would assist the regulator in process evaluation.	M
R7-9 (B)	Develop criteria for reviewing test plans and specifications	Quality attributes for test plans and specifications are not well defined or measurable. This research would provide auditors with guidance on the analysis of these test documents as part of regulatory review.	M
R7-10 (B)	Evaluate reliability demonstration test techniques	It is difficult to apply reliability growth models to systems with a low number of faults, including high integrity software developed with good quality assurance practices. This research would compare approaches for reliability demonstration testing and produce guidance that could be incorporated into regulatory review for assessing whether an adequate approach was used in a new system.	M

Table ES-2. Research Needs Supporting the Regulatory Function (Continued)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R7-11 (B)	Develop procedures for reliability prediction and allocation	Software reliability growth models rely on data generated after the design and coding, hence have little impact on the planning and design stages of the project. This research would develop procedures for software reliability prediction in order to allocate reliability to software components and evaluate designs (as is commonly done for hardware reliability) prior to software testing. The results of this research could be used in regulatory review of the process to estimate reliability in the design stage of the project.	L*
R7-14 (B)	Develop adaptable software operational profiles for each generic class of safety systems and a measure of their fit to specific systems	[OECD HALDEN, 1993] concludes that none of the software reliability growth models represent an entirely adequate fit to any of the data sets to which they are applied. This research would provide standard software operational profiles of data input to the software system. These could be used in the regulatory review to determine whether realistic test suites were used (especially statistical testing) and to evaluate the reliability of a new system. This research potentially is part of the domain-specific software engineering activity.	H
R7-16 (B)	Develop techniques for estimating the reliability of distributed systems	Most software reliability growth models are based on centralized architectures. Safety systems are increasingly using a distributed software architecture, which requires more complex reliability growth models. This research would develop models for distributed systems with common software platforms (i.e., operating systems, and databases) and for unique application services executing on different hardware platforms connected by networks. The techniques developed could be used in regulatory review to estimate the reliability of these distributed systems.	M
R8-2 (A)	Develop method for assessing high-level language compiler risks	Some risk is introduced into safety systems by commercially available compilers. The risk is that the compiler does not correctly translate source code into object code and therefore may affect plant safety. Many compiler defects pose a risk to safety, but not all. For example, a compiler may abort when attempting to translate a particular construct; this poses a problem for the software developer, but does not constitute a safety problem. This research would address whether high-level language compilers or optimizing compilers are less deterministic and therefore higher-risk. The results of this research would directly benefit auditors in assessing the overall risk of using specific compilers by formulating realistic guidance for compiler usage for high integrity software systems, including the safety implications of changing compilers or upgrading to new versions of the same compiler.	M

Table ES-2. Research Needs Supporting the Regulatory Function (Continued)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R8-4/ R7-21 (B)	Determine interaction protocols for V&V activities	The conventional wisdom is that software V&V must be independent to be effective. However, there are some tradeoffs with the degree of independence. This research would develop interaction protocols that would provide an independent perspective (i.e., avoid repeating the same errors as in the development because of overfamiliarity with the approach used), but still address the issues accurately and competently (i.e., based on sufficient knowledge and understanding of the system concept and requirements). These protocols could be incorporated into criteria for regulatory review of V&V activities.	L
R9-1 (B)	Develop techniques for creating and evaluating software safety plans	There is little experience in applying standards for safety plans, such as [IEEE1228], to software development projects, and in monitoring project progress and activity against the plans. This research would develop techniques necessary for creating such a plan, and for monitoring actual project progress against planned progress. The results could be incorporated into safety plan evaluation criteria for regulatory review.	M
R9-2 (B)	Identify common software-related ACEs or hazards based on domain-specific experience	The identification of ACEs that software must address currently relies on performing, for each new system, safety analyses for requirements, design, code, test, and change. This research would provide a list of the minimum number of standard ACEs that could occur and that the new software system must prevent or mitigate. This list could be used to evaluate whether the new software system has addressed all the items on the list. This research potentially is part of the domain-specific software engineering activity.	H
R10-2 (B)	Identify typical safety-significant software changes during maintenance phase	There is a lack of empirical data showing the types and quantity of changes typically required during the operation and maintenance phase of software systems. This research would provide typical software changes that occur during maintenance of high integrity software for nuclear plants. The identified set of changes could be used in the regulatory review to determine whether these changes have been anticipated and accounted for in the design of the new system. This research potentially is part of the domain-specific software engineering activity.	M

Table ES-2. Research Needs Supporting the Regulatory Function (Concluded)

