

Evaluation of the Regulatory Review Process  
for the Software Development Life Cycle

by

Andrew Patrick Gnau

B.S., Systems Engineering (1995)  
United States Naval Academy

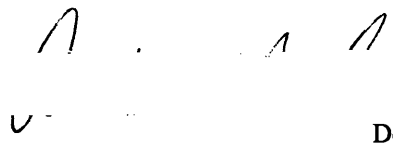
SUBMITTED TO THE DEPARTMENT OF NUCLEAR ENGINEERING  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN NUCLEAR ENGINEERING  
AT THE  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 1997

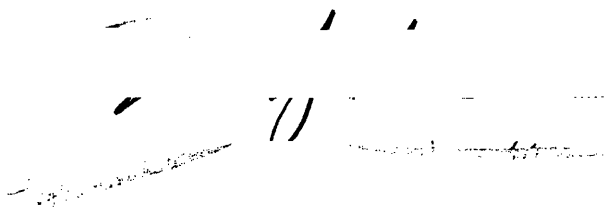
© 1997 Massachusetts Institute of Technology  
All rights reserved

Signature of Author .....



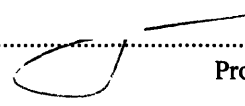
.....  
Department of Nuclear Engineering  
January 17, 1997

Certified by .....



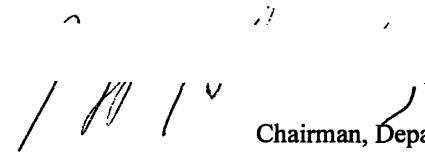
.....  
Professor George E. Apostolakis  
Thesis Supervisor

Certified by .....



.....  
Professor Michael W. Golay  
Thesis Reader

Accepted by .....



.....  
Professor Jeffrey P. Freidberg  
Chairman, Department Committee on Graduate Students

MAY 19 1997 Science

LIBRARIES



EVALUATION OF THE REGULATORY REVIEW PROCESS  
FOR THE SOFTWARE DEVELOPMENT LIFE CYCLE

by

ANDREW PATRICK GNAU

Submitted to the Department of Nuclear Engineering  
on January 17, 1997 in partial fulfillment of the  
requirements for the Degree of Master of Science in  
Nuclear Engineering

**Abstract**

The U.S. Nuclear Regulatory Commission (USNRC) has been dealing for some time with the issue of how to regulate safety-related digital instrumentation and control (I&C) software to assure that it is safe and reliable enough for introduction into nuclear power plant (NPP) applications. The review process that exists was designed with analog technology in mind. New plant designs and many proposed retrofits, however, involve digital systems that are less well understood and less suited to current evaluations than analog systems. The need for a new and more appropriate regulatory method is, therefore, becoming increasingly important.

The problem addressed in this work is to analyze the USNRC's proposed software development review standard -- Branch Technical Position HICB-14, or 'BTP-14,' "Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems" -- and Chapter 7 of the Standard Review Plan ("Instrumentation and Controls") in light of current technology in the software development field to determine how best the development process and the review standard can be applied to digital I&C technology. This analysis resulted in the five following suggestions for the standard:

1. The major points and guiding principles of BTP-14 and associated documents need to be expressed clearly and concisely in one location.
2. The guidance in BTP-14 and its associated documents should be consolidated into one stand-alone document or should have a clear roadmap to guide the reviewer from one document to another with minimal effort.
3. The guidance needs to have more than strictly a qualitative nature -- perhaps by requiring a mathematical notation for all requirements.
4. More emphasis needs to be placed on using software development techniques with a *system*-safety approach (e.g., checklists, HAZOP, information hiding, and the Dynamic Flowgraph Methodology).
5. A complementary mix of functional, structural, *and* random testing must be specified, possibly with more emphasis on error *prevention* (as is emphasized in the Cleanroom software engineering process) than on error detection and debugging.

Thesis Supervisor: George Apostolakis

Title: Professor of Nuclear Engineering



## **Acknowledgments**

First I would like to thank my advisor, Professor George Apostolakis, for his support and guidance in steering me around the trees so that I could better see the forest. I would also like to thank my fellow research project members, Chris Garrett and Xinhui Chen, for their valuable insights into how to improve my various thesis drafts, and my thesis reader, Professor Michael Golay, for taking the time to evaluate my final draft.

In addition, I would like to thank all of the people who allowed me to vent my thoughts, ideas, concerns, frustrations, and sometimes even joy in their direction: my great friends Yool Kim (without whose help and support I never would have made it), Jen Rochlis (my sister separated-at-birth), Rob McHenry (who also had the dubious pleasure of being my roommate), and my parents, Mr. and Mrs. Scott and Beth Gnau. Thanks also to Associate Professor Mark J. Harper of the United States Naval Academy, who helped me to get here in the first place, and to Drs. William Thompson and George W. W. Jones of Dallastown (Pa.) Area High School, my lifetime career counselors.



---

## Table of Contents

---

List of Figures.....	11
Important Acronyms.....	13
Frequently Referenced Documents .....	15
<b>1 Introduction.....</b>	<b>17</b>
1.1 Motivation and Problem Statement.....	17
1.2 Scope of the Work .....	17
1.3 Background .....	18
1.3.1 Digital Systems vs. Analog Systems .....	18
1.3.2 Digital I&C Safety Issues Identified by the National Research Council...19	
1.3.3 Discussion of Key Digital I&C Safety Issues.....	20
1.3.3.1 <i>Software Quality Assurance and Common-Mode Failure Potential</i> .....	20
1.3.3.2 <i>System Aspects of Digital I&amp;C Technology</i> .....	21
1.3.3.3 <i>Safety and Reliability Assessment and Case-by-Case Licensing</i> .....	24
1.3.4 Process vs. Product and Cleanroom Engineering .....	27
1.3.4.1 <i>Error Prevention, NOT Error Correction</i> .....	28
1.3.4.2 <i>Debugging</i> .....	29
1.3.4.3 <i>Simplicity</i> .....	29
1.3.4.4 <i>Functional (Correctness) Verification</i> .....	30
1.3.4.5 <i>Testing and Statistical Quality Control</i> .....	31
1.3.5 Fundamental Software Design Questions.....	32
1.3.6 The USNRC’s Approach to the Software Development Process .....	32
<b>2 Software Review.....</b>	<b>35</b>
2.1 The BTP-14 Review Process .....	35
2.1.1 BTP-14.....	35
2.1.2 SRP Appendix 7.0-A .....	35
2.1.3 SRP Section 7.0 .....	36
2.1.4 SRP Section 7.1 .....	36
2.1.5 SRP Appendix 7.1-A .....	39
2.1.6 SRP Appendix 7.1-B.....	39

2.1.7	SRP Appendix 7.1-C.....	39
2.1.8	Back to SRP Appendix 7.0-A.....	40
2.1.9	Back to BTP-14.....	42
2.2	Evaluation of the BTP-14 Review Process .....	46
2.2.1	Background.....	46
2.2.2	Evaluation .....	48
2.2.2.1	SRP Section 7.0.....	50
2.2.2.2	SRP Section 7.1.....	51
2.2.2.3	SRP Appendix 7.1-A.....	51
2.2.2.4	SRP Appendices 7.1-B and 7.1-C .....	52
2.2.2.5	SRP Appendix 7.0-A.....	53
2.2.2.6	BTP-14.....	59
2.3	The Place of V&V and Testing in the BTP-14 Review Process.....	62
2.3.1	V&V Definitions.....	62
2.3.2	V&V Theory .....	63
2.3.3	V&V in BTP-14.....	64
2.3.4	V&V in the SRP .....	67
2.3.5	V&V Standards: Reg. Guide 1.152 and IEEE 7-4.3.2.....	67
2.3.6	V&V Standards: IEEE 1012, IEEE 1028, and Reg. Guide 1.1yy.....	68
2.3.7	Testing and V&V in NUREG/CR-6101 .....	69
2.3.8	Methods and Types of Testing.....	70
2.3.9	What is the Right Mix of Testing Methods?.....	72
2.3.10	Problems with Testing .....	72
<b>3</b>	<b>Improving Software Design and the BTP-14 Review Process.....</b>	<b>77</b>
3.1	General Observations on BTP-14 and the Ontario Hydro Software Development Standard.....	77
3.1.1	Mathematical Notation.....	79
3.1.2	Mathematical Verification .....	79
3.1.3	Information Hiding .....	79
3.1.4	Testing.....	81
3.2	Recommended Actions to Improve the BTP-14 Review Process...	82
3.2.1	Recommendation #1 .....	82



3.2.2	Recommendation #2 .....	82
3.2.3	Recommendation #3 .....	82
3.2.4	Recommendation #4 .....	83
3.2.5	Recommendation #5 .....	83
<b>4</b>	<b>Conclusions .....</b>	<b>85</b>
	References .....	87
Appendix A	Comparison Between Analog and Digital Instrumentation and Control Systems .....	91
Appendix B	Development of the NRC List of Digital Instrumentation and Control Issues.....	93
Appendix C	Sample “Completeness Criteria” for Requirements.....	99
Appendix D	Lutz’s “Safety Checklist” .....	101
Appendix E	Sample Guidance from BTP-14 .....	103
Appendix F	Sample Guidance from SRP Section 7.0.....	107
Appendix G	Digital Issues Discussed in SRP Section 7.1 .....	111
Appendix H	Sample Guidance from SRP Appendix 7.1-A ....	113
Appendix I	Guidance Topics from SRP Appendix 7.1-B.....	115
Appendix J	Digital Issues Discussed in SRP Appendix 7.0-A .....	117
Appendix K	BTP-14 Software Characteristic Definitions .....	119
Appendix L	Safety Impact of Software Qualities from a Regulator Viewpoint .....	121

Appendix M	Testing Strategies Appropriate to Software Qualities.....	123
Appendix N	Sample Prerequisites for and Extent of Testing..	125
Appendix O	Typical Testing Strategies for Investigating Software Qualities .....	127
Appendix P	Human Factors.....	131

---

## List of Figures

---

<b>Figure 1</b>	Overview of Review Process .....	37
<b>Figure 2</b>	Overview of the Process for Reviewing the Unique Aspects of Digital Instrumentation and Control Systems .....	41
<b>Figure 3</b>	Software Review Process .....	43
<b>Figure 4</b>	Documents Produced During Each Life Cycle Stage .....	45
<b>Figure 5</b>	Defense-in-Depth and Diversity Review .....	54
<b>Figure 6</b>	Review of Software Lifecycle Process Planning.....	55
<b>Figure 7</b>	Special Considerations in the Review of Functional Requirements for Digital Instrumentation and Control Systems .....	56
<b>Figure 8</b>	Review of Software Development Process Implementation .....	57
<b>Figure 9</b>	Review of Design Outputs .....	58
<b>Figure 10</b>	Verification and Validation Activities .....	66
<b>Figure 11</b>	Illustration of BTP-14 Review Path Discussed in Chapters 2 & 3.....	74
<b>Figure 12</b>	Sample of Tabular Representation .....	80



---

## Important Acronyms

---

ACE	-- abnormal conditions and events
ACRS	-- Advisory Committee on Reactor Safeguards
AECL	-- Atomic Energy Canada, Ltd.
ANSI	-- American National Standards Institute
BTP	-- Branch Technical Position
CFR	-- Code of Federal Regulations
COTS	-- commercial-off-the-shelf (software)
DFM	-- Dynamic Flowgraph Methodology
D-in-D&D	-- defense-in-depth and diversity
FMEA	-- Failure Modes and Effects Analysis
FTA	-- Fault Tree Analysis
GDC	-- General Design Criteria
HAZOP	-- hazard and operability (analysis)
HF	-- human factors
I&C	-- instrumentation and controls
IEEE	-- Institute of Electrical and Electronics Engineers
MTTF	-- mean-time-to-failure
NPP	-- nuclear power plant
NRC	-- National Research Council
OH	-- Ontario Hydro
SQC	-- statistical quality control
SRP	-- Standard Review Plan
SRS	-- Software Requirements Specification
Std	-- Standard (as in IEEE Std 279)
SVVP	-- Software Verification and Validation Plan
USNRC	-- United States Nuclear Regulatory Commission
V&V	-- Verification and Validation



---

## Frequently Referenced Documents

---

(in order of appearance)

1. **'BTP-14' -- Branch Technical Position HICB-14 (Proposed)** -- "Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems" (Version 10.0), August 23, 1996.
2. **'SRP Chapter 7' -- Standard Review Plan, Chapter 7 (Draft) -- "Instrumentation & Controls"**
  - \* **Section 7.0** -- "Instrumentation and Controls -- Overview of Review Process" (Version 4.0), August 22, 1996.
  - \* **Appendix 7.0-A** -- "Review Process for Digital Instrumentation and Control Systems" (Version 7.0), August 23, 1996.
  - \* **Section 7.1** -- "Instrumentation and Controls -- Introduction" (Version 8.0), August 23, 1996.
  - \* **Appendix 7.1-A** -- "Acceptance Criteria and Guidelines for Instrumentation and Control Systems Important to Safety" (Version 8.0), August 23, 1996.
  - \* **Appendix 7.1-B** -- "Guidance for Evaluation of Conformance to ANSI/IEEE Std 279" (Version 8.0), August 22, 1996.
  - \* **Appendix 7.1-C** -- "Guidance for Evaluation of Conformance to IEEE Std 603" (Version 2.0), August 22, 1996.
3. **10 CFR 50 (and 52) and corresponding appendices** -- respective sections of the Code of Federal Regulations
4. **NUREG/CR-6101** -- "Software Reliability and Safety in Nuclear Reactor Protection Systems," November 1993.
5. **'IEEE 603' -- IEEE Std 603-1991** -- "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations."
6. **'IEEE 279' -- IEEE Std 279-1971** -- "Criteria for Protection Systems for Nuclear Power Generating Stations."
7. **Regulatory Guide 1.153** -- "Criteria for Power, Instrumentation, and Control Portions of Safety Systems," 1985.
8. **'Reg. Guide 1.152' -- Regulatory Guide 1.152** -- "Criteria for Digital Computers in Safety Systems of Nuclear Power Plants," January 1996.
9. **'IEEE 7-4.3.2' -- IEEE Std 7-4.3.2--1993** -- "IEEE Standard for Digital Computers in Safety Systems of Nuclear Power Generating Stations."
10. **'Reg. Guide 1.1yy' -- Regulatory Guide 1.1yy (Draft)** -- "Verification, Validation, Reviews, and Audits for Digital Computer Software."
11. **'IEEE 1074' -- IEEE Std 1074-1991** -- "IEEE Standard for Developing Software Life Cycle Processes."

12. **'IEEE 1012'** -- **IEEE Std 1012-1986** -- "IEEE Standard for Software Verification and Validation Plans."
13. **ANSI/ANS-10.4-1987** -- "American Standard Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry."
14. **'IEEE 1028'** -- **IEEE Std 1028-1988** -- "IEEE Standard for Software Reviews and Audits."
15. **NUREG/CR-6263** -- "High Integrity Software for Nuclear Power Plants," June 1995.
16. **'IEEE 830'** -- **IEEE Std 830-1993** -- "IEEE Recommended Practice for Software Requirements Specifications."
17. **'RG 1.1ww'** -- **'Reg. Guide 1.1ww'** -- "Software Requirements Specifications for Digital Computer Software."
18. **CE-1001-STD Rev. 1** -- "Standard for Software Engineering of Safety Critical Software," January 1995.



---

# 1 Introduction

---

## 1.1 Motivation and Problem Statement

The U.S. Nuclear Regulatory Commission (USNRC) has been considering for some time the issue of how to regulate safety-related (especially safety-critical) digital instrumentation and control (I&C) software to ensure that it is safe and reliable enough for introduction into nuclear power plant (NPP) applications. The process that exists was designed with analog technology in mind, but many new plant designs and proposed retrofits involve digital systems that are less well understood and less suited to current evaluations than analog systems. The need for a new and more appropriate regulatory method is, therefore, becoming increasingly important.

While digital systems have many advantages over analog systems, they also introduce new problems. Their discontinuous behavior is much less predictable than the continuous behavior of analog systems. That is, the behavior of a continuous analog system between discrete test points can be interpolated; but assumptions of such continuous or smoothly varying behavior for digital systems, in which discrete logic applies, are not valid.<sup>1</sup> In addition, many failures can be difficult to foresee due to the extremely large number of possible input, processing, and output states of a digital system. In general, both the performance and the failure mechanisms of digital systems are different from those of analog systems.<sup>1</sup>

The USNRC's (Proposed) Branch Technical Position HICB-14 ('BTP-14')<sup>2</sup>, "Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Safety Systems," is based on the principle that evaluation of software quality must include not only an evaluation of the software *product* itself (as has traditionally been the focus), but also an evaluation of the software development *process*. The problem then is to analyze the USNRC's proposed software development process review in light of current technology in the software development field to determine how best the development process and the review process can be applied to digital I&C technology.

## 1.2 Scope of the Work

The contents of this work are as follows. Section 1.1 of this introductory chapter discusses the work motivation and problem statement, followed by discussion of the scope of the work in this section. Section 1.3 discusses the background to the problem. This includes a basic comparison of digital and analog technology, a brief summary of important digital I&C issues followed by a more detailed discussion of the most pertinent ones, a discussion of the software development process vs. the software product, a

listing of the fundamental software questions to be answered, and brief discussion of the USNRC's approach to the software review process. Chapter 2 discusses the software review process. Section 2.1 gives an objective discussion of the USNRC's proposed review process for software and instrumentation & control; 2.2 evaluates this review process; 2.3 discusses the specific issue of V&V within the BTP-14 regulatory process. This covers V&V definitions and theory; the place of various V&V standards; various types and mixes of testing; and problems with testing. Chapter 3 deals with improving the BTP-14 review process and the software development process. Section 3.1 compares and contrasts BTP-14 with the Ontario Hydro software review standard; 3.2 makes suggestions for how to improve the BTP-14 review process. Chapter 4 summarizes the conclusions of this work. (The initial focus of this work was human factors. The focus soon changed, but some of the pertinent information from that pursuit has also been included in the final appendix, Appendix P.)

## 1.3 Background

### 1.3.1 Digital Systems vs. Analog Systems

Digital, software-based technology offers many advantages over analog, hardware-based technology, including reduced calibration (less device "drift") and maintenance, flexible and improved information displays, and improved accuracy. Another notable difference with software is the absence of a "wear-out" period of age-related failures, which can be important when traditional, hardware-based safety mechanisms are replaced with new, software-based alternatives. However, digital technology also brings with it problems not seen in the same way -- if at all -- in analog systems, such as the potential for common-mode failures.<sup>3</sup> As indicated in one of the most important digital I&C regulatory documents, the *key* difference that makes it so much more difficult to assure reliability in digital systems is the discontinuous behavior they exhibit.<sup>1</sup>

The National Research Council (NRC) was asked by the USNRC to define "the important safety and reliability issues (concerning hardware, software, and man-machine interfaces) that arise from the introduction of digital instrumentation and control technology in nuclear power plant operations, including operations under normal, transient, and accident conditions."<sup>3</sup> The NRC noted a variety of inherent differences between analog and digital systems in the areas of data collection, data transmission, control logic implementation, operator interface, reliability, and preoperational and in-service testing. (The complete description of these differences is in Appendix A.<sup>3</sup>)

The NRC identified some potential digital I&C problems based on recent retrofit experiences. These problems include the areas of common-mode failures; diversity as protection against common-mode failures; software specification development; software verification and validation; excessive complexity (and resultant programming difficulties); environmental sensitivity (to such things as electromagnetic and radio frequency interference, temperature, humidity, and smoke); lack of on-site plant experience with the

new technology; commercial dedication of hardware and software; reliability of software tools; configuration management; and the need for an efficient regulatory certification process that can keep up with rapidly changing digital technology.<sup>3</sup>

### 1.3.2 Digital I&C Safety Issues Identified by the National Research Council

In order to develop a specific list of safety issues related to the introduction of digital technology into NPP I&C, the NRC studied the relation of digital technology to five standard ‘approaches to safety’ given below. A brief description of the concern(s) about each issue is also given.<sup>3</sup>

- 1) *Defense-in-depth* -- Multiple elements of a defense-in-depth system can be compromised by common-mode failures (e.g., a specification error could cause all versions of a multi-version group of programs to fail despite their design for defense-in-depth).
- 2) *Safety margin assessment* -- This involves a mix of *probabilistic* methods (to assess the relative frequency and consequences of postulated accident scenarios and to find design weaknesses) and *deterministic* methods (to assess the response and performance of safety systems to identified accident scenarios). Applying such mathematical techniques to the discrete logic and mathematics of software can be very difficult for a number of reasons. Mathematical logic (rather than straightforward, continuous mathematical methods) must be applied to software specifications; such a process can often be more complex than the design of the software itself. In addition, the extremely large number of possible inputs, outputs, and system states makes definitive quantification of reliability very questionable.
- 3) *Environmental qualification* -- Environmental effects on software are different and less well understood than such effects on hardware, and there is no corresponding “wear-out” period for software as it ages. In fact, software can even become *more* reliable with age, since more bugs may be found and corrected. (Another difference is that once errors with software are detected, they can be removed permanently, while errors with hardware can occur again despite being fixed.)
- 4) *Requisite quality* -- Requisite quality control and quality assurance techniques (e.g., testing and documentation) are also less well understood for software systems than they are for analog, hardware-based systems.
- 5) *Failure vulnerability* -- Independence, separation, and redundancy to reduce failure vulnerability are often required to a greater degree for software than for hardware because of the less-well-understood nature of common-mode failures in software.

After analyzing these approaches to safety, the NRC investigating committee ultimately identified six key technical issues (issues 1 through 6 on the following list) and two key strategic issues (issues 7 and 8 on the following list) of concern with respect to the introduction of digital I&C into NPPs. The particular issues among these eight that are most directly related to this work will be discussed in more detail in Section 1.3.3. The eight issues are:<sup>3</sup>

1. Software quality assurance,

2. Common-mode software failure potential,
3. System aspects of digital instrumentation and control [I&C] technology,
4. Human factors and human-machine interfaces or interaction,
5. Safety and reliability assessment methods,
6. Dedication of commercial off-the-shelf [COTS] hardware and software,
7. Case-by-case licensing process, and
8. Adequacy of technical infrastructure.

Issues 1, 2, 3, 5, and 7 deal most directly with the topic of this work. Discussion of them will elaborate on the concerns in each of these areas and on possible methods for addressing them. A detailed list of specific questions pertaining to the eight issues identified by the NRC can be found in Appendix B.

### **1.3.3 Discussion of Key Digital I&C Safety Issues**

#### *1.3.3.1 Software Quality Assurance and Common-Mode Failure Potential*

Of the first issue, software quality assurance, the committee said that the “quality of software is measured in terms of its ability to perform its intended functions. This, in turn, is traced to software specifications and compliance with these specifications.”<sup>3</sup> The committee went on to say that “neither of the classic approaches of (a) controlling the software development process or (b) verifying the end-product appears to be fully satisfactory in assuring adequate quality of software, particularly for use with safety-critical systems.”<sup>3</sup>

The NRC evaluated a variety of quality assurance techniques, including various static and dynamic analysis methods as well as reliability models and reliability growth models (which, respectively, attempt to estimate and to predict reliability). There was no confidence expressed that any of these methods is sufficient to provide confidence in the required level of reliability for a nuclear reactor I&C system. ‘Static analysis’ as used here entails manual or automated inspections of such software development products as the requirements, design, source code, or test plans. Dynamic analysis, the most common example of which is program testing, involves checking the product behavior to determine if it fulfills its requirements. Testing can only cover a fraction of the total possible software states, and proper behavior of the untested states cannot simply be inferred from proper behavior of the tested states. Reliability growth models are used to predict when the mean-time-to-failure (MTTF) of software in development will be high enough to release the software. These predictions are then validated experimentally during software testing. Reliability models, on the other hand, are used to estimate a system’s MTTF after its development. Some of the MTTF estimates derived with these models have been too high to demonstrate practically, and several of their assumptions are questionable in the NRC’s opinion

(e.g., that repairs always lead to failure rate decreases and that MTTF is independent of program size despite the typical system quality of roughly constant defects per line).<sup>3</sup>

How to prevent or mitigate common-mode failures is also an issue. The USNRC staff believes this problem to be likely in digital I&C systems due to the fact that “digital I&C systems may share code, data transmission, data, and process equipment to a greater degree than analog systems.”<sup>2</sup> Thus care must be taken to prevent such sharing, which is counter to redundancy and results in severe consequences when a single (shared) component fails.<sup>2</sup> In fact, common-mode failure is the issue that the USNRC staff is most worried about and the issue on which they have focused most of their efforts.<sup>4</sup> Do analog solutions also apply to digital systems? Are diversity and redundancy the best answers for high reliability? There have been cases where the redundancy management was so complicated that it was the only source of errors in a program (e.g., during flight testing of one NASA software system).<sup>5</sup> The effectiveness and reliability of methods such as ‘N-version programming’ have also been questioned.<sup>6</sup> This programming method is based on questionable assumptions of independence, because the N different versions start from the *same* requirements. The requirements phase is the development stage to which many software design errors are ultimately traced. Two different versions of the same program could thus fail from the same error if it is a requirement or specification error. N-version programming is capable of providing at least partial independence, but it cannot be relied upon exclusively for absolute independence.

Littlewood discusses numerous models for “failure behaviour of systems involving redundancy and diversity” in a 1996 article.<sup>7</sup> In particular, she contrasts the use of redundancy with the use of diversity by comparing the Hughes model and the Eckhardt & Lee model. The Hughes model<sup>7</sup> (for hardware) is based on a probability distribution of percentage of pieces of (the same) hardware that will fail for each different stressful environment. This model attributes component failure dependencies to “variation in the stressfulness of operating environments.” A stressful environment will have the same effect (that is, an increased likelihood of failure) on every component. Redundancy (more than one of the same component) is the answer to this problem, since the more of the components there are, the less likely it is that they will *all* fail. The Eckhardt & Lee<sup>7</sup> model (for software), on the other hand, attributes failure dependencies in “design-diverse software” to the varying “intrinsic difficulty” of the inputs the software receives. Diversity (different versions of the same program) is the answer to this problem; one software version may fail on some particular difficult input, while another may not. Some empirical experiments have shown, however, that a major factor in software failure dependencies is the fact that different program versions sometimes have identical faults, often due to errors within the specifications.<sup>7</sup>

### 1.3.3.2 System Aspects of Digital I&C Technology

The NRC also believes that focusing on *system* aspects of digital systems (as opposed to focusing on particular components or parts) is important in order to maintain operation within an adequate margin of safety. For example, it is possible that each module or component in a system could function within its

own margin of safety, while the overall function of the *system* of modules falls outside the total margin of safety for the system. Important aspects of digital systems and their interfaces were suggested for close consideration, including sequential operation (i.e., control sequences must be faster than response times or the responses will no longer be valid by the time they are processed), multiplexing, memory sharing (i.e., this must not lead to corruption of data), data transmission methods, and storage media. Designing a new system with these factors in mind is fairly straightforward, but retrofitting into an existing plant can be much more difficult.<sup>3</sup>

Reference 8 discusses the use of HAZOP, short for Hazard and Operability Study, as one suggested approach to the issue of system safety. This is a technique that originated in the chemical industry for the identification and analysis of hazards. Though designed with the chemical industry in mind, it has generic aspects which are useful to apply to software development.<sup>8</sup>

Many of the methods for ensuring high software reliability that are covered in this work involve looking specifically at code or other components of the software (and the system of which it is a part) to determine what could go wrong with them and figure out how to prevent it or compensate for it. HAZOP, on the other hand, looks not at these design components but at the physical or logical interconnections between them.<sup>8</sup>

As described in the HAZOP article by Redmill et al.<sup>9</sup>, an interconnection “represents the flow of some entity (for example a fluid in a chemical plant, data in a computer system) between the components which it connects.” These entities have characteristics that depend on the particular situation (see the “experimental frame” discussion in Appendix P), and the characteristics have intended “design values.” HAZOPs seek to find “what deviations from the design might occur, and then to determine their possible causes and hazardous effects.” This search is done via “guide words” which help to narrow the scope of the search to certain classes of deviations. The meaning of the guide words can change with varying situational contexts, leading to multiple “attitude-guide-word interpretation” possibilities. Questions raised during the HAZOP study must lead to recommendations for action. The documented output of a HAZOP, according to Redmill et al., should include:<sup>9</sup>

1. The objectives of the study,
2. The interconnection attributes evaluated, guide words used, and attribute-guide-word interpretations,
3. Details of the hazards identified,
4. Any features existing in the design for the detection, elimination, or mitigation of the hazards,
5. Recommendations, based on the [HAZOP] team members’ expert knowledge, for the elimination or mitigation of the hazards or their effects,

6. Recommendations for the study of uncertainties surrounding the causes or consequences of identified hazards,
7. Questions to be answered, as follow-up work, regarding uncertainties about whether a hazard could occur, and
8. Operational problems to be investigated for possible further hazards.

Also pointed out are benefits of using HAZOP in combination with a Failure Modes and Effects Analysis (FMEA). HAZOPs search for both the cause of a deviation from a design intent and the possible results of the deviation. FMEAs search strictly for consequences (in the system) given a failure (of a component). At the very start of the software development process, when only high level system information is known, a HAZOP might be applicable for investigating the interfaces between the system and the environment (or other systems). As component information becomes available, a preliminary FMEA can be done to find non-interaction hazards and to guide further hazard searches. With more detailed information, both types of analyses can be useful. HAZOP may produce results that can be further analyzed with a FMEA and vice versa. At high levels of detail (a *low* level in the system), a HAZOP can become very tedious, so a FMEA is a better option.<sup>9</sup>

In the book *Safeware*, Leveson addresses many of the problems of and possible solutions to system safety from the perspective of software.<sup>10</sup> In a study of the Galileo/Voyager space program, safety-related software errors were found to stem primarily from system requirements, documented specifications, and problems with the software interfaces to the rest of the system. Leveson suggests that the standard generic methods to deal with problems such as these are ineffective: Making the requirements and code “correct” (completely) is impractical, and using fault-tolerant techniques has limited effects only on coding errors. Emphasis should instead be on *system* safety concepts such as requirements/code analysis, hardware or software interlocks, fail-safe systems, and system self-monitoring.<sup>10</sup>

Leveson details a long series of “completeness criteria” to be used when evaluating system requirements. These criteria cover such topics as deterministic matching of input with output, proper system initialization and updating, specified system behavior during any “off-normal” situation, handling of various input arrival rates and response times, and proper handling of hazard situations (e.g., fail-safety or fault-tolerance).<sup>10</sup> (See Appendix C for a more detailed sample list of these criteria.)

Lutz of NASA’s Jet Propulsion Laboratory has developed a “Safety Checklist” which stresses a deterministic approach for finding software errors (the same type of approach endorsed by the USNRC in BTP-14 via Reg. Guide 1.152, and IEEE 7-4.3.2).<sup>11</sup> The checklist is a series of “informal, natural-language” translations of formal design criteria that target the particularly troublesome software design areas of interface specifications and system robustness. It covers such issues as deterministic software behavior, responses to early or late inputs/out-of-range inputs/varying input arrival rates, and proper handling of errors and hazards (including predictable “performance degradation” when necessary).<sup>11</sup> As

can be seen, there is some overlap of this list with that proposed by Leveson. Both of the lists deal with such issues as:

- \* The software's ability to deal with any manner of input (e.g., early, late, non-existent, or out-of-range, excessive or insufficient arrival rate),
- \* Deterministic software behavior (based on the inputs and software state),
- \* The existence of unused sensor information or unreachable software states (possibly indicating superfluous -- or missing -- functionality),
- \* The presence of at least one path to a safe or low-risk state from all hazardous states, and
- \* The capability for smooth, predictable performance degradation when that is the chosen error response (e.g., sacrificing some accuracy in order to decrease response time).

(For the complete "Safety Checklist," see Appendix D.)

The guidance of the Safety Checklist is driven by the same principles that drive BTP-14. They are both concerned with verifying deterministic measures of the software performance, but the checklist is somewhat more explicit in how to make such assessments. For example, question #10 on the list starts generally ("Can input that is received before startup, while offline, or after shutdown influence the software's behavior?") and then provides specific examples of how it might be manifested in a program to guide the user ("For example, are the values of any counters, timers, or signals retained in software or hardware during shutdown? If so, is the earliest or most-recent value retained?").<sup>11</sup>

It is hard to determine at what point a list such as this is sufficient. Could this list be combined with the requirements-targeted questions Leveson poses (discussed above)? Are the lists exclusive? Are they complementary? A realistic solution might be to use a combination of available possibilities. Combining multiple available technologies is a strategy favored in the software development standard used by Ontario Hydro, which uses a variety of software development techniques, each with their own merits and weaknesses, to strengthen and increase confidence in each other.

### *1.3.3.3 Safety and Reliability Assessment and Case-by-Case Licensing*

The NRC feels that there must be agreed-upon methods of software safety and reliability assessment in order to evaluate safety margins, compare performance with regulatory criteria (including *quantitative* safety goals), and make safety trade-off comparisons (e.g., improved self-testing capability at the cost of increased complexity to achieve it). Many current safety and reliability methods were not designed with digital applications in mind. It has been suggested that perhaps a shift from statistical methods to mathematical proof and logic techniques would be appropriate. The NRC suggests that new methods should combine deterministic and probabilistic aspects and should stand up to reasonable empirical testing. They should be acceptable to the appropriate experts and understandable and believable



to the public; and they should be consistent across applications, yet adaptable enough to keep up with rapidly changing technology.<sup>3</sup>

Various methods have been proposed to improve the safety-critical software development process. NUREG/CR-6263 offers a comprehensive list of “candidate guidelines” based on current software industry practice and standards to aid in software life cycle review and assurance activities. It also addresses areas where more research is needed, such as:<sup>12</sup>

1. Development of review criteria based on NPP software domain analyses
2. Identification of proven software designs that fulfill system safety
3. Better definition and measurement of reliability
4. Common notation for translation from system-level to software-level requirements
5. Common domain-related software safety performance requirements
6. An empirical study of software failures in high-integrity systems
7. Identification of successful fault-tolerant and error-handling techniques, and
8. Criteria for evaluating adequacy of path coverage in software testing.

The candidate guidelines offered by NUREG/CR-6263 cover a wide array of topics. One important topic is the Software Requirements Specification (SRS). It is recommended that the SRS should specify what abnormal software conditions or events (ACEs) the software should be able to detect, and that it should specify the time-dependent input-output relations the software must implement not only for valid inputs and conditions but also for invalid inputs and ACEs.<sup>12</sup>

Numerous guidelines are offered with respect to software safety, such as the following:<sup>12</sup>

- \* It is stated that software requirements, designs, and safety-critical code “essential to the safety function should be analyzed to identify ACEs that could prevent the software...from meeting all software safety requirements...” (Guidelines 9-5 through 9-7).
- \* Also, “test cases should [include and] 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” (from Guidelines 7-18 & 9-8). In addition, the “reliability of the software should be demonstrated using a reliability demonstration test.” “A description of the operational profile used should be provided [including] a list of assumptions made in constructing the profile and all information necessary to enable verification” (from Guideline 7-22).
- \* It is mentioned in the related research interests that it would be useful to “identify common software-related ACEs or hazards based on domain-specific experience.” Many people have expressed the need for a common database of nuclear-related

accidents, mishaps, and related data (information from all high-integrity/safety-oriented systems and fields would be valuable). The proprietary nature of such information continues to be a hindrance to these efforts.

In addition to the areas of necessary research described by NUREG/CR-6263 above, Littlewood has proposed a “list of areas where technical work is required,” many of which are major topics of concern within the BTP-14 review process (and the associated Standard Review Plan). It is interesting to note that Littlewood is admittedly “pessimistic about the outcome” of these topics, which include:<sup>13</sup>

1. Real statistical evidence for the relationship between processes and product attributes.
2. Formal quantification of expert judgment, combination of expert judgments, calibration, etc.
3. Composition of evidence from disparate sources, such as proof, statistics, judgment, and others.
4. Quantitative theory to replace the present qualitative “claim limits.”
5. Better probabilistic modeling of “structured” software.
6. Accelerated testing.
7. Better data: experiments, case studies and mandatory reporting requirements for safety-critical systems in operational use.
8. Standards: better scientific basis, studies of efficacy, and so forth.

Littlewood discussed these thoughts at a two-day workshop in 1993 on the development of safe software. This workshop involved twelve experts from the nuclear and computer software industries (four panelists, eight observers from the USNRC and Lawrence Livermore National Laboratory). The observers drew general as well as specific conclusions on the topics of diversity, reliability and testing, failure modes, and other subjects during analysis of the panel discussions that were held. The following paragraphs describe some of their more relevant conclusions on the respective (italicized) topics.<sup>13</sup>

*General.* Software safety must be achieved with a *variety of methods in tandem* -- for example, testing, formal methods, expert judgment, etc. In addition, there must be systematic methods for combining the information from these different sources. Hazard analysis must be included in this process, at both the system and software levels.<sup>13</sup>

*Diversity.* This can be helpful when introduced into the design as early in the process as possible - for example, by backing up a software component with a hardware component that executes the same function. ‘N-version programming’ was not judged to be particularly helpful. It can even become a major source of failure if overused.<sup>13</sup>

*Reliability and Testing.* Testing can demonstrate reliability only to the level of approximately  $10^{-4}$  failures-per-demand. If higher reliability levels are required, other methods must be used to extend the claims. In addition, to gain more knowledge and confidence, failure data must be kept in some organized fashion for operational systems. Interestingly, doubt was expressed as to the merit of using FMEAs or FMECAs ('Failure Modes, Effects, and Consequences Analyses') for software; these methods were felt to be more appropriate for an overall system. It was also felt that software can have too many states to reasonably use these methods.<sup>13</sup>

*Formal Methods.* Much confidence was expressed in the ability of the rigor of formal methods (and some less rigorous methods) to improve the software design process. It was felt that their rigor could greatly increase individuals' understanding of software, and in so doing that they could lead to a better ability to scrutinize software requirements and to a reduction of software design complexity.<sup>13</sup>

During the workshop, Leveson (one of the panelists) discussed various "myths" related to the field of software safety. One is that increasing software reliability, as is the focus of much of the literature surveyed and mentioned previously, will increase safety. This is not necessarily true, since errors removed may be unrelated to safety; software can be correct and 100% reliable, but still be unsafe due to poor specifications. Another "myth" is that testing or 'proving' software correct (with formal verification) can remove all errors. In addition, specifying 'correct' behavior in a formal, mathematical language can be as difficult and error-prone as writing code, and it cannot account for some problems (e.g., timing/overload problems).<sup>13</sup>

Safety and reliability assessment are part of the licensing process, which the NRC also questioned. They raised questions as to how to make the NPP regulatory process more efficient, effective, and flexible enough to suit quickly changing digital I&C technology. Digital upgrades and retrofits can generally be licensed fairly quickly and easily, but not if the change involves an "unreviewed safety question," in which case the licensing process is much more uncertain in terms of time and money. In addition, as the process is now, by the time a design receives approval, the technology it uses may already be out-of-date.<sup>3</sup>

#### 1.3.4 Process vs. Product and Cleanroom Engineering

BTP-14's focus on the process rather than the product is not a totally new concept. This is an idea that is used in the manufacturing world in the 'Cleanroom engineering' process and its reliability improvement methodology (referred to as the "Qualified Manufacturer's Listing").<sup>14</sup> A discussion of Cleanroom engineering will help to clarify the difference between these two approaches.

Cleanroom engineering seeks to manufacture products correctly from the start of the design process, contrary to the traditional method of sampling finished products to find mistakes and then going back through the production process to fix them. Engineers using the Cleanroom process seek to determine what the *root* causes of problems are rather than simply to find the *direct* or immediate causes.

This is more efficient, in the long run, because not only will the direct causes be fixed, but any causes *similar* to the direct cause will also be found and prevented from causing problems later.<sup>15</sup>

An example of fixing only the direct cause of a problem would be recalibrating a piece of equipment that is creating parts slightly outside some size limit tolerance. Fixing the root cause might involve determining *why* the piece of equipment is working improperly (e.g., faulty maintenance or improper usage by the operators, both of which could result from inadequate training) and then making changes to fix this weakness in the process. In software engineering, the difference between fixing only a direct cause and fixing a root cause might be the difference between simply rewriting a piece of faulty code versus determining the reason or the specific error underlying the fault (e.g., an ambiguous specification that is misinterpreted by a programmer who is not familiar with nuclear engineering or NPP operation) in order to avoid it in the future. A particular benefit of fixing the root cause of an error in software is that, once a software error is removed, it will be gone *permanently*. This is an important difference from hardware. It must be said that typically the failure causes in the manufacturing industry are known entities, while with software the failure causes are often new factors that have not been thought of or considered previously. However, the mindset of using error detection information to *improve* reliability (by fixing root causes) rather than simply to *predict* reliability is valuable.

#### 1.3.4.1 Error Prevention, NOT Error Correction

Cleanroom software (or system) engineering is a well-documented approach to software development that has been maturing since the 1980's. This approach to software design is based on the Cleanroom approach in industry and manufacturing, which "achieves reliability through elimination of all product weakness at the design phase of the product" rather than by elimination of errors found during testing.<sup>15</sup> The closest that BTP-14 comes to this is in its initial Software Safety Plan, which includes a requirement for safety analyses (aimed at error prevention) at each development stage. (These safety analyses are described in Appendix E.)

The Qualified Manufacturer's Listing bears a strong similarity to the USNRC's software evaluation process proposed in BTP-14 in that it emphasizes documentation. It also monitors and controls development process steps in order to find "critical failure mechanisms."<sup>14</sup> Tests are done *only* at critical points and are much more efficient than tests placed randomly throughout the process.

One tenet which forms the basis of the Cleanroom approach is that, as described in a May 2, 1996, memo from Dr. Donald W. Miller, Chairman, Instrumentation and Control Systems and Computers Subcommittee of the Advisory Committee on Reactor Safeguards (ACRS) to other ACRS members, "a significant fraction of errors in software systems occur in the requirements and design stages of the software life cycle."<sup>16</sup> This is why Cleanroom software design emphasizes carrying out the process more strictly during the initial design phases rather than during the later testing phases.

#### 1.3.4.2 Debugging

Cleanroom software engineering assumes that most well trained, capable software designers can in fact design very good software from the beginning of the design process on their first attempt. Mills, the original developer and proponent of the Cleanroom software development process at IBM, claims that box structures create a “completeness and precision” in software design that fosters human design capability.<sup>17</sup> The software created for the U.S. census in 1980, consisting of 25,000 lines of code and operating on twenty miniprocessors nationwide, never experienced an observed error. There are other such examples of very well designed software too, such as the software for IBM wheelwriter typewriters (no failures ever detected since the software came out in 1984). The designers of these software products all used functional verification to achieve stepwise refinement of their software.<sup>15</sup>

It is the notion of high designer ability that makes it possible for designers to go straight from specifications to design without unit testing or debugging, instead waiting until the system level to test. In fact, in Cleanroom engineering there are separate teams of software developers for specification, development, and certification. In other words, the people who actually write a program do not debug it at all. The certification team finds the errors and sends them back to the development team to be fixed.<sup>17</sup>

No one assumes that software designers will make *no* mistakes, but Cleanroom supporters do believe that the mistakes software designers would make in attempting to debug a program during the design and coding phase could have complex and unforeseen effects that would be more difficult to fix later on than the comparatively simple mistakes they make while programming. Estimates show that debugging introduces new errors into a program approximately fifteen percent of the time. Debugging is not used in Cleanroom software design because it is typically done with the goal of producing a correct *output* rather than a correct *function*, which should be the goal of the programmer.<sup>17</sup> Simple programming mistakes can be caught by the testing team once an entire first version of the software product has been completed. Another methodology that can complement the Cleanroom approach of error prevention rather than debugging is the use of CASE (Computer-Aided Software Engineering) tools to automate the requirements analysis and design specification phases and, potentially, the coding phase of the software development life cycle. These tools allow a software designer to work graphically with a software schematic, automating much of the manual work that is normally done and minimizing the chance for inadvertent errors (e.g., incomplete requirements specifications or inaccurate design specifications). In addition, they can automatically generate code, saving time not only in code implementation but also in unit testing and integration (since there is more confidence in code that has been generated based on generally tested and accepted formats).<sup>18</sup>

#### 1.3.4.3 Simplicity

The ACRS wants simplicity to be stressed more strongly in regard to reactor safety control software. They feel that such systems “should be simple and separated from all other control systems

which may be significantly more complex.”<sup>16</sup> Extra (unnecessary) functionality simply raises the complexity level and, thus, the chance for error. One of the benefits realized from Cleanroom projects already done is the simplicity of the software created. The designers tend to be conservative in light of the scrutiny the Cleanroom design process receives and because of the need to actually demonstrate the correctness of their work. Dyer, a Cleanroom expert, says that “a reasonable rule of thumb is that when reading a design becomes difficult and proof arguments are not obvious, it is probably time to think harder about the design and come up with a simpler, more valuable design.”<sup>15</sup>

#### 1.3.4.4 *Functional (Correctness) Verification*

Cleanroom software developers use mathematical proofs to achieve functional (or correctness) verification of software parts, then refine the units stepwise until assembling them into a complete program. Functional verification, which consists of a limited and well-defined set of verification-based inspections, is based on a view of a program being a “rule for a mathematical function, mapping all possible input states into final states.”<sup>17</sup>

While this sounds ideal theoretically, the practical merit of it is questionable. If “all possible” input states are mapped to output states, that would indicate that we have accounted for all possible scenarios and can therefore be confident that no unforeseen situations will arise (if the implementation of our requirements was done properly). We are concerned, however, that in writing our requirements and specifications and turning them into a program, we have not accounted for every possible input state. After all, many accidents and disasters that have occurred and continue to occur are due to unforeseen circumstances (e.g., strange combinations of factors that are acceptable individually but are dangerous when combined).