Number and Category(1)	Title	Importance to Regulatory Function	Overall Priority(2)
R10-3/ R9-8 (B)	Develop techniques for evaluating software modifications and limiting the safety impact of modifications	There are many circumstances under which operational software may be modified. Criteria are needed for determining what types of software modifications could raise safety concerns. Certain changes to software might not involve any significant safety issue, while others might be such that their effects could be mitigated through a proper implementation approach and adherence to certain practices. This research would develop techniques for evaluating software modifications and limiting the safety impact of those modifications. The results would be valuable for regulatory review in support of implementing the requirements of [10 CFR Part 50: 50.59].	M
R13-1 (B)	Streamline software management and development plans, and develop criteria for evaluating them	The diversity of software management and development plans and formats for these plans makes it an imposing task for any organization to verify the conformance, completeness, and accuracy of plans submitted. While the overall contents of the plans are remarkably consistent across the standards, the differences in format make document analysis particularly difficult. This research would develop effective and efficient mechanisms for extracting the relevant information and assessing it for consistency with the software safety plan, and for ensuring that such verification could be performed. It would also develop a rationale for merging certain plans that are likely to create conflicts. The results of this research would be useful in regulatory review of software management and development plans as part of process review.	M

Table ES-3. Ranking of High-Priority Research Needs Supporting the Regulatory Function

Priority Rank	Number*	Title
1	R3-1/ R6-1	Develop regulatory review criteria based on domain analysis of nuclear power plant software systems (Domain)
2	R9-2	Identify common software-related ACEs or hazards based on domain-specific experience (Domain)
3	R4-3	Identify proven software architectures or designs that satisfy system safety principles (including diversity and redundancy) in software systems (Domain)
4	R3-4/ R6-4	Determine definition and measurement of software reliability
5	R7-14	Develop adaptable software operational profiles for each generic class of safety systems and a measure of their fit to specific systems (Domain)
6	R3-2/ R6-2	Determine common notation for or translation between system-level engineering and software safety requirements (Domain)
7	R3-10	Identify common safety software performance requirements based on domain-specific experience (Domain)
8	R7-2/ R8-1/ R9-4	Conduct empirical study of software failures in high integrity systems (Domain)
9	R4-6	Identify software architectures that use self-monitoring functions and other approaches to fault tolerance and still satisfy performance requirements (Domain)
10	R4-4	Define criteria for acceptable design of error handling and performance margin
11	R4-2	Define criteria for acceptable design and use of software interrupts
12	R7-6	Identify criteria for evaluating adequacy of path coverage in software testing

*The first number corresponds to the section number in the main report (Volume 2), while the second is sequential within that section. For example, R3-1 is the first research need in Section 3 of Volume 2. A research need applicable to more than one framewrok element has multiple identification numbers, and is discussed in the section of the main report identified by the first number. For example, [R3-1/R6-1] is discussed in Section 3 of Volume 2.

REFERENCES

[10 CFR Part 50] U.S. Government, "Domestic Licensing of Production and Utilization Facilities," *Code of Federal Regulations,* Title 10, Part 50.

[Arango, 1994] G. Arango, "Domain Analysis Methods," *Software Reusability*, W. Schafer, R. Prieto-Diaz, and M. Matsumoto, editors, Ellis Horwood, pp. 17–49, 1994.

[ASME-NQA-2a] American Society of Mechanical Engineers, *Quality Assurance Requirements of Computer Software for Nuclear Facility Applications*, Part 2.7, ASME NQA-2a-1990, May 31, 1990.

[Basili and Perricone, 1984] V. R. Basili and B. T. Perricone, "Software Errors and Complexity: An Emperical Investigation," *Communications of the ACM*, Vol. 27, No. 1, pp. 42–52, January 1984.

[Beltracchi, 1994] L. Beltracchi, "NRC Research Activities,"*Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop*, September 13–14, 1993, conducted by the U.S. Nuclear Regulatory Commission, in cooperation with the National Institute of Standards and Technology, NUREG/CP-0136, p. 39, March 1994.

[Bennett, 1991] P. A. Bennett, "Forwards to Safety Standards," *Software Engineering Journal*, pp. 37–40, March 1991.

[Boehm, 1975] Barry Boehm et al., "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," *IEEE Transactions on Software Engineering*, Vol. 1, No. 1, pp. 125–133, March 1975.

[Brooks, 1987] Frederick P. Brooks Jr., "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, April 1987.

[Butler and Finelli, 1993] Ricky Butler and George Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering*, Vol. 19, No. 1, pp. 3–12, January 1993.

[Davis, 1990] Alan Davis, *Software Requirements Analysis and Specification*, Prentice Hall, New York, 1990.

[Dijkstra, 1989] Edsger W. Dijkstra, "A Debate on Teaching Computer Science," *Communications of the ACM*, Vol. 32, No. 12, December 1989.

[DOD, 1992] U.S. Department of Defense, DOD Software Reuse Initiative, *DOD Software Reuse Vision and Strategy*, Technical Report 1222-04-210/40, July 1992.

[DOD-STD-2167A] U.S. Department of Defense, *Defense System Software Development*, DOD-STD-2167A, 1988.

[EPRI, 1992] Siddharth Bhatt and Laurent Chanal, *Comparison of International Standards for Digital Safety Systems Verification and Validation*, Electric Power Research Institute, September 9–11, 1992.

[Fenton et al., 1994] N. Fenton et al., "Science and Substance: A Challenge to Software Engineers," *IEEE Software*, Vol. 11, No. 4, pp. 86–95, July 1994.

[Fujii, 1993] R. Fujii, "How Much Software Verification and Validation is Adequate for Nuclear Safety?," *Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop*, September 13–14, 1993, conducted by the U.S. Nuclear Regulatory Commission, in cooperation with the National Institute of Standards and Technology, NUREG/CP-0136, p. 250, March 1994.

[Grady, 1992] Robert B. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[Hooper and Chester, 1991] J. W. and R. O. Chester, *Software Reuse: Guidelines and Methods*, Plenum Publishing Corporation, New York, 1991.

[IAEA, 1993] International Atomic Energy Agency, *State of the Art Report on Software Important to Safety in Nuclear Power Plants*, Draft, May 13, 1993; subsequently published as "Software Important to Safety in Nuclear Power Plants," *Technical Report Series No. 367*, 1994.

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

[IEEE1012] Institute of Electrical and Electronics Engineers, Inc., *IEEE Standard for Software Verification and Validation Plans*, ANSI/IEEE 1012-1986.

[IEEE7-4.3.2] Institute of Electrical and Electronics Engineers, Inc., *Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*, ANSI/IEEE7-4.3.2-1993; including *Correction Sheet* issued December 27, 1994.

[Jones, 1991] Capers Jones, *Applied Software Measurement: Assuring Productivity and Quality*, McGraw Hill, New York, 1991.

[Lavine, 1990] C. Lavine, "Risk of Computer-Controlled Systems to Human Safety," *WESCON/90 Conference Record*, Electronics Convention Management, Los Angeles, CA, pp. 758–762, 1990.

[Leveson et al., 1991] Nancy Leveson et al., "Safety Verification of Ada Programs Using Software Fault Trees," *IEEE Software*, Vol. 8, No. 4, pp. 48–59, July 1991.

[Littlewood and Strigini, 1992] Bev Littlewood and Lorenzo Strigini, "The Risks of Software," *Scientific American*, pp. 62–75, November 1992.

[LLNL NUREG/CR-6101] J. Dennis Lawrence, *Software Reliability and Safety in Nuclear Reactor Protection Systems*, Lawrence Livermore National Laboratory, NUREG/CR-6101, June 11, 1993.

[MOD55] British Ministry of Defense, *The Procurement of Safety Critical Software in Defense Equipment*, Interim Defense Standard 00-55/Issue 1, 1991.

[Naur, 1993] P. Naur, "Understanding Turing's Universal Machine Personal Style in Program Description," *Computer Journal*, No. 4, pp. 351–371, 1993 (cited in [Fenton et al., 1994]).

[NIST, 1993] National Institute of Standards and Technology, *A Framework for the Development and Assurance of High Integrity Software*, Draft, October 7, 1993.

[NIST NUREG/CR-5930] National Institute of Standards and Technology, *High Integrity Software Standards and Guidelines*, NUREG/CR-5930, NIST SP 500-204, September 1992.

[NUREG-0493] U.S. Nuclear Regulatory Commission, *A Defense-in-Depth and Diversity Assessment of the RESAR-414 Integrated Protection Systems*, NUREG-0493, March 1979.

[OECD HALDEN, 1993] OECD HALDEN Reactor Project, *A Lessons Learned Report on Software Safety and Software V&V*, August 24, 1993.

[Prieto-Diaz and Arango, 1991] R. Prieto-Diaz and G. Arango, *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, Los Alamitos, CA, 1991.