In fact, according to Hecht, one of the biggest sources of software failures is rare events. The need to test for such conditions was also described in the discussion of NUREG/CR-6263 (Guideline 9-8). In a study of the Space Shuttle software, rare events were the leading cause of the most serious failure categories. One rare event could almost always be handled; two were very difficult to handle; and three virtually guaranteed failure.<sup>19</sup> It is apparent that the more complex a program is, the more difficult it is to assure that strange interactions of rare events will not occur.

To make up for this weakness, the use of software fault tree analysis (FTA) *in addition to functional verification* has been proposed for the purpose of program safety verification. FTA starts at the system (not software) safety specifications and assumes few hazardous states. It moves forward from the requirements to the actual program, using proof by contradiction to show that unsafe system states (whether they are correct or incorrect states) are not reachable via the program code.<sup>20</sup> Therefore, FTA can help in showing that the program does *not* do what it is not supposed to do. This indicates the potential of the Dynamic Flowgraph Methodology (DFM) for our project as well, since DFM is used to construct fault trees which identify paths leading to critical system events. DFM techniques can thus be used to single out

particular problem spots within a program or system, so that efforts to fix the problems (e.g., testing) can then be focused on a much more narrow target.<sup>21</sup>

#### 1.3.4.5 Testing and Statistical Quality Control

Debugging is not the only traditional element of software design that is left out of the Cleanroom process. The traditional coverage testing methods (of each branch within a program or each complete path through a program) are replaced with statistical usage testing.<sup>17</sup> Some of the key problems with traditional software testing, as opposed to statistical usage testing (which originates from the hardware community), are identified by Levendel, head of a system verification department at AT&T Bell Laboratories, in an article he wrote for *IEEE Software*.<sup>22</sup> These problems include, among others:

- \* The lack of a clear testing and “hand-over” process between *each* development phase (partly due to the large amount of overlapping of phases), and
- \* The use of testing almost exclusively for debugging rather than for estimating product quality to aid in process control

Statistical usage testing is based on the premise that for most programs of substantial size, complete testing of every case cannot be done, so the testing that is done is based statistically on usage experience and expectations. (Some programs are simple enough that complete testing can in fact be accomplished. The concern here, however, is with programs that are not so simple and are difficult or impossible to test completely.) While coverage testing is theoretically more complete than statistical usage testing, it is difficult in practice and is much less effective than usage testing, which focuses on high failure-rate errors. Adams (1980) illustrated with data from an evaluation of nine IBM products which showed that fixes stemming from usage testing caused an approximately thirty-times greater increase in mean-time-to-failure per correction, on average, than fixes stemming from coverage testing.<sup>17</sup>

Of the standard functional and structural testing types, only functional testing (to verify correct implementation of specifications) is retained “as is” in the Cleanroom process. Structural testing is replaced by the correctness verification already described. If errors are found they are sent back through the development and certification process. When the confidence in error-free software reaches a high enough level, it is released. With continuing error-free usage, confidence can climb even higher. This whole process achieves ‘statistical quality control’.<sup>17</sup>

It is the opinion of many people involved with quality control in industry that statistics is the only tool that can lead to zero-defect level quality. Skeptics of SQC claim that statistical methods cannot be applied to the deterministic functioning of computers, but supporters counter that the *usage* of computers (as opposed to the computer functioning) is statistical.<sup>17</sup>

Numerous ideas have been proposed that might supplement the SQC process used in Cleanroom engineering. Levendel proposes comparing test results to specified quality criteria in order to determine whether to fix particular problems or proceed with development.<sup>22</sup> He further proposes a method to reduce

system complexity and seek out “design holes,” areas of design incompleteness where errors are likely to congregate. These design holes result from the inability of designers to fully comprehend the complexity of the system and its requirements. Levendel reiterates the idea that when designers try to account for this weakness in the usual way (program debugging), it leads to even more complex interactions which the designers cannot always foresee. He also emphasizes that tests must be designed with actual system usage profiles in mind and the tests must be run enough times for *statistical* confidence.<sup>22</sup>

Dyer advocates achieving statistical assurance of software reliability by compiling test failure data for successive software increments (versions of the same program), extrapolating the data to predict the MTTF for each increment, and then weighting the results for each increment to determine an overall system MTTF.<sup>15</sup> Laprie also calls for the inclusion of historical failure data (much like the use of increments) rather than just the present software version’s data to determine reliability. He terms this the “product-in-a-process” approach. This approach seeks to find the conditional probability of failure of software upon execution given the failure data from previous operational versions of the software. There is an assumption inherent to this methodology that different generations of the same software product are similar. Laprie says that this method can be applied to an entire software system or to individual parts and then combined. He claims that this adds the elusive quantitative nature to the otherwise qualitative focus of software development process evaluation.<sup>23</sup>

### 1.3.5 Fundamental Software Design Questions

The two most fundamental questions which must be answered in order for software for safety-critical applications to be licensed are: 1) How do we determine that the system does what it is supposed to do?, and 2) How do we determine that the system does *not do* what it is *not supposed* to do? The first question has historically been dealt with in much more depth than the second question, as software is usually subjected to regimented testing in order to determine if it does what it is supposed to do.

It is important to remember, however, that testing can only provide *confidence* that an error is *absent*; it cannot prove it. In other words, testing cannot *prove* that a program will not do what it is not supposed to do. In fact, program testing cannot even prove, except for trivially simple programs, that a program *will* correctly do everything that we *do* want it to do. The USNRC’s proposed software review (and development) methodology focuses on the software development process in addition to (testing of) the developed software product. Considering the *entire* development process, while not foolproof either, provides more confidence that all aspects of the software’s behavior (i.e., both behavior required of the software *and* behavior from which it is prohibited) have been accounted for and designed into the software at each step of its development.

### 1.3.6 The USNRC’s Approach to the Software Development Process

BTP-14 is a standard used in the examination of software and its development process. Evaluations are conducted for the software life cycle *planning*, *implementation*, and *design outputs*



(‘design output’ is the name for any of the documentation that is produced throughout the eight stages of the software development life cycle -- planning, requirements, design, implementation, integration, validation, installation, and operations & maintenance). The vehicle for analyzing all of these phases is documentation: development project planning documents for the planning phase, ‘process documents’ for the implementation phase, and the ‘design outputs’ themselves for the design output phase.<sup>2</sup>

BTP-14 is part of a larger regulatory document called the Standard Review Plan (SRP). The section of the SRP pertinent to software dependability in digital I&C systems is Chapter 7, “Instrumentation and Controls,” which provides guidance for reviewing I&C portions of applications for nuclear reactor licenses or permits (or amendments to them) and related topical reports. Section 7.0 of SRP Chapter 7 gives an overview of the I&C review process. SRP Appendix 7.0-A gives an overview of the review process with regard to the unique digital issues involved with I&C systems. Section 7.1 gives an overview of the regulatory bases for the SRP review process. The purpose of the BTPs, is to provide enough guidance so that a reviewer with any amount of domain background (in this case, the nuclear domain) can carry out an effective review of the corresponding BTP topic (in this case, the software development process for software used in NPP digital I&C systems).

The BTP and the SRP are the major documents discussed in this work, but there are numerous other supporting documents that are also used throughout the software review process. Please note that throughout the discussion and evaluation of these various documents, individual references will not be made every time one of the documents is mentioned. For convenience, however, there is a list of the most commonly referenced documents (including the abbreviated names used for them in this work) on pages 10-11.



---

## 2 Software Review

---

### 2.1 The BTP-14 Review Process

The easiest way to explain what the term ‘software development process’ means in the context of the USNRC’s proposed software review process is to follow through the review process from beginning to end, because the requirements that the reviewer must review are obviously the requirements that the developer must design into the software during its development. Although the process is too complicated to describe every activity in it completely, detail will be used in some key examples to illustrate the complex nature of this regulatory process.

The purpose of this section is to develop an orderly description of the software development review process. However, it is difficult to discern from all of the documentation involved an ordered, logical procedure. Obviously, software development is not a linear process, but, theoretically, at least the review process for software development should be fairly well defined. Certainly, some degree of backtracking and cross-checking within the review process is inevitable (just as it is during the development process), but at least the overall review *framework* should be well structured. (All of the documents referenced in the following software review synopsis will be typed in boldface print where discussion of them begins. Individual references will not be given, as the information here comes from the corresponding documents being discussed.)

#### 2.1.1 BTP-14

BTP-14 starts off with background material on a variety of topics. These topics include the BTP’s regulatory basis (i.e., from what laws and guidance it is derived), some basic definitions, and its overall goal, which is to provide “guidelines for evaluating software life cycle processes for digital computer-based instrumentation and control [I&C] systems.”

In the very first paragraph of the background material comes the first reference to another document, when it says that “the structure of this BTP is derived from the review process described in Appendix 7.0-A [of the SRP].” This process is not described directly in the BTP, but it is necessary to be familiar with it and with the rest of SRP Appendix 7.0-A in order to understand how to undertake the BTP software review process in the larger context of the overall digital I&C system review.

#### 2.1.2 SRP Appendix 7.0-A

The first introductory paragraph of SRP Appendix 7.0-A continues the referencing to other regulatory documents. It says that Appendix 7.0-A “provides an overview of the process for reviewing the

unique aspects of digital instrumentation and control (I&C) systems. It supplements.....(1) the overall I&C system design described in Section 7.0 [of the SRP], (2) [and] the design criteria and commitments described in Section 7.1.....” It goes on to say that “more detailed information on the regulatory bases, acceptance criteria, and review processes for specific issues are described in Section 7.1, related [BTPs], and regulatory guides.” Therefore we must also understand SRP Sections 7.0 and 7.1 (and the regulatory guides) in order to understand Appendix 7.0-A, all of which we need to understand in order to correctly use BTP-14.

### 2.1.3 SRP Section 7.0

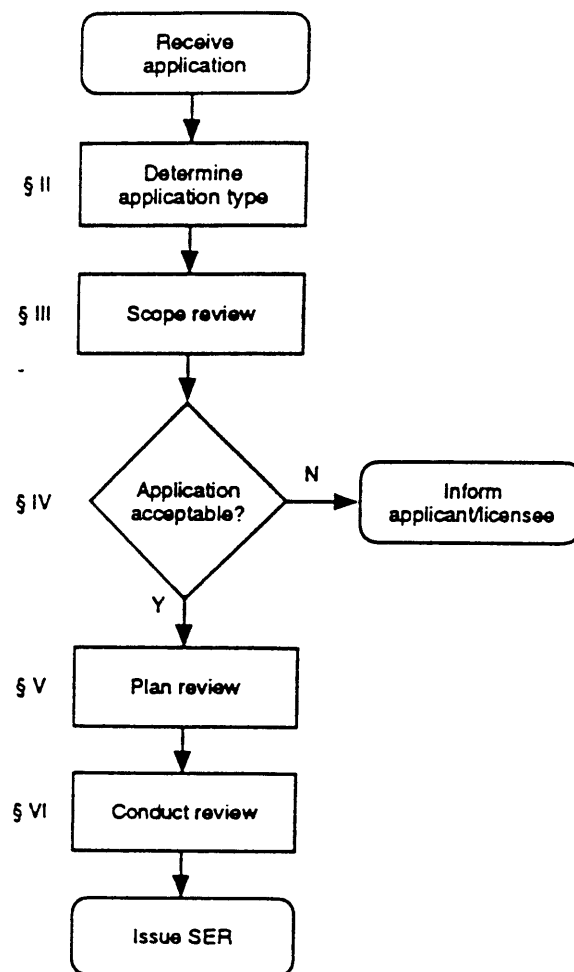
The overall purpose of SRP Section 7.0 is to give an overview of the review process, (illustrated in Figure 1) for I&C portions of reactor license applications and “generic safety evaluations of specific topics.” The first step according to SRP 7.0, after determining the application type, is to review the scope and contents of the application in order to determine the necessary extent of the review and the resources needed for it. (The scope and content points to consider for each type of application are explained in Appendix F.)

After the reviewer determines that the information contained in the application is sufficient for the review to progress (by consulting such other documents as Regulatory Guide 1.70 -- “Standard Format and Content of Safety Analysis Reports for Nuclear Power Plants,” appropriate SRP sections or BTPs, and/or SRP Appendix 7.0-A), he should create an application-specific review plan to provide the applicant with an appropriate schedule and all necessary information (e.g., specific review criteria) to prepare for the review. Finally, the review should be carried out “using the acceptance criteria and review processes of the SRP.” This concludes the information from SRP Section 7.0 that is necessary for defining the software development process.

### 2.1.4 SRP Section 7.1

This leads us to SRP Section 7.1. Section 7.1 discusses in-depth the types of I&C systems important to safety (e.g., protection systems, emergency safety feature actuation systems, interlock systems, diverse actuation systems, etc.) that are considered by the SRP as well as various acceptance criteria and guidelines and their applicability. Some of the other important topics addressed by this section include V&V, reliability, and testability.

Section 7.1 also describes the basis of certain rules for I&C systems important to safety. The rule origins are found in IEEE 603 (a modified version of IEEE 279), Reg. Guide 1.153, 10 CFR 50, and various GDC from Appendix A of 10 CFR 50. *Official* I&C system acceptance criteria come from the “technical requirements of 10 CFR 50 including IEEE 279 and the GDC.” Guidance for ways of



**Figure 1** Overview of Review Process  
(From SRP Section 7.0, p. 7.0-9)

conforming to the requirements is provided in various industry codes and standards and the regulatory guides that endorse them.

In addition, Section 7.1 provides seven clarifications for *digital* systems in the following review areas (from Regulatory Guide 1.152 and IEEE 7-4.3.2):

1. *Electromagnetic capability* (in light of the range of environmental electromagnetic conditions that could be experienced, e.g. lightning)
2. *Computer system quality* (in terms of software development and hardware/software integration; qualification of existing commercial computers and pre-existing software products; use of software tools; V&V; and Software Configuration Management)
3. *Equipment qualification*
4. *System integrity* (to maintain functional capabilities under extreme conditions; may include test and calibration capabilities in the system to detect failures)
5. *Communications independence* (i.e., of safety systems from non-safety systems)
6. *Reliability* (purely quantitative measures cannot be used when dealing with software), and
7. *Defense against common-mode failures*. (GDC-21 is a particularly important criterion in relation to common-mode failures. It mandates that there must be “high functional reliability and inservice testability” and “redundancy and independence...to assure that (1) no single failure results in loss of the protection function, and (2) removal from service of any component or channel does not result in loss of the required minimum redundancy unless the acceptable reliability...can be otherwise demonstrated.”)

(For more detailed information on the clarifications in these areas, see Appendix G.)

Compliance with all of these topics is assessed by the review process of SRP Section 7.1. First an analysis must be done to ensure that all safety-related I&C systems necessary to comply with 10 CFR 50 are addressed and that all the appropriate acceptance criteria and acceptance guidelines for them, as listed in Appendix 7.1-A, are identified. Exceptions taken to the (non-mandatory) acceptance guidelines are reviewed to determine their acceptability and to ensure that they do not violate any of the (mandatory) acceptance criteria. Items that have been approved by the USNRC staff previously should be identified (and modifications should be noted for review), while completely new technologies should be identified so that an acceptance basis may be developed for them. Any digital equipment reviewed must be evaluated with the seven digital clarifications listed above in mind. A review of proposed solutions to any ‘unresolved safety issues’ must also be done. Guidance for reviewing such issues is found in Appendix 7.1-A and a related NUREG document (NUREG-0933).

The ‘Implementation’ instructions for Section 7.1 say that “implementation schedules for conformance to parts of the method discussed herein are contained in the referenced regulatory guides.”

Again the user is left with the feeling that many of the important details of this part of the regulatory process are not in this document itself, but scattered throughout many other documents. The searching and cross-referencing required to do all of this makes the process seem quite a bit more disorganized and tedious than it should be.

#### **2.1.5 SRP Appendix 7.1-A**

Before we return to Appendix 7.0-A, we must consider Appendices A, B, and C to Section 7.1. SRP Appendix 7.1-A, “Acceptance Criteria and Guidelines for Instrumentation and Control Systems Important to Safety,” gives a more detailed description of the acceptance criteria and guidelines mentioned above, which are divided into four major categories: (1) regulations from 10 CFR 50 & 52 (including ANSI/IEEE Std 279, which is paragraph 50.55a(h) of 10 CFR 50), (2) the General Design Criteria (GDC) of 10 CFR 50 Appendix A (“General Design Criteria for Nuclear Power Plants”), (3) regulatory guides (‘Reg. Guides’) and their accompanying endorsed industry codes and standards, and (4) branch technical positions (BTPs). The first two of these categories provide mandatory requirements, while the second two provide non-mandatory suggestions of how to meet the requirements. (See Appendix H for some detailed examples of guidance from SRP 7.1-A.)

#### **2.1.6 SRP Appendix 7.1-B**

SRP Appendix 7.1-B, “Guidance for Evaluation of Conformance to ANSI/IEEE Std 279,” provides discussion of the requirements for reactor protection systems, specifically in regard to the Reactor Trip System and the Emergency Safety Feature Actuation System. (Appendix 7.1-C extends application of this section to systems that are important to safety but are not so crucial as to be called “safety systems.”) The appendix begins with a large section based on Section 3 of IEEE 279, which concerns the constituents of the “design basis” for protection systems. This is followed by a variety of more detailed sections based on Section 4 of IEEE 279, which covers particular topics of concern for reactor protection systems. (See Appendix I for some specific examples of guidance from SRP 7.1-B.)

#### **2.1.7 SRP Appendix 7.1-C**

SRP Appendix 7.1-C is summed up by its title: “Guidance for Evaluation of Conformance to IEEE Std 603.” IEEE 603, as endorsed by Reg. Guide 1.153, “Criteria for Power, Instrumentation, and Control Portions of Safety Systems,” has superseded IEEE 279. It is supplemented by IEEE 7-4.3.2 (and its Reg. Guide, 1.152) for application to digital topics. References to IEEE 603 are automatically intended to include IEEE 7-4.3.2 and Reg. Guides 1.152 and 1.153. This appendix applies to the safety systems covered in SRP Sections 7.2 through 7.6 as well as to any system requiring high functional reliability and it includes an accompanying chart to show specific matching of guidance sections with appropriate systems. The guidance in this appendix is practically the same as that of Appendix 7.1-B.

At this point, depending on the particular item being reviewed, the reviewer would probably also need to consult numerous Reg. Guides and other standards. The most pertinent ones to the software review process are Reg. Guide 1.152 (which endorses IEEE 7-4.3.2) and Reg. Guides 1.1uu through 1.1zz (which endorse various IEEE standards and are designed specifically for use with Section 7 of the SRP). A description of the place of these documents in the software review process will be given in Sections 2.3.5 and 2.3.6.

### 2.1.8 Back to SRP Appendix 7.0-A

Now we are ready to return to SRP Appendix 7.0-A. The previous point of departure from this document was in the introductory paragraph, which explains the purpose of this appendix. Appendix 7.0-A is used to supplement the description of the review processes for the overall I&C system (SRP 7.0), for its design criteria and commitments (SRP 7.1), and for the individual digital I&C systems (SRP 7.2-7.9).

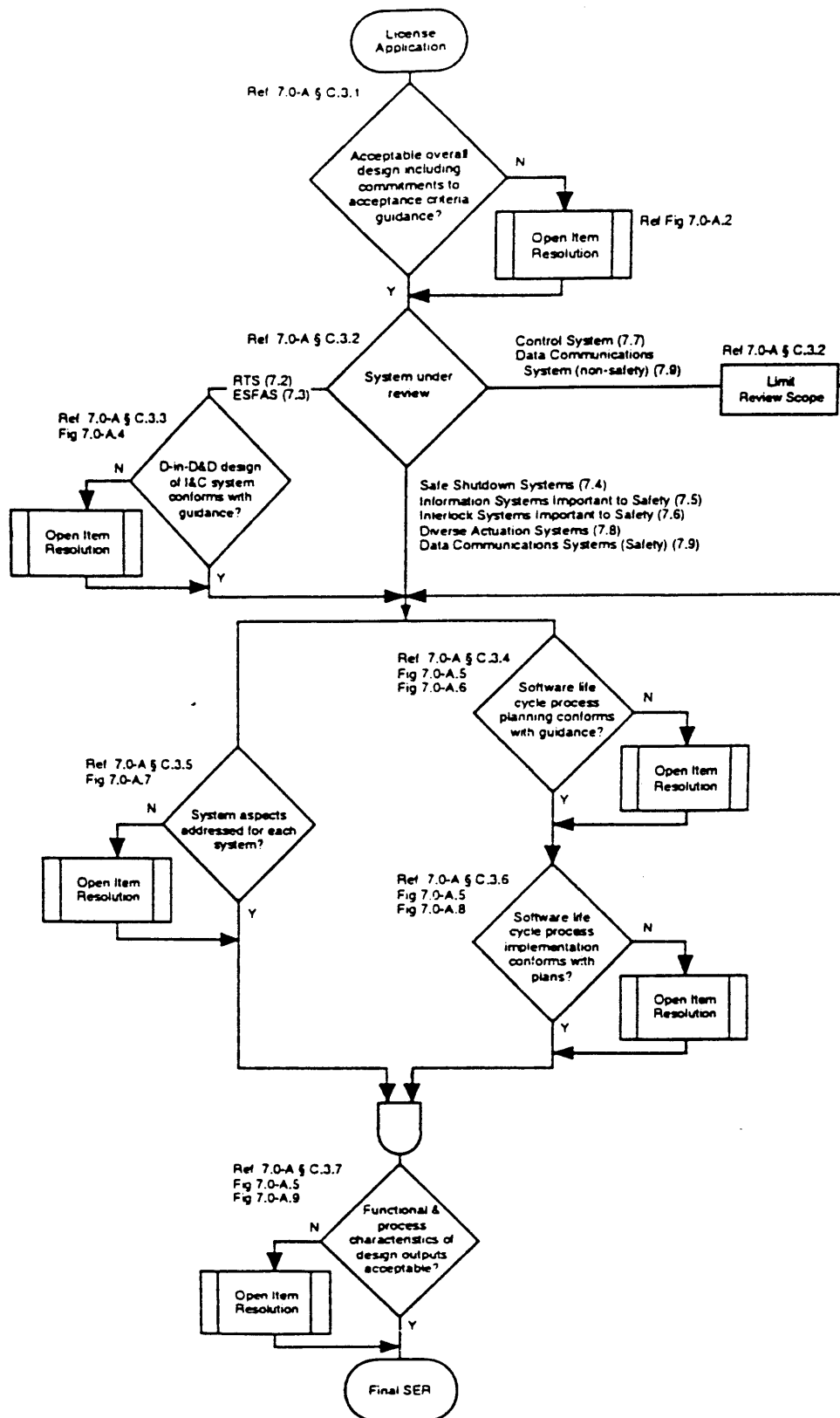
After a list of important definitions and the regulatory source documents for this appendix are listed, the aspects of *digital* I&C systems that warrant special attention in determining compliance with 10 CFR 50 are described, including “(1) design qualification of digital systems, (2) protection against common-mode failure, and (3) selected functional requirements of IEEE Std 603 and the General Design Criteria that pose new assurance challenges when implemented using computers.”

In relation to the first of these issues -- qualification of digital I&C systems -- emphasis is placed on supplementing the typical analog qualification process (e.g., inspections, acceptance testing) with assurance of a “*high-quality development process* that [incorporates] *disciplined specification and implementation design requirements*” [emphasis added]. This is to account for the difference in behavior (i.e., discontinuity and point failures) between digital systems and analog systems. In regard to the second issue, emphasis is placed on ‘quality’ and ‘defense-in-depth and diversity’ (‘D-in-D&D’) to combat the propagation of common-mode failures that can result from the high degree of sharing of data, code, and process equipment in digital I&C systems. Concerning the final issue, it says that the review process for such system aspects as real-time performance, independence, and on-line testing “must recognize the special characteristics of digital systems.”

Next a brief summary of the overall review of digital I&C systems is given (see Figure 2). Seven key topics that are said to be necessary for any *digital* I&C system review are discussed. These issues (with a summary of the guidance given about them) are:

1. *Adequacy of design criteria* -- requires a commitment to the following standards: Reg. Guide 1.152 (and IEEE 7-4.3.2, which it endorses) and Reg. Guides 1.1uu through 1.1zz





**Figure 2** Overview of the Process for Reviewing the Unique Aspects of Digital Instrumentation and Control Systems

(From SRP Appendix 7.0-A, p. 7.0-A-11)

2. *Identification of review criteria* -- based on the system's safety significance
3. *D-in-D&D* -- requires compliance with the corresponding BTP (19) and the Staff Requirements Memorandum to SECY-93-087
4. *Life cycle process planning* -- requires compliance with BTP-14 Section B.2.1 (essentially, to make sure that the system is planned, specified, and implemented appropriately and that there is sufficient testing or other verification of this fact)
5. *Adequacy of software functional requirements* -- in regard to the difficulties that arise as a result of using *digital* technology (e.g., in real-time performance, communications independence, etc.)
6. *Adequacy of software life cycle process implementation* -- per BTP-14, Section B.2.2
7. *Software life cycle process design outputs* -- per BTP-14, Section B.2.3 (involves assuring their adequacy by assuring the presence of adequate functional and software development process characteristics)

(For a more complete description of these review topics, see Appendix J at the end of this work.)

The last four topics in this list pertain specifically to software; their interaction is depicted in Figure 3. A description is given of how this illustration depicts the coming together of the I&C system-level functional requirements and the software development process requirements into software functional and development process characteristics.

### 2.1.9 Back to BTP-14

Finally, we return to BTP-14, which proposes a review of the planning, implementation, and design outputs of the software life cycle. In fact, the three stated purposes of BTP-14 are:

- 1) (*For the planning phase*): "to confirm that plans exist that will provide a high-quality software life cycle process, and that these plans commit to documentation of life cycle activities that permit the USNRC staff to evaluate the quality of the design features upon which the safety determination is based,"
- 2) (*For the implementation phase*): "to verify that implementation of the software life cycle process meets the criteria expected for high-quality software," and
- 3) (*For the design output phase*): "to assess the adequacy of the design outputs."

The introductory paragraphs of this branch technical position describe some basic concepts applicable to the subject of the document (e.g., the need for a system design commensurate with the system's safety importance, common-mode failure concerns and ways to avoid them, etc.). The need for a well-defined life cycle, with the activities listed in Reg. Guide 1.1zz (which endorses IEEE 1074) as a minimum, is described. Life cycle activities are grouped into various 'activity groups,' each of which requires certain input documents and produces required outputs. These outputs are grouped into 'design

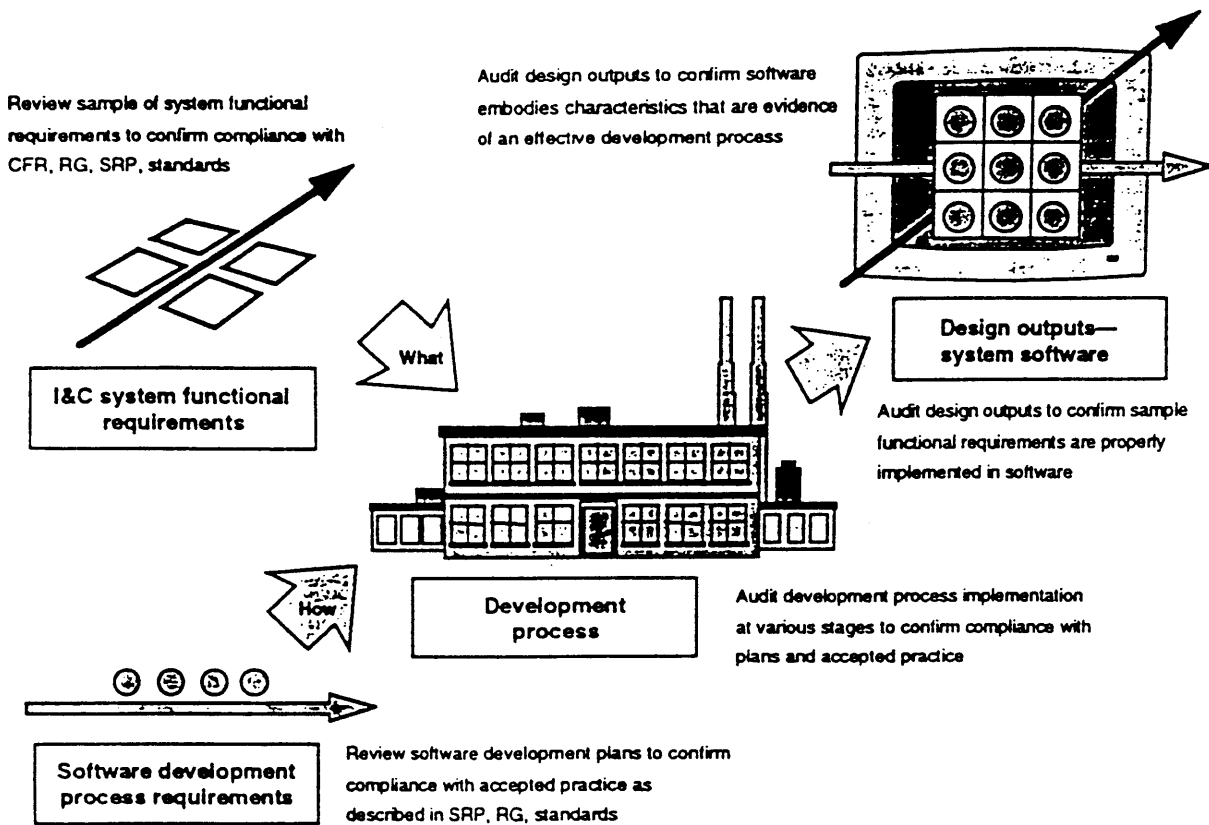


Figure 3 Software Review Process  
(From SRP Appendix 7.0-A, p. 7.0-A-13)

outputs,' which are unique to each activity group, and 'process documents' (safety analyses, V&V reports, and configuration management reports) which are produced for all activity groups. (Consult Appendix E to see some direct examples of BTP-14 review guidance for some pertinent sample documents from each of the three review areas.)

BTP-14 defines two basic sets of 'functional characteristics' and 'development process characteristics,' which are then used as the basic acceptance criteria to show that each section of the life cycle has been implemented effectively. The functional characteristics are *accuracy, functionality, reliability, robustness, safety, security, and timing*. The software development process characteristics are *completeness, consistency, correctness, style, traceability, unambiguity, and verifiability*. Some of these terms are rather vaguely defined (for example, accuracy is simply defined as the "degree of freedom from error in input, calculations, and output"). (See Appendix K for the definitions of all of these qualities.)

The functional characteristics are concerned only with the ability of the software *product* to accomplish the functions for which it is intended. The software development process characteristics, which are supposed to be a natural by-product of a well-implemented software design *process*, build confidence that the functional characteristics were successfully implemented.

For the planning phase, BTP-14 lists all of the required output documents (as seen in the 'Planning Activities' column of Figure 4) and a qualitative list of the contents to be included in each one. The information to be reviewed for the software life cycle planning review is found throughout all of the planning documents listed in the 'Planning Activities' activity group (the first column of the software life cycle diagram in Figure 4). Examples of these documents include the Software Quality Assurance Plan, the Software Safety Plan, and the Software Verification and Validation Plan (SVVP). The reviewer is referred to NUREG/CR-6101 for supplementary guidance on evaluating each of these documents.

The implementation phase analysis lists the activities to be required as part of safety analysis, V&V, and configuration management. The information to be reviewed for this phase is found in the safety analyses, V&V analyses and test reports, and configuration management reports for each life cycle activity group. Again, supplementary information "relevant to each subject" can be found in NUREG/CR-6101. The reviewer is never referred to any particular part of NUREG/CR-6101, and it is not always obvious what the appropriate part would be.

For the design output phase, there is a list of all of the design outputs, the characteristics they should exhibit, and a brief description of how those characteristics relate to that particular document. The information to be reviewed for the software life cycle design output review is found in the documents listed in the 'design outputs' section of each life cycle activity group. It is during this review that the software is evaluated (by way of the design output documents) for each of the previously described functional and software development process characteristics. For example, the SRS, listed under the 'Requirements Activities' column in Figure 4 should exhibit *all* of the characteristics -- that is: accuracy, functionality,

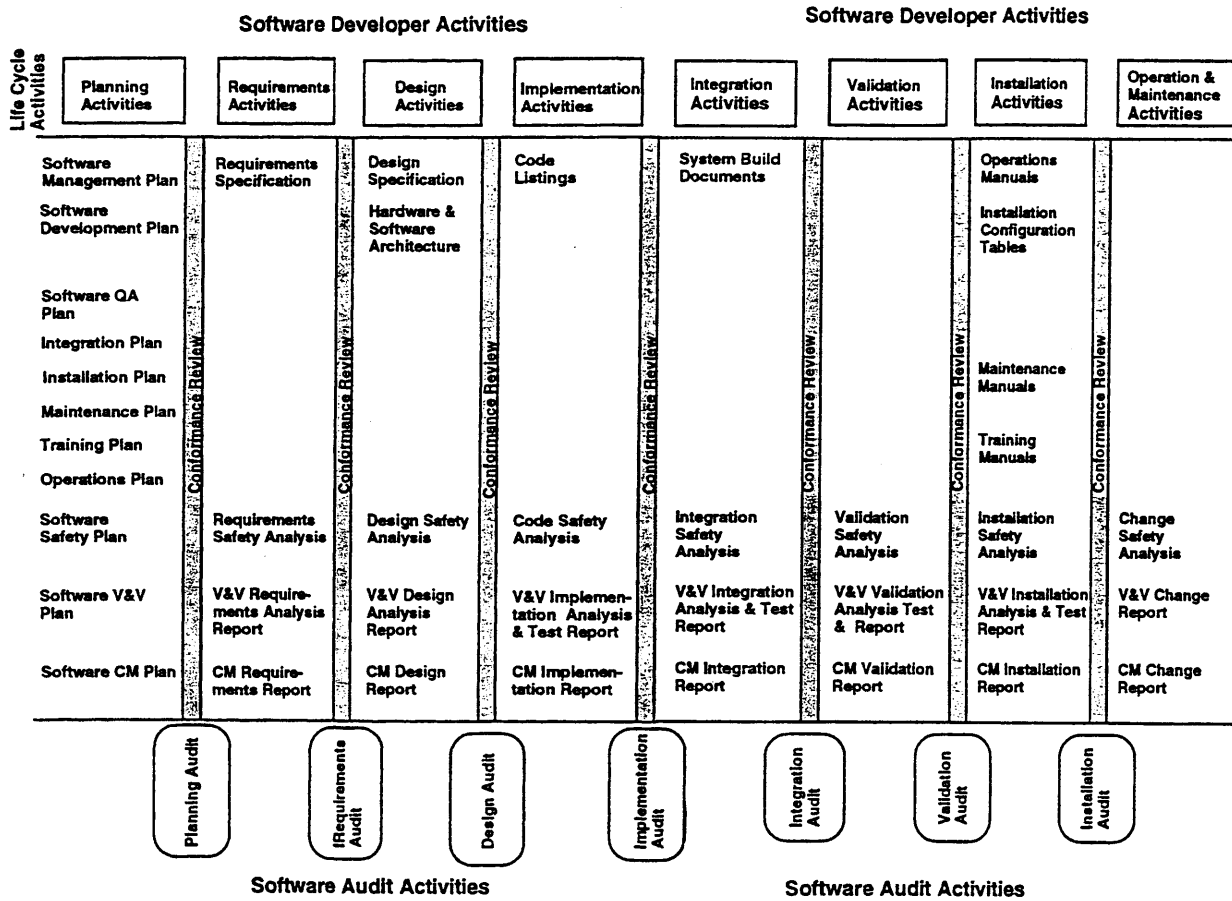


Figure 4 Documents Produced During Each Life Cycle Stage  
 (From NUREG-CR/6101, pp. 8-9)

reliability, robustness, safety, security, timing, completeness, consistency, correctness, style, traceability, unambiguity, and verifiability.

This concludes the description of the contents of the BTP-14 software review process. Now the process will be evaluated.

## 2.2 Evaluation of the BTP-14 Review Process

This section begins with background information on defining a software development process model. Then an evaluation of BTP-14 (and the greater regulatory structure of which it is a part) will be given by stepping through the review (and design) process it prescribes in the same order that was followed in Section 2.1. This evaluation will be used to point out both large scale problems with the process in general and small scale problems that help to illustrate some of the larger problems. Next the issue of V&V within the process will be analyzed more closely; this will serve as a specific example to illustrate the problems already noted. Finally, after looking both at the big picture of the process and at this detailed part of it, a summary of conclusions and recommendations for the entire process will be given.

### 2.2.1 Background

Like any evolutionary process, software development is something that for many years defied attempts to be clearly defined or arranged into an orderly process. Thus it has tended to be a somewhat haphazard, if not chaotic, process which is different from case to case, depending on the particular application involved. Attempts have since been made to clearly define the software development process in order to make it more manageable. One of the best known and often-used models comes from efforts at the Software Engineering Institute (SEI). The SEI uses a five-level Capability Maturity Model (CMM) to determine the maturity and success of a software development process. The levels, from lowest (Level 1) to highest (Level 5), are: *initial*, *repeatable*, *defined*, *managed*, and *optimizing*.<sup>24</sup> It must be said that even the value of this model is questioned by many. Since it is currently the most commonly referenced model, however, it can serve as a benchmark for BTP-14.

The basic characteristics and challenges of each level in the CMM are described as follows:<sup>24</sup>

1. Level 1 -- Initial -- 'ad hoc/chaotic process'
  - a. *Characteristics* -- no formal procedures, cost estimates, or project plans; no management mechanism to ensure procedures are followed; tools not well integrated; change control is lax; senior management does not understand key issues
  - b. *Challenges* -- project management and planning, configuration management, and software quality assurance

2. Level 2 -- Repeatable -- 'intuitive'
  - a. *Characteristics* -- process dependent on individuals; established basic process controls; strength in doing similar work, but faces major risk when presented with new challenges; lacks orderly framework for improvement
  - b. *Challenges* -- training, technical practices (reviews, testing), and process focus (standards, process groups)
3. Level 3 -- Defined -- 'qualitative'
  - a. *Characteristics* -- process defined and institutionalized; Software Engineering Process Group established to lead process improvement
  - b. *Challenges* -- process measurement and analysis, *quantitative* quality plans
4. Level 4 -- Managed -- 'quantitative'
  - a. *Characteristics* -- measured process; minimum set of quality and productivity measurements established; process database established with resources to analyze its data and maintain it
  - b. *Challenges* -- changing technology, problem analysis, and problem *prevention*
5. Level 5 -- Optimizing
  - a. *Characteristics* -- improvement fed back into process; data gathering is automated and used to identify weakest process elements; *numerical* evidence used to justify application of technology to critical tasks; rigorous defect-cause analysis and defect *prevention*
  - b. *Challenges* -- still human-intensive process; maintain organization at optimizing level

It will be shown that the software development procedure that stems from the BTP-14 review process seems to be strong only through Level 3. It can be argued that it has some of the necessary traits of Levels 4 & 5 too, but this is debatable. That BTP-14 has the qualities of Levels 1 & 2 is fairly obvious, and the process is also "defined" as required in Level 3. However, the existence of "*quantitative* quality plans" in BTP-14 is minimal at best. This is in keeping with such plans being a "challenge" at Level 3. Though BTP-14 is not necessarily devoid of the characteristics listed under Level 4, it is deficient in one of the "challenge" areas listed -- namely, problem *prevention*. Likewise, it is deficient in the Level 5 characteristics of defect *prevention* and use of *numerical* evidence to justify application of technology. The BTP should strive to be at these higher levels.

### 2.2.2 Evaluation

As mentioned, the original purpose of Section 2.1 was to develop an orderly summary of the software development review process. This was difficult to do, however, due to the number of interwoven references to other documents that are such a pervasive part of the BTP-14 process. It is apparent that, if branch technical positions are supposed to serve as stand-alone documents to guide someone through a process (in this case to guide a reviewer through the development process for high-reliability software), then this BTP does not meet the goal.

It must be stated at this point that in comparison to the other BTPs that are part of the overall instrumentation and controls regulatory process for NPPs (that is, BTPs 1 through 21, excluding 7, 15, and 20), this one is quite thorough. While there are examples in the other BTPs of guidance written similarly to that of BTP-14, there are also examples of more precise, detailed, practical guidance.

An example of guidance similar to that found in BTP-14 would be the following quote from Branch Technical Position HICB-8 (which provides guidance for applying Reg. Guide 1.22 with respect to periodic testing of protection system actuation functions).

*“All portions of the protection systems should be designed in accordance with ANSI/IEEE Std 279, as required by 10 CFR Part 50, 50.55a(h). All actuated equipment that is not tested during reactor operation should be identified, and a discussion of how each conforms to the provisions of paragraph D.4 of Reg. Guide 1.22 should be submitted.”* [emphasis added]

This guidance is similar to that found throughout BTP-14 and the SRP in that it is vague and relies on sending the reader to numerous other documents.

An example of more detailed and useful guidance would be the following recommendations from Branch Technical Position HICB-2, concerning features that should be “incorporated in the design of [motor-operated isolation valve] systems for safety injection tanks to meet the intent of ANSI/IEEE Std 279”. (Notice the guidance terms used below, which are more specific than the italicized guidance words in the paragraph above.)

“Automatic opening of the valves when either primary coolant system pressure exceeds a preselected value (to be specified in the technical specifications), or a safety signal injection signal is present. Both primary coolant system pressure and safety injection signals should be provided to the valve operator.....Visual indication in the control room of the open or closed status of the valve.....Bypassed and inoperable status indication in accordance to Regulatory Guide 1.47.....Utilization of a safety injection signal to remove automatically (override) any bypass feature that may be provided to allow an isolation valve to be closed for short periods of time when the reactor coolant system is at pressure (in accordance with provisions of the technical specifications).”

It is true that the subject of BTP-14 (the software review process for digital I&C safety systems) is one that is generally less clearly and less tangibly understood than the subjects of many of the other BTPs



mentioned above (e.g., motor-operated valves). This does not, however, mean that the guidance of BTP-14 cannot be improved. For example, the BTP says that in order for the SRS to exhibit *completeness*, it must, among other things, specify “all actions required of the computer system” (and all “actions that the software is prohibited from executing”) under “all operating modes and all possible values of input variables (including anomalous values)” and that it must describe “[the] operational environment.” This is a very broad and ill-defined set of requirements. *How* does one *specify* actions (and modes and input variables) of the computer system -- in plain English, in some type of formal notation, or some other method? In the same vein, how does one “describe” the “operational environment”? What specific factors should be considered as relevant to the operational environment? *How* should the environment be described -- in normal language, with a mathematical representation, or in some other manner?

In the passage from BTP-2 above, the instruction to use a “safety injection signal” to “automatically [override] any bypass feature” that allows “an isolation valve to be closed for short periods of time when the [Reactor Containment System] is at pressure” is much clearer and less ambiguous than the regulation just cited from BTP-14. This BTP-2 guidance is not flawless either (e.g., how exactly is a “short period of time” defined?), but it certainly has less ambiguity than that of BTP-14.

The goal of BTP-14, however, should not be simply to be adequate in comparison with other BTPs, but rather to be as good as possible (i.e., as good as the limits of our current knowledge) at guiding a reviewer through the review of digital I&C safety system software. If it exceeds the quality of other BTPs, then maybe the other BTPs can also be improved.

Attention will now be turned away from a comparison of BTP-14 with other BTPs to an evaluation of it on its own merits. The crudest ordered outline of the BTP-14 review is described below. The order of some of these steps is debatable. (Since BTP-14 is a guidance document that fits into a larger scheme of review documents, it cannot meaningfully be separated from the larger picture, as seen in this list.):

- > Step 1 -- Determine application type (1<sup>st</sup> step of SRP 7.0).
- > Step 2 -- Review application scope and contents (2<sup>nd</sup> step of SRP 7.0).
  - >> Step 2.a -- Ensure that all safety-related I&C systems necessary to comply with 10 CFR 50 are addressed and that their acceptance criteria and guidelines are identified (1<sup>st</sup> step of SRP 7.1). -- [This step and steps 4 through 6 will require reference to the criteria and procedures described in Appendices 7.1-A, B, and C.]
  - >> Step 2.b -- Review exceptions taken to non-mandatory guidelines for acceptability (2<sup>nd</sup> step of SRP 7.1).
  - >> Step 2.c -- Items previously approved by the USNRC and completely new technologies should be identified separately (3<sup>rd</sup> step of SRP 7.1).