[RTCA DO-178B] Radio Technical Commission for Aeronautics (RTCA, Inc), *Software Considerations in Airborne Systems and Equipment Certification*, RTCA/DO-178B, 1992.

[SAND93-2210] T. I. Barger et al., *Software-based Safety Subsystems for Nuclear Weapons*, Sandia National Laboratories, SAND93-2210, November 1993.

[URD] Electric Power Research Institute, *Advanced Light Water Reactor Utility Requirements Document Passive Plant*, NP-6780, Chapter 10, "Man-Machine Interface Systems," Vol. III, Rev. 4, 1993.

[USNRC-BTP DRAFT] U.S. Nuclear Regulatory Commission, Draft, Branch Technical Position (HICB), *Digital Instrumentation and Control Systems in Advanced Plants*, presented at the Digital Systems Reliability and Nuclear Safety Workshop, September 13-14, 1993.

[USNRC GL 95-02] U.S. Nuclear Regulatory Commission, *Use of NUMARC/EPRI Report TR-102348, 'Guideline on Licensing Digital Upgrades,' in Determining the Acceptability of Performing Analog-to-Digital Replacements Under 10 CFR 50.59*, Generic Letter 95-02, April 26, 1995.

[USNRC-RETROFITS DRAFT] U.S. Nuclear Regulatory Commission, *Operating Reactors Digital Retrofits, Digital System Review Procedures*, Draft, Version 1, presented at the Digital Systems Reliability and Nuclear Safety Workshop, September 13–14, 1993.

[USNRC SECY-91-292] U.S. Nuclear Regulatory Commission, *Digital Computer Systems for Advanced Light Water Reactors*, SECY-91-292, Sep ember 16, 1991.

[USNRC SECY-93-087] U.S. Nuclear Regulatory Commission, *Policy, Technical, and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor (ALWR) Designs*, SECY-93-087, April 2, 1993.

[USNRC SRM/SECY-93-087] Staff Requirements Memorandum (SRM) on SECY-93-087, July 21, 1993.

[Zucconi, 1991] L. Zucconi, *Software Safety and Reliability Issues in Safety-related Systems*, poster presented at the Thirteenth International Conference on Software Engineering, 1991.

NRC FORM 335 (2-89) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER (Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, If any.)

NUREG/CR-6263
MTR 94W0000114
Vol. 1

2. TITLE AND SUBTITLE

High Integrity Software for Nuclear Power Plants

Candidate Guidelines, Technical Basis and Research Needs

Executive Summary

3. DATE REPORT PUBLISHED

MONTH	YEAR
June	1995

4. FIN OR GRANT NUMBER

L2610

5. AUTHOR(S)

S. Seth, W. Bail, D. Cleaves, H. Cohen, D. Hybertson,
C. Schaefer, A. Ta, B. Ulery

6. TYPE OF REPORT

Technical

7. PERIOD COVERED *(Inclusive Dates)*

9/93 - 6/95

8. PERFORMING ORGANIZATION – NAME AND ADDRESS *(If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)*

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

9. SPONSORING ORGANIZATION – NAME AND ADDRESS *(If NRC, type "Same as above"; if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)*

Division of Systems Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

11. ABSTRACT *(200 words or less)*

The work documented in this report was performed in support of the U.S. Nuclear Regulatory Commission to examine the technical basis for candidate guidelines that could be considered in reviewing and evaluating high integrity computer software used in the safety systems of nuclear power plants. The framework for the work consisted of the following software development and assurance activities: requirements specification; design; coding; verification and validation, including static analysis and dynamic testing; safety analysis; operation and maintenance; configuration management; quality assurance; and planning and management. Each activity (framework element) was subdivided into technical areas (framework subelements). The report describes the development of approximately 200 candidate guidelines that span the entire range of software life-cycle activities; the assessment of the technical basis for those candidate guidelines; and the identification, categorization and prioritization of research needs for improving the technical basis. The report has two volumes: Volume 1, Executive Summary, includes an overview of the framework and of each framework element, the complete set of candidate guidelines, the results of the assessment of the technical basis for each candidate guideline, and a discussion of research needs that support the regulatory function; Volume 2 is the main report.

12. KEY WORDS/DESCRIPTORS *(List words or phrases that will assist researchers in locating the report.)*

High Integrity Software
Nuclear Plant Software
Software Standards
Safety-Critical Software Standards
Nuclear Plant Safety System Software

13. AVAILABILITY STATEMENT

Unlimited

14. SECURITY CLASSIFICATION

(This Page)

Unclassified

(This Report)

Unclassified

15. NUMBER OF PAGES

16. PRICE