- >> Step 2.d -- Reference Appendix 7.0-A for guidance on the seven digital issues that it says are mandatory for any review of *digital* I&C systems. This is the section of the review where the guidance in BTP-14 is most directly applicable, since all seven of the important digital issues it describes are critical to the software design life cycle. (The BTP-14 review can be executed in the order in which it is described: software life cycle process planning; then implementation; then design outputs.)
- >> Step 2.e -- Review proposed solutions to unresolved safety concerns from (prior) design certification (4<sup>th</sup> step of SRP 7.1).
- > Step 3 -- Create an application-specific review plan (3<sup>rd</sup> step of SRP 7.0).
- > Step 4 -- Carry out the review “using the acceptance criteria and review processes of the SRP” (4<sup>th</sup> step of SRP 7.0).

The reason this sequence is somewhat illusory is that many of the steps and procedures in it are not restricted to one point in the process. For instance, many of the documents that must be used are used as references at many points throughout the review. At any rate, there is no clear entry point into and exit point out of this series of steps. It is iterative at best and *confusing* at worst.

In stepping through the detailed BTP-14 description given above, numerous points warrant further comment. To use BTP-14, we first detoured to several other related documents and reference documents. First we referred to SRP Appendix 7.0-A, from which we went to SRP Section 7.0.

#### 2.2.2.1 SRP Section 7.0

As already explained, the SRP describes the most fundamental steps in the I&C review. Much of the detail on specific sub-topics of this process (e.g., digital- and software-related topics) is provided by other documents (e.g., BTP-14). However, the level of detail in SRP 7.0 is too minimal even for a higher-level document. For instance, the descriptions of the scope and content points to consider for each type of I&C license or application are not very helpful at all (see Appendix 3). In essence, all it says is to review the I&C system application’s adherence to functional requirements (such as those of 10 CFR 50), its resolutions or proposed research and development to solve any problems or questions, its principal design criteria and their relationship to the design bases, its technical specifications and safety margins (to assure performance requirements can be met), and its demonstration of requirement satisfaction and design process characteristics (as evidenced by its implementation, testing, and design outputs). These are very broad, high level instructions.

The final guidance statement in SRP 7.0 says to execute the I&C review “using the acceptance criteria and review processes of the SRP.” This is a very open-ended statement. The SRP is a large document covering a lot of material. Guidance should narrow in on something a bit easier to handle than the *entire* SRP.

### 2.2.2.2 SRP Section 7.1

Continuing on with the review process, the user must consult the information in SRP Section 7.1 as well. The purpose of searching through many levels of references is, presumably, to find a final, source-level answer on a particular topic. This is not often the case with these documents, however. For instance, in talking about V&V requirements for computer system quality, SRP 7.1 refers the reviewer to “the software engineering process as described by BTP HICB-14,” among other documents. This reference is not particularly helpful, since it refers the user to BTP-14 as a whole rather than to any specific subsection. Even after consulting all of these references, the reviewer still comes back to the question of *how* to implement the suggestions that are found all along this path.

### 2.2.2.3 SRP Appendix 7.1-A

SRP Appendix 7.1-A provides general guidance on the basic assessment criteria and guidance for the digital I&C and safety software review process. Most of this guidance is just as unclear as that already discussed. For example:

- \* The basic requirement on safety system quality standards (10 CFR 50.55a(a)(1)) says that all parts of the safety system should be designed and built “to quality standards commensurate with the importance of the safety function to be performed.” Review guidance for this requirement simply refers the user to the regulatory guides and standards referenced in Sections 7.1 through 7.9 and Chapter 7 Appendix A. This is not a very helpful reference; it refers the reviewer to a substantial portion of the *entire* review process rather than to any particularly useful sections of it.
- \* The basic requirement on the required level of detail for reactor license applications (10 CFR 52.47(a)(2)) mandates “detail sufficient to.....judge the.....licensee’s [conformance to the design] and to reach a final conclusion on all safety questions.....[and] to permit the preparation of acceptance and inspection requirements by the NRC.” Also, “sufficient information for an NRC safety determination should be provided for each I&C system.”
  - > The reviewer is referred to BTP-16 for guidance on determining what level of detail is sufficient. To determine what is sufficient in reference to SRP 7.1, BTP-16 provides more vague advice: It reiterates guidance from other SRP sections (e.g., 7.0) and says that applications “should describe the computer system development process [and, where applicable,] the design of the overall I&C systems with respect to D-in-D&D requirements.” It also says that the “discussion should include a commitment to a design process compatible with that described in Reg. Guide 1.152 and BTP HICB-14.”
- \* The GDC on quality standards and records (GDC 1) gives the following guidance: “A quality assurance program shall be.....implemented in order to provide *adequate assurance* that these.....systems.....will *satisfactorily perform* their safety functions.....” The review guidance for doing this is not any more useful; it says to consult “the *applicable* regulatory guides and endorsed codes and standards” [emphasis added]

- \* The review methods guidance for all of the Reg. Guides is fairly similar, and for the most part it simply refers the reviewer to multiple other documents that support the corresponding Reg. Guides.
- \* The review methods guidance for the BTPs is particularly useless, because it says only that “the BTPs provide bases for evaluating specific review areas.”

#### 2.2.2.4 SRP Appendices 7.1-B and 7.1-C

SRP Appendices 7.1-B and 7.1-C are very similar. Appendix 7.1-B begins by listing and briefly describing the elements of the “design basis” for protection systems. This brings out another trend in this regulatory process. The driving principles and important points are not unified into an overarching theme at any one point. There are numerous key topics of concern that guide the review process, but they seem to be mentioned haphazardly at random locations throughout the review documents. Consider the following:

- \* The BTP says that the “[USNRC] Staff’s acceptance of software for safety system functions is based upon” (1) confirmation of acceptable software development plans, (2) evidence that the life cycle followed the plans, and (3) acceptable design outputs. However, the BTP also mentions the USNRC Staff’s emphasis on quality and D-in-D&D “as protection against common-mode failures within and between channels” as well as the importance of “software quality [as] an important element in preventing the propagation of common-mode failures.”
- \* SRP Appendix 7.0-A emphasizes the following three major topics: (1) qualification of digital I&C systems and components, (2) defense against common-mode failure, and (3) system aspects of digital I&C, in addition to the seven digital-specific topics mentioned in the discussion of SRP 7.0-A in Section 2.1
- \* SRP Section 7.1 provides yet another list of important point of “supplemental guidance for digital computer-based safety systems, similar to but slightly different from those in 7.0-A

Some unification of all of these different important points would be helpful in clarifying the driving forces behind the BTP/SRP software review process.

After this introductory information, Appendix 7.1-B offers advice on such topics as the following, which further illustrates the problems of vagueness and constant referencing mentioned so many times already:

- \* In discussing general functional requirements, it says that the “degree of redundancy, diversity, testability, and quality.....[must be] *adequate* to achieve functional reliability commensurate with the safety functions to be performed.” [emphasis added] How should these qualities be shown to be adequate? The reviewer is referred back to BTP-14 and to Reg. Guide 1.152 for guidance on designing for and determining software reliability.

### 2.2.2.5 SRP Appendix 7.0-A

One of the issues addressed by SRP Appendix 7.0-A is supplementing the normal analog system qualification process with steps to assure a “high-quality development process that [incorporates] disciplined specification and implementation design requirements” when dealing with digital systems. It also says the review process for systems aspects of digital systems “must recognize the special characteristics of digital systems,” another vague guidance statement that could be clarified to be more helpful.

The following specific comments are made regarding the digital I&C system software review:

- \* “.....This review should confirm that the special design considerations of digital systems are *appropriately* considered.” [emphasis added]
- \* “Review of planning activities confirms that.....development process requirements establish a commitment to an effective and disciplined software development process.”
- \* “Inspection of the development process confirms that the planned process is actually used, and that *appropriate* safety analysis, verification and validation, and configuration control activities are conducted.”
- \* “Audits of design outputs confirm that functional requirements are traceable through all intermediate design products to the final product [and] that they exhibit the required software development process characteristics.”

These statements are very vague (notice especially the italicized words above) and offer no new insights on *how* to accomplish the task of software design and review effectively.

This appendix also lists seven issues that should be part of *any* digital I&C system review. The interaction of four of these issues (life cycle process planning, adequacy of software functional requirements for individual I&C systems, adequacy of software life cycle process implementation, and software life cycle process design outputs) is illustrated in Figure 3. This diagram is a particularly unhelpful addition to the SRP/BTP review process. Although it may make perfect sense to those intimately involved with defining this software development process, it does not carry much intuitive meaning. None of the illustrations at the end of Appendix 7.0-A are of much benefit (see Figures 2, 3, and 5 through 9). They offer no clear insights into their respective topics.

The exact relationship between the different parts of this review process and documentation (i.e., BTP-14, SRP Section 7.0, SRP Appendix 7.0-A, SRP Section 7.1 and its appendices) is sometimes hazy and quite difficult to summarize. Consider SRP Appendix 7.0-A and SRP Section 7.1. Appendix 7.0-A is concerned primarily with clarifying the I&C system review process so it will work better for digital systems. Three general digital issues and seven specific subissues within the review process that warrant special attention are described. Section 7.1 gives an overview of many aspects of the review process (e.g., what I&C systems are important to safety; who has review authority; the background of, and relationship

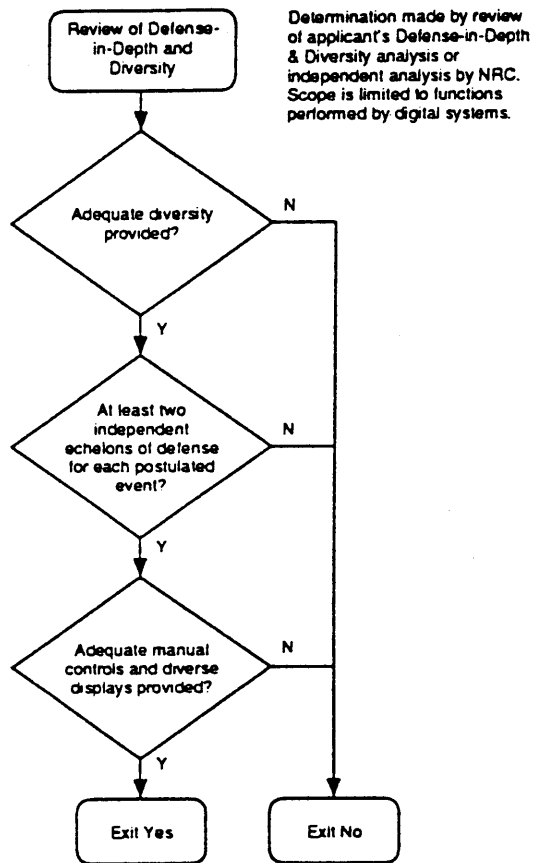


Figure 5 Defense-in-Depth and Diversity Review  
(From SRP Appendix 7.0-A, p. 7.0-A-14)

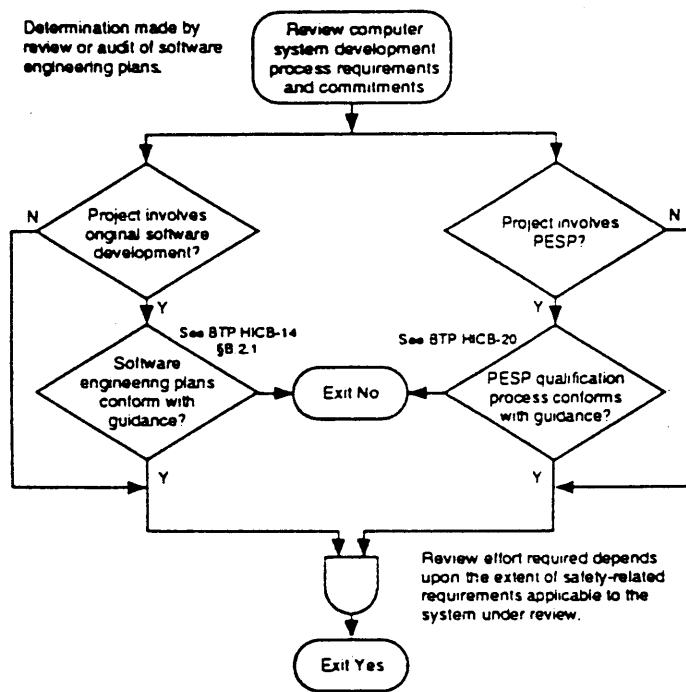
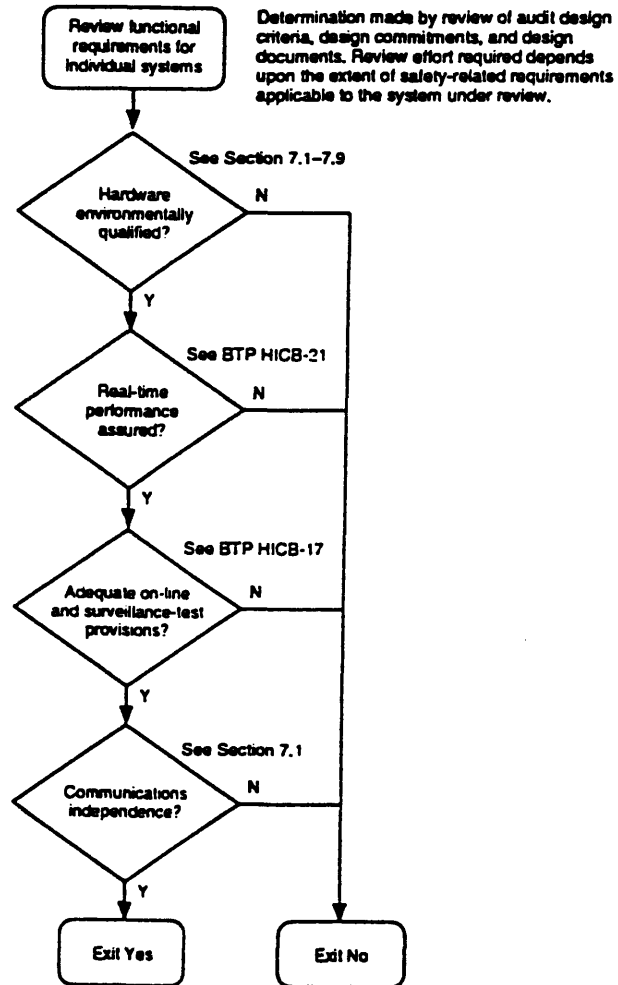


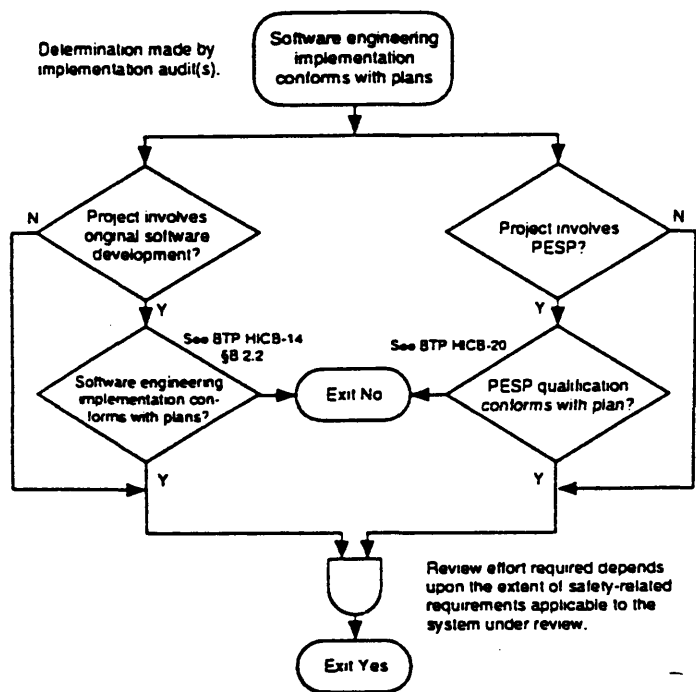
Figure 6 Review of Software Lifecycle Process Planning  
 (From SRP Appendix 7.0-A, p. 7.0-A-16)



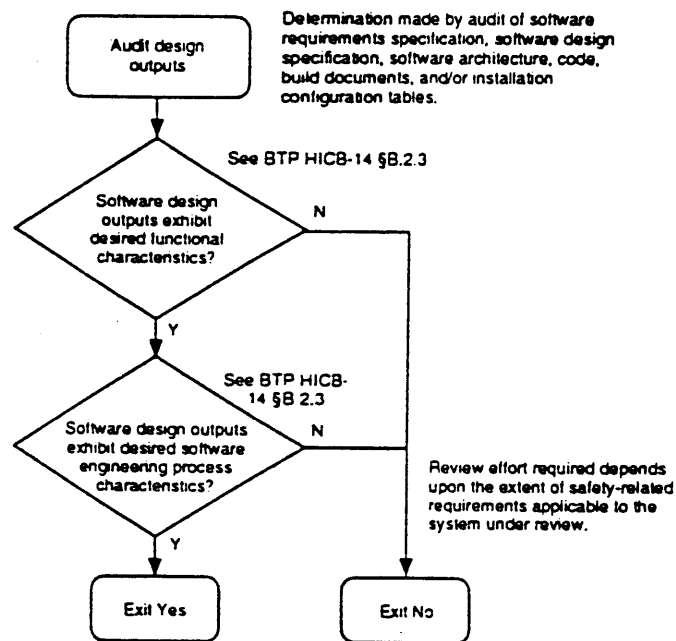
**Figure 7** Special Considerations in the Review of Functional Requirements for Digital Instrumentation and Control Systems

(From SRP Appendix 7.0-A, p. 7.0-A-17)





**Figure 8** Review of Software Development Process Implementation  
(From SRP Appendix 7.0-A, p. 7.0-A-18)



**Figure 9** Review of Design Outputs  
(From SRP Appendix 7.0-A, p. 7.0-A-19)

between, acceptance criteria and guidelines), but it also devotes a large section to “supplemental guidance for digital computer-based safety systems” (and how to apply it). It is not clear what the relationship is between the digital issues discussed in Appendix 7.0-A and the digital issues discussed in Section 7.1. They are all important to the overall I&C system review, but why are they discussed in separate locations?

It seems that it would make more sense to list all of these issues together in one location. Again, there needs to be some type of all-encompassing structure of principles and rules that unites these disjointed pieces of the puzzle.

#### 2.2.2.6 BTP-14

Finally we come back to BTP-14, which, as is now apparent, is just one part of a rather interdependent mass of support documents for this I&C regulatory process. The problems with BTP-14 itself are the same types of problems as are evident in the SRP and the other documents evaluated up to this point.

One major problem with BTP-14 and the accompanying standards is that they are confusing. Though these documents are certainly thorough in their total breadth of coverage, one needs a roadmap to determine how to get started, where to go from there, and how to know when to move to the next step of the process. It also does not emphasize clearly what its main points are, or, at the very least, underemphasizes some very important points. The closest thing to a summary statement of the BTP-14 guiding principles that can be found is the following statement from page 4 of the BTP:

“Because of these concerns [relating to common-mode failures], the Staff review of digital I&C systems emphasizes *quality*, *defense-in-depth*, and *diversity* as protection against common-mode failures within and between channels. Software quality is an important element in preventing the propagation of common-mode failures.” [emphasis added]

This statement, however, is buried in among many other lengthier paragraphs describing various issues of importance in BTP-14. It is easily overlooked, because none of these paragraphs stand out as being particularly more important than the others.

Another problem with BTP-14 is that it does not make clear what exactly should be done to accomplish many of its goals. While the ultimate goal of the NRC is to be less prescriptive in its regulations, this proposed process seems to leave too many gray areas. For example, the BTP says that one of the characteristics the SRS should exhibit is *completeness*. As mentioned, to fulfill this criterion, all actions the system must [or must not] perform should be fully described in terms of all monitored or controlled variables for all modes and all possible inputs. Although this requirement is theoretically ideal, it is rather broad and impractical to implement, and it provides no quantitative goals by which to gauge whether the software meets some minimum standard.

There are many guidance documents on other subjects that offer a better standard for which to strive. Take, for example, Reg. Guide 1.52, “Design, Testing, and Maintenance Criteria for Post-Accident

Engineered Safety Feature Atmosphere Cleanup System Air Filtration and Adsorption Units of Light-Water-Cooled Nuclear Power Plants.” It offers guidance such as the following: “The design of an [Engineered Safety Feature] atmosphere cleanup system should be based on the maximum pressure differential, radiation dose rate, relative humidity, maximum and minimum temperature, and other conditions resulting from the postulated [design basis accident] and on the duration of such conditions.” This statement is not prescriptive (i.e., “should be based on” leaves some room for interpretation), but it spells out what must be accounted for (maximum pressure differential, radiation dose rate, etc.) in order to satisfy its requirements and leaves less room for interpretation.

While it may be good to be less prescriptive, another goal of the USNRC is to get away from a costly and time-consuming case-by-case licensing process. The proposed review process, because it is so often vague and open-ended, makes it relatively easy for designers to stray from the guidance (as long as they can explain their rationale). For this reason, too little standardization may ultimately detract from the goal of saving time and money.

Both the planning review and the design output review focus on heavily qualitative evaluations of documents. The many acceptance criteria that BTP-14 offers are almost entirely qualitative too. This may be due to the attempt to make this review process less prescriptive than that currently in use. It may also stem from the fact that software behavior is not as tangible and easy to work with as a hardware component or system. Nevertheless, it would be helpful to have more clear, physically or quantitatively meaningful advice on how to achieve certain requirements.

BTP-14 fits into the broader category of review of digital instrumentation and controls (covered by SRP Appendix 7.0-A), which in turns fits into the even broader category of review of instrumentation and controls in general (covered by SRP Section 7.0). Both of these areas are explained in further detail in SRP Section 7.1. As described already, the main points of BTP-14 and these pertinent SRP sections do not always seem to mesh, and the flow between them lacks continuity.

An example of the meandering path one needs to follow through these documents will provide some insight into their complex organization and into how difficult the use of BTP-14 can be. (This will be expanded upon in Sections 2.3.3 through 2.3.6.) In going through BTP-14, Section B.2.1.2.j (a bulleted list of acceptance criteria for the Software Verification and Validation Plan), it says that “appropriate organization” for the SVVP is shown in (proposed) Reg. Guide 1.1yy, which endorses ANSI/IEEE Std 1012 (and 1028).

Upon consulting Reg. Guide 1.1yy, one finds, essentially, a rehash and endorsement of many other related documents and standards (10 CFR 50, IEEE 1012 & 1028, etc.). Reg. Guide 1.1yy does not, in and of itself, provide an “appropriate organization” for the SVVP; it simply endorses (with a few exceptions and additions) the organization described in IEEE 1012.

The IEEE standards are usually the lowest-level documents in this hierarchy of references). However, even IEEE 1012-1986, which is devoted entirely to SVVPs and hence is essentially the lowest-

level document on V&V, does not offer a lot of clear insight into the topic. It simply goes through the V&V process phase-by-phase, *qualitatively* describing what minimum tasks must be accomplished (e.g., trace software requirements to system requirements and analyze the relationships between them for such qualities as correctness, consistency, completeness, and accuracy) and what inputs and outputs (in the form of documents) are required for those tasks. It is basically just a ‘laundry list’ of items to include in the SVVP; it does not offer any useful suggestions as to how to execute any of the tasks that it requires. In addition, while IEEE 1012 does not specifically rely on other standards for its use, it is to be “used in conjunction with” IEEE 729, IEEE 730, IEEE 828, IEEE 829.

This leads back to a problem with much of BTP-14 that has been pointed out many times already: it is very vague. Following are some examples of particularly vague statements from BTP-14 (with their location in the BTP in parentheses):

1. The software integration plan should contain a “description of the hardware and software integration process.” (B.2.1.2.d)
  - > *What exactly should be described about the integration process?*
2. The operations plan should contain a “description of the procedures for executing the software in all operating modes, and procedures for ensuring that the software state is consistent with the plant operating mode at all times.” (B.2.1.2.h)
  - > *This implies a lot in a very short sentence. However, it does not specify what should be included in these descriptions. What are the important elements of the operating modes and the corresponding software states?*
3. The software V&V plan should contain a “description of the V&V activities, including the methods and procedures, the acceptance criteria for each activity, risk management procedures, relationships with the product development life cycle tasks, and coordination with [Software Configuration Management] activities,” and a “description of all required testing plans, specifications, procedures and cases, [including] unit testing, integration (subsystem) testing, system testing, and acceptance testing, [which should also include] test documentation requirements, evaluation criteria, error reporting, and anomaly resolution procedures.” (B.2.1.2.j)
  - > *Again there are a lot of high-level terms here (e.g., description, methods, procedures, relationships) whose specific meanings are not described.*
4. Software V&V documentation should confirm that “the requirements, design elements, and code elements satisfy the appropriate functional characteristics of accuracy, functionality, reliability, robustness, safety, security, and timing [and the software development process] characteristics of completeness, consistency, correctness, style, traceability, and unambiguity.” (B.2.2.2.b)
  - > *How do we determine which qualities are appropriate for which elements, and how exactly do we satisfy them (for instance, exactly what conditions are required for requirements to be “complete”)?*

5. “An [installation or] acceptance test report should be produced describing the execution of the [test] plan and summarizing the results. This report should contain a statement that the plan was successfully executed, and the system is ready for operation.” (B.2.2.2.b)
  - > *These are rather uninformative statements. In addition, it sounds as if the designer simply needs to create a test plan that is executable rather than one that is conclusive.*
6. System build documents should exhibit *completeness*, which “requires that all build procedures be fully specified...” (B.2.3.2.e)
  - > *What exactly is entailed by the phrase ‘fully specified’?*

## 2.3 The Place of V&V and Testing in the BTP-14 Review Process

This section will begin by giving some basic definitions important to V&V and pointing out the confusion involved in defining these terms. This will be followed by a brief discussion of some of the theory behind V&V. Several sections will then describe the position of V&V in various standards and documents that provide guidance for conducting a V&V effort. Then there will be a discussion of various methods and types of testing (which is just one part of V&V) and of how they might be used effectively in combination with each other. A final section is included on problems with testing.

### 2.3.1 V&V Definitions

The issue of V&V is at the center of much of the debate over how to improve software and the software development process. Some very basic definitions of essential V&V terminology (taken from IEEE Std 610.12-1990, “IEEE Standard Glossary of Software Engineering Terminology”) are appropriate, since much confusion in this field stems simply from improper use of terminology.<sup>25</sup>

This confusion in definitions is a particular problem with the BTP/SRP software review process. Verification and validation are particularly important elements of this process, and the definitions that they are based on (from IEEE Std 610.12, the standard IEEE software engineering glossary) are not well written. *Verification* is defined as “the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase” (definition 1 from the glossary). One way to implement this is with a “formal proof of program correctness” (definition 2 from the glossary). *Proof of correctness* is a “formal technique used to prove mathematically that a computer program satisfies its specified requirements.” *Validation* is defined as “the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.” *Verification and validation (V&V)* is defined (separately) as “the process of determining whether requirements for a system or component are complete and correct, whether the products of each development phase fulfill the requirements or conditions

imposed by the previous phase, and whether the final system or component complies with specified requirements.”<sup>25</sup>

There appears to be inconsistency among these standard definitions, as the definition of ‘verification & validation’ contains more than the sum of the (separate) definitions of ‘verification’ and ‘validation’ (i.e., determining whether system requirements are complete and correct is not included in the definition of either single term, but it is part of the combined term). This may just be an issue of semantics, but it is confusing nonetheless.

In the above definitions, ‘validation’ seems to be little more than a repetition of ‘verification’ (i.e., it involves doing the same tasks at the end of the software development process as are done in between each phase of the process). Rushby better clarifies the distinction between these two terms.<sup>26</sup> He says that *formal verification* is “the process of showing, by means of formal deduction, that a formal design specification satisfies its formal requirements specification.....Assumptions and designs at one level become requirements at another,” allowing for an iterative verification process. *Formal validation*, on the other hand, is “a process for gaining confidence that top-level formal specifications of requirements and assumptions are correct.” Since “there are no higher-level requirements or more basic assumptions against which to verify” the top-level specifications, formal validation consists of “challenging the formal specifications by proposing and attempting to prove theorems that ought to follow from them.” Validation, then, involves determining not only that requirements are fulfilled (as verification determines) but also that the requirements are the *right* requirements in the first place.

The field of computer simulation also has a more well defined difference between the terms (note that these definitions apply specifically to computer modeling rather than to software). *Model verification* is defined as “substantiation that a computerized model represents a conceptual model within specified limits of accuracy.” *Model validation* is defined as “substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model.”<sup>27</sup> Here verification deals with how well a model is implemented by a computer program, while validation deals with whether or not the model is, in fact, an accurate reflection of the real world.

### 2.3.2 V&V Theory

There is often a great deal of confusion between the terms *verification* and *validation*. Much effort is focused on verification (i.e., by testing), but little effort has been dedicated to validation to determine if the requirements are realistic, necessary, and sufficient in the first place. Sheng et al. have identified several major types of impediments in the way of an organized methodology for model validation, including definitional impediments and theoretical impediments.<sup>27</sup>

Definitionally, validation is still quite ambiguous, as illustrated above. Many organizations, such as the USNRC, the U.S. Department of Energy, and the International Atomic Energy Agency, offer definitions of validation similar to the definition of ‘model validation’ given above, but the model

validation definition captures all of the concepts included in the other definitions more succinctly. However, even in this definition there is some confusion as to what exactly is meant by ‘domain of applicability,’ ‘satisfactory range of accuracy,’ and ‘intended application.’ What these phrases entail can be largely determined by the purpose of the model. For example, if the purpose of the model is prediction of a system’s behavior, accuracy is important; if the purpose of the model is simply to gain insight into how the system works, accuracy is not so important.<sup>27</sup>

Theoretically, the basis of validation (using modeling and simulation) as a well-defined ‘science’ rather than as a subjective ‘art’ is young and not very strong. Sheng et al. say that this changeover has occurred only in the last fifteen to twenty years.<sup>27</sup>

### 2.3.3 V&V in BTP-14

In one sense, while V&V is just one aspect of BTP-14, execution of the review process described by BTP-14 *is* V&V on a large scale, not just of software but of a whole system. The overall purpose of BTP-14 is to guide the review of digital computer-based I&C systems in order to help ensure their reliable performance. This involves the validation and verification of the software and all other parts of the I&C system.

Specific places where V&V activities are mentioned within BTP-14 are (1) in the description of what should be contained in the SVVP (see Appendix E) and (2) in the description of what tasks are part of Software V&V Activities (see Figure 10). Analysis of these descriptions reveals many of the weaknesses of this process that have been discussed.

The guidance on the SVVP lists many general, high-level items that should be included, such as “descriptions” of the organization of the software V&V effort, responsibilities within the organization, the V&V schedule and required resources, V&V activities, V&V reporting requirements, tools and methods used for V&V tasks, and “how the V&V effort will be managed.” There must also be a “description of all required testing plans, specifications, procedures and cases.....[including unit, integration, system, and acceptance testing]” and a description of “test documentation requirements, evaluation criteria, error reporting, and anomaly resolution procedures.” For more information, the designer/reviewer is referred to the corresponding Reg. Guides and the standards they endorse: 1.1yy (and IEEE Std 1012) on V&V plans, 1.1xx (and IEEE Std 1008) on software unit testing, and 1.1vv (and IEEE Std 829) on test documentation. This provides a good example of the vague and unspecific nature of BTP-14 (and its associated Reg. Guides and standards).

The guidance on software V&V activities requires, among other things, that the different software development entities display the appropriate functional and development process characteristics; that problems be documented and actions be suggested and/or taken to fix them; that tests be described, executed, and documented in accordance with testing plans; and that the “result of each test should clearly show that the associated requirement has been met.” What is meant by the term ‘clearly show’?. Once



again, this guidance states the obvious of what needs to be done but offers no specific suggestions on how to do it. For any additional clarification the reader is again referred only to the guidance in NUREG/CR-6101.

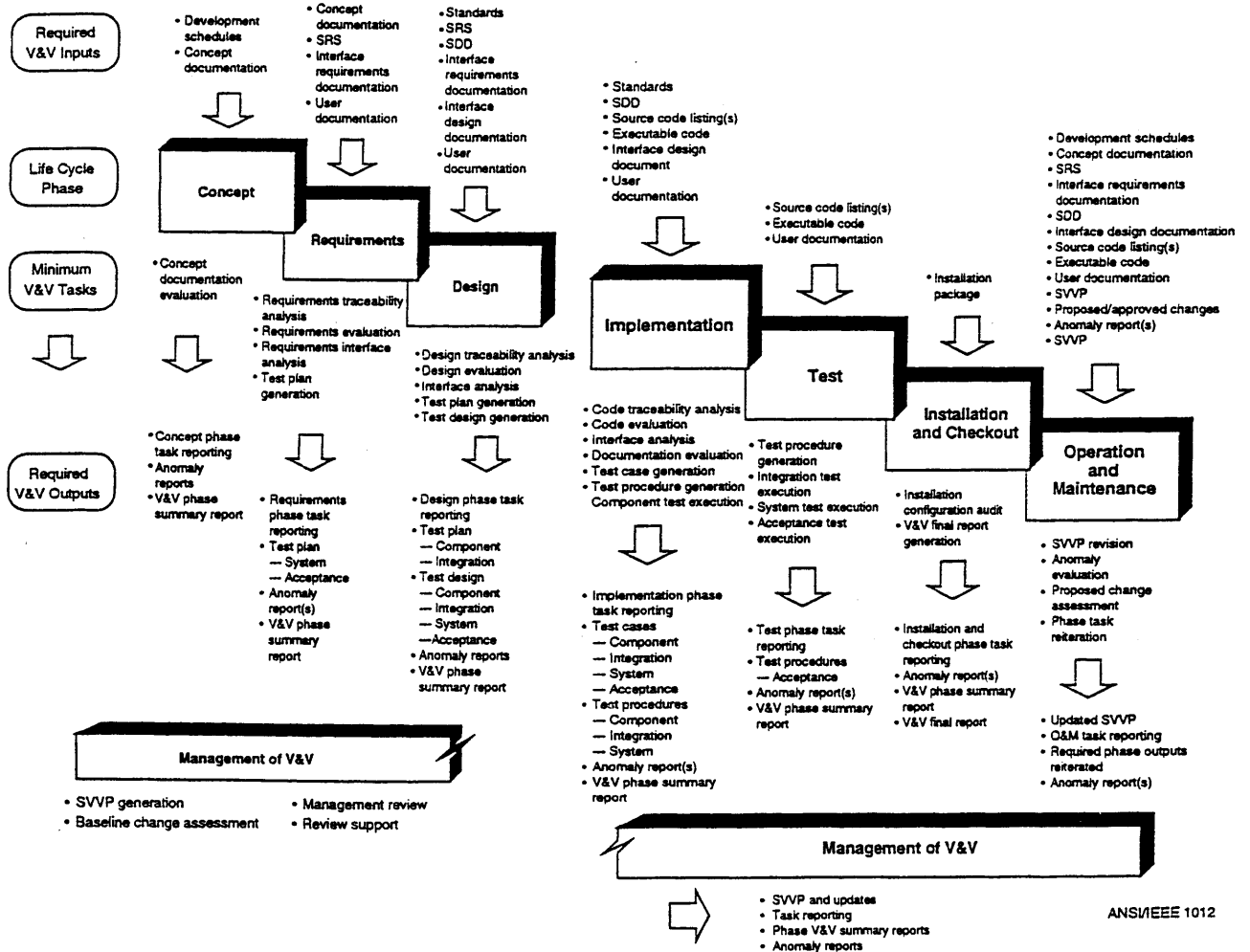


Figure 10 Verification and Validation Activities

(From NUREG-CR/6101, pp. 28-29)

### 2.3.4 V&V in the SRP

In SRP 7.0's big picture description of the steps in the overall I&C review process, V&V comes into play in the 'System Validation Evaluation' section of the guidelines for operating license and combined license applications. This evaluation is to be based on testing, analysis, and technical justification that the I&C system can perform its required safety functions, and on compliance with interface requirements and the design certification.

Appendix 7.0-A, which lists review topics specifically for *digital* I&C systems, directly mentions V&V in describing the review of the adequacy of the software life cycle process implementation, and it indirectly mentions it in describing the review of both the life cycle process planning and design outputs. Again, the guidance is very high level and does not give the developer or reviewer any idea of *how* to do what he is supposed to do. For instance, it is stipulated that the process planning should be done such that "the specified process and products, including design outputs, are designed to be inspected." It also says that the implementation "should be audited to confirm that the planned process is being implemented."

SRP Section 7.1 mentions V&V when describing its clarifications for review of *digital* I&C systems. It reiterates the need for verification and validation of requirements both at a system level and at each specific development stage. The user is told only that "implementation of a software engineering process as described by BTP HICB-14 will ensure adequate verification and validation" and that other guidance can be found in Reg. Guides 1.1xx, 1.1vv, and 1.1yy. As has been evident so many times before, this 'guidance' seems to consist only of references to other documents without ever resulting in a final answer.

### 2.3.5 V&V Standards: Reg. Guide 1.152 and IEEE 7-4.3.2

It was mentioned in Section 2.1 that, depending on the particular subject being analyzed, the reviewer would probably need to consult other forms of documentation besides the BTP and the SRP. For the subject of V&V, the other documents referred to include Reg. Guide 1.152, IEEE-7-4.3.2, Reg. Guide 1.1yy, IEEE 1012, and IEEE 1028.

The first document to consider in this series is Reg. Guide 1.152, an endorsement (with one exception) of IEEE 7-4.3.2, which was written to provide guidelines for satisfying the USNRC's requirements on "high functional reliability" and "design quality" for digital computers in safety systems of NPPs. Unfortunately, Reg. Guide 1.152 does not do much to clarify the stance on these topics; it primarily just restates a few important principles about them. For instance, Criterion III of 10 CFR 50 Appendix B, one of the basic requirements on which Reg. Guide 1.152 is based, gives the vague guidance that "quality standards be specified and that design control measures be provided for verifying or checking the adequacy of design." Reg. Guide 1.152 does little to expand on what exactly this entails.

The next document to consider after Reg. Guide 1.152 is IEEE 7-4.3.2, which clarifies the use of IEEE 603 for digital applications. One of the statements in this regulatory guide, as described earlier, says

that V&V “shall provide *adequate confidence* that the safety system requirements.....at each stage of development.....have been implemented [with respect to] computer software and hardware, noncomputer hardware, and the integration of these items.” Though the term ‘adequate confidence’ is standard regulatory jargon, what exactly does it mean here? Is it reflected in a certain number of correct test runs, by a completed check-off list of safety system requirement implementation methods, by whatever the software developer deems appropriate, or by some other method?

This standard goes on to say that V&V plans shall “confirm the correctness and completeness of the design [and] shall specify activities and tests that shall be [executed] by competent [authorities].....” *How* do they confirm the correctness and completeness of the design? Perhaps some specific types of activities and tests should be described rather than letting the developer pick and choose any method he wants. In this way the reviewer would know exactly what to look for rather than have to try to figure out the developer’s thought process and methodology.

Most of the detail on V&V in IEEE 7-4.3.2-1993 is found in Annex E, which is not officially part of the standard, but is rather a list of suggestions deemed acceptable by those who developed the standard. Annex E stresses functional diversity, D-in-D&D, and design diversity.

One of the topics discussed by this annex is the use of “black box” testing to test program module input/output correctness and “white box” testing to examine more closely the internal program details (the annex defines white box testing in essentially the same way that branch- or path-coverage testing is usually defined). The use of Failure Modes and Effects Analyses (FMEAs) to detect failure modes, design-basis events, and abnormal conditions & events, from which lower level hardware and software requirements can be derived, is also discussed. Some guidance on what issues to consider when carrying out these analyses is offered in other references as well, but again there is little advice on *how* to do the analyses. While this annex is more detailed than much of the other guidance used in this process, it still leaves the user in search of more detail and in need of other documents.

### 2.3.6 V&V Standards: IEEE 1012, IEEE 1028, and Reg. Guide 1.1yy

Reg. Guide 1.1yy (on V&V, reviews, and audits) serves as an example of Reg. Guides 1.1uu through 1.1zz, since they are all quite similar (and generally not of much benefit for the review). The basic purpose of these Reg. Guides is to offer endorsements of various standards pertinent to their particular topics. For Reg. Guide 1.1yy, the topic is V&V, reviews, and audits, and the corresponding standards are IEEE 1012 and 1028. There are some exceptions taken and some modifications made, but many of these changes deal with surface details rather than substance.

It is interesting to note, however, that hidden inconspicuously in the midst of the guidance of Reg. Guide 1.1yy is an important addition to the configuration management stipulations of IEEE 1012-1986 (paragraph 3.7.4). It says that “any V&V materials necessary for the verification of the effectiveness of the V&V programs or to furnish evidence of activities affecting quality must be maintained as quality

assurance records. Those materials necessary for the re-verification of changes must be maintained under configuration management.” In other words, any materials necessary to properly make modifications to safety-related software must be maintained very carefully under configuration management (in addition to all of the configuration management contents required by IEEE 1012).

This is an important point that should be highlighted more prominently than it is. Not heeding this requirement can have disastrous consequences, as demonstrated by the Therac-25 debacle discussed in Appendix P. (In the Therac-25 case, an improper implementation of ‘improved’ software technology to replace a hardware safety mechanism in the previous model of the Therac radiation administration device led to a new, “upgraded” machine that was, in fact, faulty and led to many injuries and even death. The software as originally modified was incapable of performing the same safety function that the hardware had performed on the previous Therac machine. Better configuration management -- in this case, more careful management of changes to the software -- could have prevented this problem from ever happening. As stated by Leveson and Turner, “...not only must safety be considered in the initial design of the software and its operator interface, but the reasons for design decisions should be recorded so that decisions are not inadvertently undone in future modifications.”<sup>28</sup>)

IEEE 1012 and IEEE 1028, both endorsed by Reg. Guide 1.1yy, come next in the analysis. IEEE 1012, which is referenced by almost every document that deals with V&V topics, is the closest thing to an authoritative standard on V&V that can be found. Yet even it avoids the issue of offering advice on *how* to do the things that it recommends software developers should do, and even it does not stand alone (it is to be used in conjunction with four other IEEE standards). For example, in explaining what to look for in a Software Requirements Evaluation, it says the requirements should demonstrate “correctness, consistency, completeness, accuracy, readability and testability.” This is very similar to the stipulations of BTP-14, but no more explanation of what is meant by these terms or how to demonstrate them is given here than is given in the BTP. In fact, though it is not much more helpful, there is actually more clarification in the BTP!

IEEE 1028 is equally unhelpful. It provides common-sense criteria on the different reviews and audits that it discusses but little else. For instance, it says that a software walkthrough is complete is when all parts of the software have been “walked through” in detail, all problems and suggested improvements have been noted, and the walkthrough report has been issued.

### **2.3.7 Testing and V&V in NUREG/CR-6101**

NUREG/CR-6101, which forms the basis for most of BTP-14, lists activities, documents, and recommendations for each of the eight life cycle phases. For a listing of those dealing with testing and V&V, see Figure 10. As described about much of the other guidance so far, this information amounts to little more than a laundry list of items to do, with little or no advice on how to go about doing them.

NUREG/CR-6101 also contains appendices with “information on certain technical issues that are pertinent” to it, including various “techniques that may be used to model reliability in general, and software reliability in particular” and several software reliability growth models. The reliability analysis and modeling techniques discussed include reliability block diagrams, fault tree analyses (FTA), event tree analyses, failure modes and effects analyses (FMEA), Markov models, and Petri net models. The reliability growth models discussed include the Duane model, the Musa model, the Littlewood model, and the Musa-Okumoto model.

Basic equations for these methods are discussed along with certain assumptions, strengths and limitations. For example, the Musa reliability growth model makes the unrealistic assumption that all program ‘bugs’ are equally likely to occur, while the other reliability growth models give a heavier weighting to errors found early on (since they tend to be the more common and frequent errors). In general, less confidence is expressed in the reliability growth models than in the reliability analysis models. Such methods (especially the reliability growth methods) are not endorsed for determining failure rates lower than  $10^{-4}$  due to the amount of time necessary to demonstrate such high reliability. The particular examples discussed seem to form a rather hit-or-miss selection, and the usefulness of many of them (especially the reliability growth models) is, at best, debatable.

### 2.3.8 Methods and Types of Testing

There are two broad categories of software testing referred to as “black box” (functional) and “white box” (structural) testing. These were touched on in Section 2.3.5. Black box testing implies looking only at the output of a program in order to determine if it is correct based on the input. This determines the correctness of the program from the *end-user’s* viewpoint. What goes on in the internal parts of the program is not of interest. In white box testing, the concern is with what goes on internally in the program. This determines the correctness of the program from the *developer’s* viewpoint.<sup>29</sup> In other words, if the program comes up with a correct output, but the output is a result of some faulty sequence of logic through the program, then the result is unsatisfactory. (For example, a program with a binary output could very easily show the correct output based on incorrect computations, because there are only two possible outputs.) White box testing requires much more in-depth knowledge of how the program works than black box testing does.

NUREG/CR-6421, “A Proposed Acceptance Process for Commercial-Off-the-Shelf Software in Reactor Applications,” provides examples of six major types of testing along with some of their advantages and disadvantages. The types described (with some examples) are:<sup>30</sup>

1. *Static source code analysis*
  - a. *Inspections* -- including code inspection, peer reviews, and walkthroughs
  - b. *Desk checking* -- stepping through code (usually by one programmer) to double check computations for faults

- c. *Automated structural analysis* -- use of an automated checker to find to find data and logic structure faults
  - d. *Other methods deemed impractical at this point* -- e.g., proof of correctness and similar (formal) methods
2. *Structural (or "white box" or "glass box") testing* -- of the software internal code
    - a. *Control flowgraphs*
    - b. *Control flow (path) testing*
    - c. *Loop testing*
    - d. *Data-flow testing*
  3. *Functional testing*
    - a. *Transaction testing* -- similar to control flow testing, but at a higher than modular level
    - b. *Domain testing* -- of input values
    - c. *Syntax testing* -- of operators, etc.
    - d. *Logic-based testing* -- somewhat similar to the tabular representations of requirements endorsed by Ontario Hydro [OH] and Atomic Energy Canada, Ltd. [AECL] in their software development process (see Section 3.2.3)
    - e. *State testing* -- based on expected state-transitions
  4. *Statistical testing* -- to predict reliability rather than to discover faults
    - \* This testing must be done based on a realistic operational profile, again as emphasized by OH and AECL in their software development process.
    - \*\* *Ontario Hydro decides the number of statistical (random) test cases n to use based on the formula  $n = \lceil \log(1-c)/\log(1-f) \rceil$ , where f is the maximum failure rate allowed and c is the confidence level required. It must be pointed out that the validity of using this formula for non-Bernoulli trials [i.e., trials that are not all identical] is questionable. The trials OH uses each model one of six possible accident sequences that their shutdown systems are designed to protect against, so each trial is not exactly the same as in Bernoulli trials.*
  5. *Stress testing* -- to test the system at loads outside of normal expected limits (i.e., if the system performs well outside of normal limits, confidence is increased in its performance within normal limits)
  6. *Regression testing* -- rerunning of original tests after a change is made to the program or system to ensure that no new errors or unexpected effects have been introduced

Because these types vary so much among their prerequisites and their applications, some mix of them would most likely be the ideal composition for a testing regimen. (These testing types are described for application to pre-existing software products and do not take into account the quality of the software developer or the development process used.)

This document also provides tables listing many different software qualities, their impact on safety from a regulator's viewpoint, the appropriate types of testing to use to evaluate them, prerequisites for and recommended extent of testing types, and typical testing strategies. (See Appendices L, M, N, and O). It is odd that this document, with much more detailed information than many of the others used in the BTP-14/SRP Chapter 7, is mentioned only in passing in Appendices B & C of SRP Section 7.1. Though its

guidance is specifically for COTS software, it could certainly provide some worthwhile momentum to an overall software development process review and V&V effort.

### 2.3.9 What is the Right Mix of Testing Methods?

NUREG/CR-6263 discusses what mix of testing is ideal. There are advocates for many different combinations. IEEE 7-4.3.2 says only that there should be some mix of functional and structural testing, although most recommendations include some mix of functional testing (with respect to requirements compliance), structural testing (with respect to path or state coverage), *and* statistical testing. Statistical (or 'operational profile,' or 'random') testing with many different combinations of input variables can be used as an aid in functional testing. NUREG/CR-6263 adds that test cases should be based mostly on *requirements* and should aim to verify the program's functionality. After testing, it says it should also be determined which requirements were *not* tested.

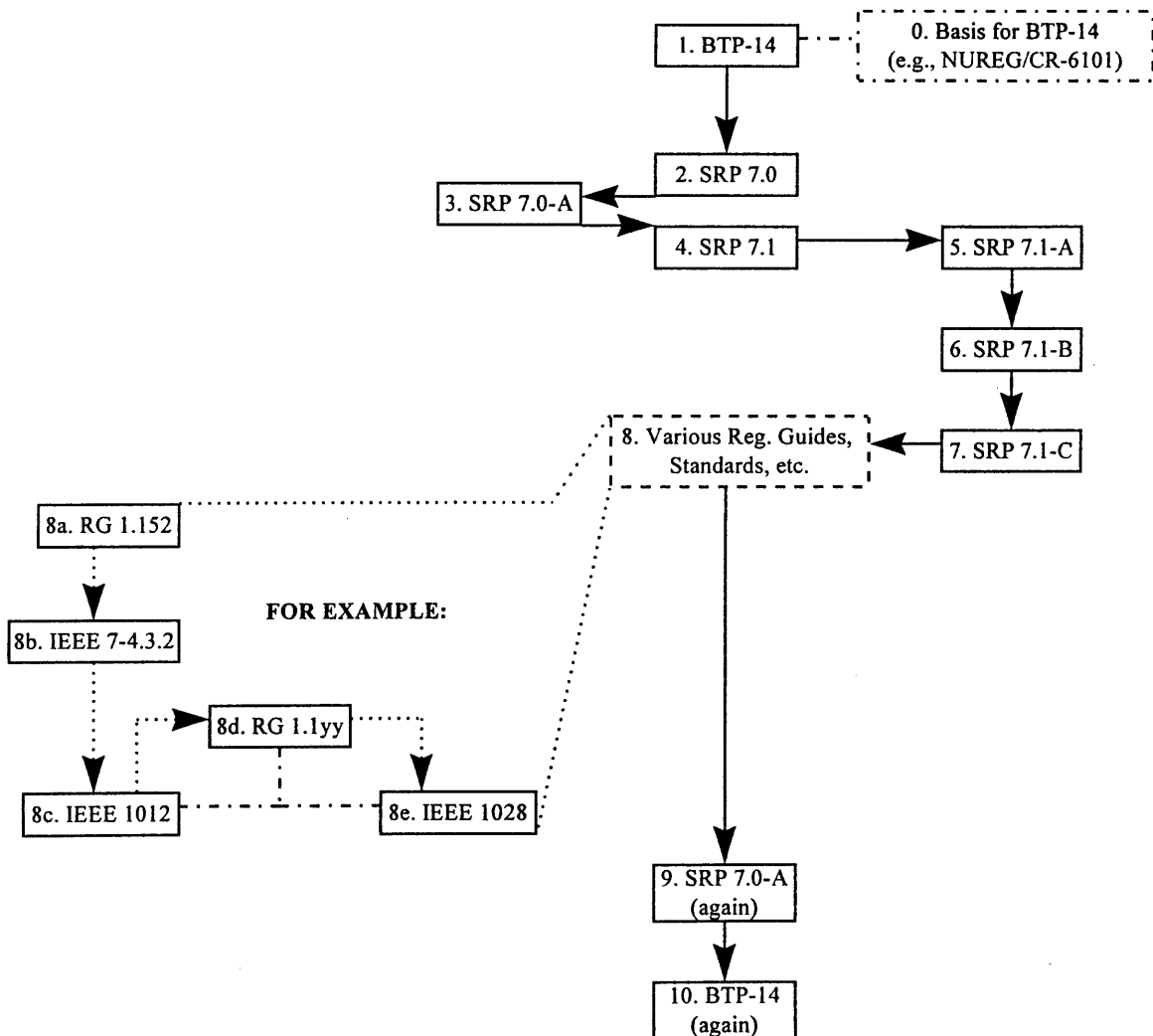
### 2.3.10 Problems with Testing

A major problem associated with testing is knowing how much is necessary. Various methods have been proposed, such as choosing test cases based on the software's input profile and then recording the execution time between failures until the desired failure-intensity level is reached.<sup>31</sup> Some suggest that testing is ineffective because faults will continue to show up throughout the software's life, and that, since an absence of faults cannot be *guaranteed*, an acceptable risk level must be determined.<sup>32</sup>

Another testing problem is the tendency to concentrate on testing only near the end of the life cycle. One reason for this problem is that testing cannot be performed until the code has been written. A major goal of V&V is to examine software from a systems viewpoint to find 'high-risk errors' (i.e., errors that would lead to safety or security hazards or large financial or social losses) *early* in the design process, allowing for incremental changes throughout the process.<sup>33,34</sup> Something other than standard testing needs to be done to achieve this, however.







**Figure 11** Illustration of BTP-14 Review Path Discussed in Chapters 2 & 3  
(continued on next page)

0. (6101 mentions such basic questions as: (1) Is each requirement for each mode identified, traceable to an identified hazard, consistent, stated numerically, and verifiable? (2) Are the functional requirements complete, consistent, unambiguous, verifiable, and traceable? (3) Do the software requirements specify what the software must *and must not* do?)

1. **BTP-14:** discusses its 3 *purposes* (evaluate software lifecycle process *planning, implementation, and documentation*) and basic *software qualities*
2. **SRP 7.0:** discusses *review points for digital (reactor) I&C designs and construction/operation permits* (functional and technical compliance with design criteria and bases of 10 CFR 50, tech specs and proper safety margin, evidence of design process characteristics in outputs)
  - "...Review of computer-based systems should focus on confirming the *acceptability and correct implementation of the life-cycle activities*." (p. 7.0-2)
3. **SRP 7.0-A:** *supplements 7.0 for digital systems* on the topics of design qualification, common-mode failure, and functional requirements that are particularly challenging for digital systems
4. **SRP 7.1:** provides the *regulatory basis* (in IEEE 603, Reg. Guide [RG] 1.153, USNRC General Design Criteria [GDC], & 10 CFR 50) of certain rules for I&C systems important to safety and provides *clarifications to 7 topics important in digital systems* (electromagnetic capability, computer system quality [software/hardware development and integration, software tools, V&V, configuration management, qualification of existing computers, equipment qualification, system integrity and design for test and calibration, communications independence, reliability, and defense against common-mode failure)
  - There should be a "*well-structured and well-executed software engineering process*" (p. 7.1-4)
5. **SRP 7.1-A:** *descriptions of the acceptance criteria and guidelines*
  - "...systems...must be [designed, constructed, tested, and inspected] to quality standards commensurate with the importance of the safety function to be performed." (p. 7.1-A-1)
  - "[reactor license applications must have] detail sufficient to ... judge the...[means of conforming to the design] and to reach a final conclusion on all safety questions]. ...[must be] *sufficiently detailed* [for the NRC to prepare] acceptance requirements." (p. 7.1-A-8)
  - *many references to other documents*

6,7. **SRP 7.1-B,C:** *digital clarifications* for IEEE 279 & 603 -- e.g., identification of safety margins and conditions requiring protective actions
 

- "...degree of redundancy, diversity, testability, and quality [should be] *adequate* [for] reliability commensurate with the safety functions to be performed."

8a. **RG 1.152:** supplements various GDC and reiterates the need for "high functional reliability" and "design quality"

8b. **IEEE 7-4.3.2:** endorsed by RG 1.152 -- adds more clarifications for digital topics -- e.g., "[V&V] shall provide adequate confidence that ...requirements...have been implemented...[and] a V&V plan be prepared to confirm the correctness and completeness of the design[via activities and tests done by independent reviewers.]"

8c. **IEEE 1012:** endorsed by RG 1.1yy -- describes required & optional V&V tasks -- e.g., the SRS should: evaluate requirements for "correctness, consistency, completeness, accuracy, readability, and testability" and assess "how well SRS satisfies software system objectives" and "the criticality of requirements to identify key performance or critical areas of software" -- required inputs & outputs to the software requirements are also listed

8d. **RG 1.1yy:** endorses IEEE 1012 & 1028, but adds little to them; explains (again) the bases for the policies in these documents (from 10 CFR 50, GDC, etc.)

8e. **IEEE 1028:** general definitions and requirements for reviews and audits

9. **SRP 7.0-A (again):** specific digital topics elaborated on are: adequacy of design criteria, identification of review topics, defense-in-depth & diversity, life cycle process planning, adequacy of system functional requirements, and adequacy of life cycle process implementation & design outputs

10. **BTP-14 (again):** the software life cycle and its activity groups are described; required contents and software development characteristics are described for various documents and activities, including the Software Quality Assurance Plan, Software Safety Plan, Software V&V Plan (and related V&V activities), the SRS, Code Listings, System Build Documents, etc.



---

## 3 Improving Software Design and the BTP-14 Review Process

---

### 3.1 General Observations on BTP-14 and the Ontario Hydro Software Development Standard

First it should be said that, as was explained earlier, BTP-14 is as good as or better than the other BTPs used in this I&C review process. It does, however, have aspects that could be improved (each of which will be discussed in Sections 3.2.1 through 3.2.5):

1. Its main points and guiding principles are not expressed clearly and concisely. They are mentioned piecemeal throughout the body of the BTP (and its many associated regulatory and supporting documents).
2. The organization of the regulatory documents and the guidance provided in the documents is complex and more tedious than necessary.
3. Most of the guidance provided is very vague.
4. While various methods of hazard identification are discussed, *system*-level hazard identification and system safety principles in general should be stressed more heavily (because many problems arise as a result of integration of system parts).
5. There are various problems with testing in the software development and review process as they are described now. Much of the testing is done rather late in the process, and the guidance on the types of testing (and mix of types) that should be used is not very helpful.

Many of the deficiencies with BTP-14 are not of content but rather of form and organization. By comparison of BTP-14 with some of the existing software development methodologies already discussed and with an actual implemented software development standard (“CE-1001-STD Rev. 1,” used by Ontario Hydro [OH] and Atomic Energy Canada, Ltd. [AECL] in licensing the digital safety shutdown systems of the CANDU reactors at Darlington Nuclear Generating Station), it will be possible to suggest some worthwhile refinements to the process proposed in BTP-14.<sup>35</sup>

As mentioned earlier, OH and AECL have already implemented a successful software development standard via licensing of the Darlington safety shutdown systems. Many of this methodology’s major concepts are the same as those of BTP-14 (and SRP Chapter 7) but are implemented in a simpler and more organized fashion. This software review standard was created by collating many existing standards into a single standard to be used throughout the entire software life cycle. Different

safety categories are assigned to software products of different safety importance (a graded approach).<sup>36</sup> The entire process is based on minimum required activities and documentation, just as BTP-14 is.

OH says in its standard that a “software product is considered acceptable if it is shown to satisfy a set of quality objectives,” which are “represented by a longer set of quality attributes” that are “built into the product by adhering to a software engineering process governed by a set of fundamental principles.”<sup>35</sup> These “quality attributes” are in many cases the same as the characteristic qualities used in BTP-14 -- completeness, correctness, etc. And the fact that these quality attributes are a result of “adhering to a software engineering process governed by a set of fundamental principles” is reminiscent of the description of the “software development process characteristics” of BTP-14, which are said to be an “artifact” of a “disciplined design process.” The OH standard goes on to say that “*measurable requirements* are derived from the quality attributes and the fundamental principles...” Some of the fundamental principles in the OH standard are very similar to some of the basic BTP-14 concepts. Take the following OH principles, for instance:<sup>35</sup>

1. A planned and systematic Software Engineering Process must be followed over the entire life cycle of the software (i.e., same degree of rigor applied to software revisions as to the original software development).
2. Verification of the software must be carried out throughout its entire life. All changes to an output must be verified in the same way as the original output.
3. Independence of design and verification personnel must be maintained to help ensure an unbiased verification process.
4. Analyses must be performed to identify and evaluate safety hazards associated with the computer system with the aim of either eliminating them or assisting in the reduction of any associated risks to an acceptable level.
5. Configuration management must be maintained throughout the entire life of the software to ensure up-to-date and consistent software and documentation.
6. Audits must be performed periodically to ensure that the software and all development-related processes conform to standards and procedures.
7. Ongoing training must be undertaken to ensure that personnel have the skills required to perform their jobs.

More importantly, however, are some of the other principles which are better, or more clearly defined. For instance:<sup>35</sup>

1. Documentation must be prepared to clearly describe the required behavior of the software *using mathematical functions* written in a notation which has a *well defined syntax and semantics*. (emphasis added)
2. Outputs of each development process must be verified against the requirement inputs. In particular, outputs written using mathematical functions must be

systematically verified against the inputs using *mathematical verification techniques* or *rigorous arguments of correctness*. (emphasis added)

3. The structure of the software must be based on '*Information Hiding*' concepts. [Identifying requirements likely to change in the future, which provides a basis for information hiding, is another aspect of *completeness* -- again a clearer explanation than that in BTP-14.]
4. *Both systematic and random testing* must be performed to ensure adequate test coverage. (emphasis added)
5. Reliability of the safety critical software must be demonstrated using *statistically valid, trajectory-based, random testing*.

### 3.1.1 Mathematical Notation

The way that OH implemented a mathematical notation to describe software behavior is with a 'tabular representation' using tables composed of condition statements, action statements, and rule columns. (See the example in Figure 12.) These tables have several advantages: they can cover all ranges of the input domain; they imply what to do but not how to do it; they can be derived from requirements or reconstructed from code (good for traceability); they facilitate mathematical comparison with the requirements; and they can be understood by (nuclear) domain experts, not just software or mathematics experts. This last advantage contributes to 'understandability' more concretely than any connection drawn in BTP-14. The overall requirement for a mathematical notation facilitates *verifiability* and *completeness*, again more clearly than any suggestions given in BTP-14.<sup>37</sup>

### 3.1.2 Mathematical Verification

Mathematical verification techniques imply that the requirements, design description, and code must all express their behavior (at either a system or program level) as a mathematical function relating the inputs and outputs.<sup>37</sup> This allows for a 'step-wise refinement' process of evaluation, first from requirements to design and then from design to code.

### 3.1.3 Information Hiding

'Information hiding' is a software design methodology based on data handling needs rather than on program functionality. Modules are designed around the data that they require rather than around the functions that the program is to perform. Data access is restricted only to what is absolutely necessary, and modules are only loosely coupled (i.e., there is restricted information sharing between them). This methodology was developed to prevent the problems that arise when information is shared -- for example, when the basic data structures of a subprogram are inadvertently changed because they are accessed by a different subprogram.<sup>38</sup>

Using information hiding, a developer can change one module without worrying about how the change will affect all the others.<sup>36</sup> This results in lower costs not only for development but also for

CONDITION STATEMENTS	1	2	3	4	5	6	7	8	9	10	11
m_up_pb = pressed	F	T	F	F	F	T	T	T	-	-	-
m_down_pb = pressed	F	F	T	F	F	T	-	-	T	T	-
m_set_pb = pressed	F	F	F	T	F	F	T	-	T	-	T
m_enter_pb = pressed	F	F	F	F	T	F	-	T	-	T	T
ACTION STATEMENTS											
f_request = no_request	X										
f_request = up_request		X									
f_request = down_request			X								
f_request = set_request				X							
f_request = enter_request					X						
f_request = wd_test_request						X					
f_request = error_request							X	X	X	X	X

Figure 12 Sample of Tabular Representation

(From "Tabular Representation of Mathematical Functions for the Specification and Verification of Safety Critical Software" by J. McDougall, M. Viola, and G. Moum)



maintenance, because each software module can be designed, implemented, and revised independently (i.e., the software has reusability). Shumate and Keller cite a 1972 paper by Parnas in which he says that “the [software] specification must provide to the intended user all the information that he will need to use nothing more” part of that explanation. In another paper they cite, Parnas describes the term ‘information hiding’ by saying, “One begins with a list of difficult decisions which are likely to change. Each module is then designed to hide such a decision from the others.”<sup>37</sup>

Techniques such as information hiding help to promote simplicity of design. The issue of complexity versus simplicity is important not only in the regulation of software development but also during the software development process. Much time and effort is devoted to this topic. Leveson discusses the issues of complexity and coupling as the two principal causes of serious accidents in software safety systems. Unfortunately, the ease of making changes to software (as opposed to hardware) introduces too much change, from which follows more complexity and error.<sup>20</sup> Indeed, unforeseen complex interactions are a primary concern of people involved with creating and certifying ultra-high reliability software. Even in relatively simple programs, state-space explosion can quickly become burdensome; in complex programs it can far exceed any current capabilities to handle it.

What is often forgotten when considering potential complexity problems, however, is that, as stated in the May 2, 1996, ACRS memo previously mentioned, we need to “emphasize more strongly that safety systems in nuclear power plants, whether they are digital or analog, should be simple and separated from all other control systems which may be significantly more complex.”<sup>16</sup> Parnas et al. say that “[if] software is to play a role critical to safety, it is essential to restrict the complexity to allow complete understanding and thorough analysis.”<sup>39</sup> Many of the methods currently used to make digital I&C systems safer are somewhat counterproductive. For instance, adding redundancy and/or diversity to a program also adds complexity to it by definition. Perhaps software developers need to concentrate more on making their designs as simple as possible in the first place to avoid the need for such measures.

#### **3.1.4 Testing**

Testing in the OH process requires a mix of systematic and random testing. The systematic testing (both black box and white box) is used to find many common errors. Since testing is typically not complete, an adequate set of systematic tests is then reinforced with random tests to acquire some level of confidence that the software will not encounter some unforeseen set of inputs that will cause it to fail. The random tests must be based on expected input distributions, expected accident scenarios within an expected time frame, and expected output distributions.<sup>16</sup>

## 3.2 Recommended Actions to Improve the BTP-14 Review Process

### 3.2.1 Recommendation #1

The major guiding principles of BTP-14 should be assembled and clearly specified in one location at the beginning of any and all of the documents used in this regulatory process. This is done in the standard used by OH and AECL in their software development regulatory process.

### 3.2.2 Recommendation #2

BTP-14 (and its associated documents) should be written in such a way that they can stand alone. The documents in this regulatory process are plagued by references and cross-references from one document to another. The user is hard-pressed to find a single source-level document on most topics. Perhaps there should first be a brief listing of the steps involved in the review process *without explanation*, and then a detailed summary of the steps including explanations and references to needed source documents. The OH/AECL standard is also based on many other documents, but it says what it needs to say without making the reviewer search through all of them.

If it is not possible to write the document(s) in such a way, then there should at least be some type of detailed guidance provided to map the path that the user must follow through all the documents. There should also be a description of *exactly* which parts of which documents are needed as references at each place in the regulatory process where such references occur.

### 3.2.3 Recommendation #3

Specific and, preferably, measurable requirements should be placed on software qualities and properties in order to avoid being vague. OH/AECL accomplish this by mandating that requirements be expressed in a mathematical notation with a well-defined syntax and semantics. This allows for mathematical verification of the requirements. They implement such a requirements notation by way of the 'tabular representation' advocated by Parnas (illustrated in Figure 12). Parnas describes the advantages of a tabular representation, the primary one being that it "parses the expression for the reader; many nested pairs of parentheses are eliminated."<sup>39</sup> Though some desirable software traits may not yet be understood well enough for such a well-defined description, thinking about each of the software requirements from a tabular framework may at least elucidate to the designer what aspects of the requirements are most important to consider. If nothing else, the designer is forced to consider the program more deeply than he might otherwise.

It is not expected that a regulatory authority can develop an *exhaustive* list of specific guidelines; there is too much variability from one designer to another and from one software firm to another to cover

every possible detail. At the same time, however, in the various review process documents considered in this work, there were numerous examples of guidance that gives designers no direction. This can ultimately make the reviewer's job a slower, case-by-case process, which is what the USNRC wants to avoid. Use of a well-defined mathematical representation removes ambiguity from requirements, and use of a straightforward expression of requirements (such as the tabular representation described above) makes them understandable to all parties involved with the software development, regardless of their particular application domain knowledge (in this case, of the reactor engineering field).

#### 3.2.4 Recommendation #4

System safety approaches in the software development process must be emphasized more strongly. This should include the use of any or all of the methods for this purpose that are addressed in this work. For instance, the use of checklists and/or HAZOP can be helpful in identifying and accounting for system-level hazards. HAZOP can also be used effectively in combination with FMEAs. DFM could also be very useful in the area of system safety, because it is intended to provide structural models of the software (from a system level) from which the existence of failure modes, ACEs, and other such system problems can be determined. Use of a programming technique such as information hiding (see the discussion in Section 3.1.3) could also be beneficial in focusing on the software project from a system-level perspective.

#### 3.2.5 Recommendation #5

A complementary mix of all three major types of testing -- functional, structural, and random -- must be used in order to capitalize on the advantages of each of the different methods. In addition, testing must be employed as early in the software development process as possible (a Cleanroom software engineering approach) to minimize the costs of fixes for problems uncovered.

Numerous problems were discussed with the testing methodology currently in place in the BTP-14 software development review process. Various testing strategies are mentioned and briefly described in the BTP and its associated documents, but little detail is given about how to execute any of them. The OH/AECL standard gives much clearer guidance by requiring a mix of *systematic* testing (both black box [functional] and white box [structural]) with statistically-valid, trajectory-based *random* testing. Though the applicability of the formula OH uses to find the required number of random tests for a particular confidence level is questionable, OH does not rely solely on these tests. The random tests are used to "[compensate] for false assumptions and biases of the tester" which influence the design of the systematic tests.<sup>36</sup> These tests are used to *quantify* the uncertainty of the systematic test results so that compliance with a specific reliability value may be demonstrated. This combination of testing methods increases overall confidence in the testing results.

In addition, most testing currently done occurs late in the life cycle. It is important to build testing into the entire life cycle from the very beginning. Hardware manufacturers use such a strategy; they test

products at critical points throughout the development process in order to catch problems early. Such testing serves the purpose of ‘statistical quality control,’ which will be explained below. In software, this allows for functional verification at each step of the design process (i.e., requirements to design, design to coding, etc.) Compare this concept of functional verification to the definition of ‘formal verification’ in Section 2.3.1.<sup>22</sup> These concepts are part of the Cleanroom engineering process described in Section 1.3.4. This methodology, used by many industries today, is based on numerous principles which could be of benefit to the software development process. Another option which can be useful in heading off software errors before they ever happen is the Computer Aided Software Engineering (CASE) technology discussed in Section 1.3.4.2, which allows the software designer to work through the requirements and design phases graphically and sometimes even takes care of the implementation (coding) details automatically.

It is interesting to look back at the SEI’s Capability Maturity Model with the benefits of Cleanroom software engineering in mind. By requiring some of the Cleanroom techniques, BTP-14 could add some of the elusive Level 4 and Level 5 characteristics to the software development process, such as the following:<sup>24</sup>

- \* A measured process,
- \* Improvement fed back into the process,
- \* Data gathering used to identify the weakest process elements,
- \* Numerical evidence used to justify application of technology to critical tasks, and
- \* Rigorous defect-cause analysis and defect prevention.

---

## 4 Conclusions

---

Five major areas for improvement of the Branch Technical Position HICB-14 ('BTP-14') software development life cycle review process have been discussed. These five points, followed by recommended changes or improvements (in italics), are:

1. Its main points and major guiding principles are not expressed clearly and concisely. They are mentioned piecemeal throughout the body of the BTP (and its many associated regulatory and supporting documents).
  - > *The major guiding principles of BTP-14 should be assembled and clearly specified in one location at the beginning of any and all of the documents used in this regulatory process.*
2. The organization of the regulatory documents and the guidance provided in the documents is too complex.
  - > *BTP-14 and related regulatory documents should be organized more clearly, either by combining all necessary information into a single, stand-alone document, or by providing some type of roadmap to guide the user through the various documents with minimum effort and confusion. The guidance itself should emphasize techniques that minimize complexity, perhaps including such methods as information hiding.*
3. Most of the guidance provided is very vague.
  - > *A more quantitative nature might be added to the guidance by including a requirement like that of the Ontario Hydro / Atomic Energy Canada, Ltd. standard, which calls for expression of all requirements in a mathematical notation (in particular, a tabular representation).*
4. While various methods of hazard identification are discussed, system-level hazard identification and system safety principles in general should be stressed more heavily (because many problems arise as a result of integration of system parts).
  - > *Various techniques are proposed to address system-level safety issues, including checklists, HAZOP, DFM (Dynamic Flowgraph Methodology), and the use of information hiding techniques.*
5. There are various problems with testing in the software development and review processes as they are now implemented. Much of the testing is done rather late in the process, and the guidance on the types of testing (and mix of types) that should be used is not very helpful.
  - > *Major recommendations made with regard to testing include the use of a mix of testing types (functional, structural, and random), a shift to earlier testing and testing at critical points between stages of the software development ('statistical quality [or process] control'), and a corresponding emphasis on error prevention rather than error detection, perhaps including the use of*

*CASE tools. The Cleanroom concept stresses these methods and offers such benefits as simplicity of design, (mathematical) functional verification, and a distinct absence of debugging.*

The changes recommended for the BTP-14 software development review process are for the most part not changes of content, but rather changes of presentation. The BTP-14 process is based on many of the principles that constitute the basis of the conventional wisdom on software development. However, the presentation of the process and its requirements can be done much more clearly and efficiently than it is in BTP-14, as illustrated by the corresponding standard developed by Ontario Hydro and AECL. Combined with their experiences in producing a software development standard and with some of the current successful approaches to software development discussed above, BTP-14 can be much more effective in helping both the software developer and the reviewer.

---

## References

---

1. Draft Standard Review Plan (Appendix 7.0-A -- "Review Process for Digital Instrumentation and Control Systems," Version 7.0, August 23, 1996).
2. (Proposed) Branch Technical Position HICB-14: "Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems" (Version 10.0), August 23, 1996.
3. National Academy of Sciences (Committee on Application of Digital Instrumentation and Control Systems to Nuclear Power Plant Operations and Safety; Douglas M. Chapin, Chair). *Digital Instrumentation and Control Systems in Nuclear Power Plants*, National Academy Press, Washington, D.C., 1995.
4. Thadani, Ashok C. and Robert L. Perch. "Consideration of Important Technical Issues for Advanced Light Water Reactors," *Proceedings of the 2nd ASME-JSME Nuclear Engineering Joint Conference*, Vol. 2 (ed. Per F. Peterson), 1993: 553-558.
5. NASA-GB-1740.13-96: *NASA Guidebook for Safety Analysis and Development*, NASA (Washington, D.C.), April 1996.
6. Leveson and Knight. "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming," *IEEE Transactions on Software Engineering*, 12(1): 96-109.
7. Littlewood, Bev. "The Impact of Diversity upon Common Mode Failures," *Reliability Engineering and System Safety*, 101-115.
8. Kletz, Trevor. *HAZOP and HAZAN: Identifying and Assessing Process Industry Hazards*, Hemisphere Publishing Corporation, Bristol, PA, 1992.
9. Redmill, F., M. F. Chudleigh, and J.R. Catmur. "Principles Underlying a Guideline for Applying HAZOP to Programmable Electronic Systems," *Reliability Engineering and System Safety*, edition to be published.
10. Leveson, Nancy G. *Safeware: System Safety and Computers*, Addison-Wesley Publishing Company (Reading, MA), 1995.
11. Lutz, Robyn. "Targeting Safety-Related Errors During Software Requirements Analysis," *Software Engineering Notes*, 18(5), December 1993: 99-105. (from SIGSOFT '93, Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering; Los Angeles, CA, December 7-10, 1993)
12. Seth, S. et al. NUREG/CR-6263: *High Integrity Software for Nuclear Power Plants*, The MITRE Corporation, June 1995.
13. Lawrence, J.D. NUREG/CP-0145: *Workshop on Developing Safe Software*, Lawrence Livermore National Laboratory, November 1994.
14. Christou, Aris. *Integrating Reliability into Microelectronics Manufacturing*, John Wiley & Sons, Ltd., New York, NY, 1994.

15. Dyer, Michael. *The Cleanroom Approach to Quality Software Development*, John Wiley & Sons, Ltd., New York, NY, 1992.
16. ACRS memorandum on SRP Chapter 7 (I&C) Update. From Dr. Don W. Miller, Chairman, Instrumentation and Control Systems and Computers Subcommittee, to all ACRS members, May 2, 1996.
17. Mills, Harlan D. "Zero Defect Software: Cleanroom Engineering," in *Advances in Computers, Volume 36* (ed. Marshall C. Yovits), Academic Press, Inc., Boston, MA, 1993.
18. Fisher, Alan S. *CASE: Using Software Development Tools*, John Wiley & Sons, Ltd., New York, NY, 1988.
19. Hecht, Herbert. "Rare Conditions - An Important Cause of Failures," *COMPASS '93 (Proceedings of the 8th Annual Conference on Computer Assurance*, Gaithersburg, MD, June 14-17, 1993), National Institute of Standards and Technology,
20. Leveson, Nancy G., Stephen S. Cha, and Timothy J. Shimeall. "Safety Verification of Ada Programs Using Software Fault Trees," *IEEE Software*, July 1991: 48-59.
21. Garrett, C.J., S.B. Guarro, and G.E. Apostolakis. "The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, 25(5): May 1995.
22. Levendel, Ytzhak. "Improving Quality with a Manufacturing Process," *IEEE Software*, March 1991: 13-25.
23. Laprie, J.-C. "For a Product-in-a Process Approach to Software Reliability Evaluation," *Proceedings. Third International Symposium on Software Reliability Engineering*, 134-139.
24. Humphrey, Watts S., Terry R. Snyder, and Ronald R. Willis. "Software Process Improvement at Hughes Aircraft," *IEEE Software*, July 1991: 11-23.
25. IEEE 610.12-1990: *IEEE Standard Glossary of Software Engineering Terminology*.
26. Rushby, John. "Formal Methods and their Role in the Certification of Critical Systems," Technical Report CSL-95-1, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1995.
27. Sheng, G., M.S. Elzas, T.I. Oren, and B.T. Cronhjort. "Model Validation: A Systemic and Systematic Approach," *Reliability Engineering and System Safety*, 42(1993): 247-259.
28. Leveson, Nancy G. and Clark S. Turner, "An Investigation of the Therac-25 Accidents," *IEEE Transactions on Software Engineering*, July 1993: 18-41.
29. Neufelder, Anne Marie. *Ensuring Software Reliability*, Marcel Dekker, Inc. (New York, NY), 1993.
30. Preckshot, G.G., and J.A. Scott. NUREG/CR-6421: "A Proposed Acceptance Process for Commercial Off-the-Shelf (COTS) Software in Reactor Applications," Lawrence Livermore National Laboratory, March 1996.
31. Musa, John D. and A. Frank Ackerman. "Quantifying Software Validation: When to Stop Testing?," *IEEE Software*, May 1989: 19-27.
32. Duke, Eugene L. "V&V of Flight and Mission-Critical Software," *IEEE Software*, May 1989: 39-45.



33. Wallace, Dolores R. and Roger U. Fujii. "Software Verification and Validation: An Overview," *IEEE Software*, May 1989: 10-17.
34. Wallace, Dolores R. and Roger Fujii. "Verification and Validation: Techniques to Assure Reliability," *IEEE Software*, May 1989: 9.
35. Atomic Energy of Canada Ltd. and Ontario Hydro. *CE-1001-STD Rev. 1: Standard for Software Engineering of Safety Critical Software*, January 1995.
36. Joannou, Paul K. "Experiences from Application of Digital Systems in Nuclear Power Plants."
37. McDougall, J., M. Viola, and G. Moum. "Tabular Representation of Mathematical Functions for the Specification and Verification of Safety Critical Software."
38. Keller, Marilyn and Ken Shumate. *Software Specification and Design*, John Wiley & Sons, Inc., New York, NY, 1992.
39. Parnas, D.L., G.J.K. Asmis, and J. Madey. "Assessment of Safety-Critical Software in Nuclear Power Plants," *Nuclear Safety*, 32(2): 189-198.

Other References (not cited with a reference number) -- in order of appearance:

- \* Standard Review Plan, Chapter 7 (Draft), "Instrumentation & Controls"
  - > Section 7.0, "Instrumentation and Controls -- Overview of Review Process" (Version 4.0), August 22, 1996.
  - > Appendix 7.0-A, "Review Process for Digital Instrumentation and Control Systems" (Version 7.0), August 23, 1996.
  - > Section 7.1, "Instrumentation and Controls -- Introduction" (Version 8.0), August 23, 1996.
  - > Appendix 7.1-A, "Acceptance Criteria and Guidelines for Instrumentation and Control Systems Important to Safety" (Version 8.0), August 23, 1996.
  - > Appendix 7.1-B, "Guidance for Evaluation of Conformance to ANSI/IEEE Std 279" (Version 8.0), August 22, 1996.
  - > Appendix 7.1-C, "Guidance for Evaluation of Conformance to IEEE Std 603" (Version 2.0), August 22, 1996.
- \* 10 CFR 50 (and 52) and corresponding appendices -- sections of the Code of Federal Regulations
- \* NUREG/CR-6101, "Software Reliability and Safety in Nuclear Reactor Protection Systems," November 1993.
- \* IEEE Std 603-1991, "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations."
- \* IEEE Std 279-1971, "Criteria for Protection Systems for Nuclear Power Generating Stations."
- \* Regulatory Guide 1.153, "Criteria for Power, Instrumentation, and Control Portions of Safety Systems," 1985.
- \* Regulatory Guide 1.152, "Criteria for Digital Computers in Safety Systems of Nuclear Power Plants," January 1996.

- \* IEEE Std 7-4.3.2--1993, "IEEE Standard for Digital Computers in Safety Systems of Nuclear Power Generating Stations."
- \* Regulatory Guide 1.1yy (Draft), "Verification, Validation, Reviews, and Audits for Digital Computer Software."
- \* IEEE Std 1074-1991, "IEEE Standard for Developing Software Life Cycle Processes."
- \* IEEE Std 1012-1986, "IEEE Standard for Software Verification and Validation Plans."
- \* ANSI/ANS-10.4-1987, "American Standard Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry."
- \* IEEE Std 1028-1988, "IEEE Standard for Software Reviews and Audits."
- \* IEEE Std 830-1993, "IEEE Recommended Practice for Software Requirements Specifications."
- \* Regulatory Guide 1.1ww, "Software Requirements Specifications for Digital Computer Software."
- \* CE-1001-STD Rev. 1, "Standard for Software Engineering of Safety Critical Software," January 1995.

---

## Appendix A    Comparison Between Analog and Digital Instrumentation and Control Systems

---

	Analog	Digital
Data collection	Continuous.	Either periodic or event-driven discrete data sampling.
Data transmission	Each measurement, function, or command is continuously transmitted over dedicated metal wire line.	Multiple measurements, functions, or commands are transmitted over shared metal wire or fiber optic lines configured as data busses or data highways, or over point-to-point data links with or without multiplexing.
Control logic implementation	Logic is implemented using fixed-function components.	Logic is implemented using programmable function components.
Operator interface	Fixed-function displays are hardwired between control room and components in the field.	Programmable interactive displays are connected to programmable control logic components. Data are synthesized for more compact presentation.
Reliability	Temperature changes, moisture, smoke, radiation, and aging cause calibration drift. Components exhibit "bathtub-shaped" failure curve: high initial number of failures due to manufacturing flaws, and high rate of failures at end of service life due to aging effects.	Temperature changes, moisture, smoke, and radiation may degrade performance. Digital I&C circuits do not drift so overall drift problems are greatly reduced. Hardware exhibits "bathtub-shaped" failure curve. There is no equivalent software failure rate curve, although initially failures tend to be higher and then decrease as the system is debugged. There are no generally accepted methods to quantify software reliability.
In-service testing (surveillance)	In-service testing is done manually by operating and maintenance personnel.	Some in-service self-testing of hardware is done automatically with preprogrammed software. Some testing is done automatically after manual initiation. Some testing is done manually.
Preoperational testing	Exhaustive output versus input functional testing and cycle-to-failure testing are commonly done.	Exhaustive functional testing and cycle-to-failure testing are done on simple combinational logic systems such as trip systems. Statistical functional testing is done on complex sequential logic systems such as process control and regulation systems. Latent software errors may be difficult to find.

---



---

## **Appendix B    Development of the NRC List of Digital Instrumentation and Control Issues**

---

At its first meeting, the Committee identified and considered a number of issues and facets of issues, as shown in this appendix. These initial deliberations, as well as those of later Committee meetings, were analyzed and organized, and they eventually led to the final list of six technical and two strategic issues. This appendix provides an insight into some of these deliberations by listing some of the earlier, more specific issues and topics, tied to each of the final list of eight.

### **SOFTWARE QUALITY ASSURANCE**

How can confidence be obtained in the safety and/or reliability of software? How should software be assessed?

What methods are appropriate and effective (e.g., verification and validation techniques, formal methods, quantification, hazard analysis, failure mode analysis and design)?

Do some software design techniques present special problems in assessment (e.g. artificial intelligence techniques)?

How can it be assured that changes and fixes do not degrade reliability and safety? What changes should require USNRC approval and which should not? What changes should be instituted for change control? (E.g., should patching be allowed?) How can it be assured that required changes are made?

Confidence level (quality, verification and validation, formal methods, lack of meaningful standards).

Certification basis (process vs. product).

Fear of unintended function(s).

Configuration control (maintenance/upgrading).

Security considerations.

### COMMON-MODE SOFTWARE FAILURE POTENTIAL

Are changes needed in the procedures for evaluating common-mode failures?

Reliability vs. safety. Do the enhanced capabilities of software allow new means of protection against computer failures or failure modes?

Quality vs. diversity. How much relative attention should be paid to each?

Diversity achievement.

Progressive approach to failure (defense-in-depth).

### SYSTEM ASPECTS OF DIGITAL I&C TECHNOLOGY

How can potential safety augmentation at the system level by the use of computers (e.g., diagnosis and accident management) be balanced and evaluated against potential safety decreases (e.g., owing to overreliance or to poor design/implementation that does not achieve assumed benefits or makes things worse)?

Performance during transients, anticipated transient without scram (ATWS) issues, fail-safe design. (E.g., can failures be detected as easily as with analog devices?) Does the use of computers make any difference in these areas?

Are there new environmental concerns (electromagnetic interference, climate control, etc.)?

What behaviors or features are of concern and how do we provide confidence (assessment) for them (e.g., unintended function, performance issues, capacity and overload, fail-safe design, networking)?

Communications system distractions.

System capacity.

Time response of the system.

Network reliability especially in advanced plants.

Recognition/detection of failure modes.

Architecture performance during transients.

Integration issues with analog.

## **HUMAN FACTORS AND HUMAN-MACHINE INTERFACES**

Should restrictions be imposed on the safety or safety-related functions that can be allocated to computers vs. operators or analog devices?

Other operator aids such as alarm analysis, value sequencing, and decision analysis.

Task allocation (computer vs. human).

Level of automation.

Human interface (role, display, information, nuances).

Use of "intelligence" aids (neural nets, artificial intelligence, etc.).

Operations and maintenance impacts (pluses and minuses).

## **SAFETY AND RELIABILITY ASSESSMENT METHODS**

Are there any implications for design basis accidents and the procedures for certifying against them?

What are the implications of using computers with respect to probabilistic risk assessment (PRA) procedures and use?

Are we taking solutions for old technology and inappropriately applying them to new technology (e.g., emphasis on diversity and redundancy, bottom-up component reliability approaches vs. risk-based or hazard analysis approaches)? Are there new approaches that may be more appropriate?

Assessment technology.

Added complexity of digital technology vs. analog.

Definition of safety margin with digital technology.

Loss of margin of safety by consolidation of data.

PRA or mathematical assessment method validity with digital technology.

---

## DEDICATION OF COMMERCIAL OFF-THE-SHELF HARDWARE AND SOFTWARE

Are special procedures required for software tools (e.g., compilers, code generators)?

What assessment procedures are appropriate for COTS software? How should dedication procedures differ from those used to certify (handle) specially constructed software?

IEEE-STD-279 compliance.

Use of standard software tools/compilers.

## CASE-BY-CASE LICENSING PROCESS

Types of software complexity: Should the assessment basis and procedures differ?

Are there fundamental differences in functionality between analog and digital devices, e.g., between their failure modes, and do they affect certification or licensing?

Use of computers in safety vs. nonsafety systems.

Does the use of computers change the basis for certification procedures at the system level?

Should restrictions be imposed on the safety or safety-related functions that can be allocated to computers vs. operators or analog devices?

What should be the limits of the USNRC regulatory activities?

How does the USNRC determine whether safety value has been added or reduced?

Should the certification basis for computers and software be different from that for the analog devices they replace?

Does certification of product imply imposition of process?

How can the USNRC determine whether safety or reliability has been degraded when we retrofit computers into existing designs?

How should version control be managed? Is this a USNRC concern?

Safety/control systems separation vs. analog.

Lack of understanding of design basis.

Safety vs. nonsafety systems.

Digital value added (e.g., accident diagnosis and management).

Regulatory constraints.

Short half-life of the technology.



### **ADEQUACY OF TECHNICAL INFRASTRUCTURE**

How should the USNRC deal with the rapid changes in technology?

Lack of strategic plan for the USNRC research program.

Other industry experience as part of the USNRC technical basis.



---

## Appendix C Sample “Completeness Criteria” for Requirements

---

(From Leveson’s *Safeware*<sup>10</sup>)

1. A single occurrence of a given stimulus or trigger must produce a *single* output, and the output must not be produced without the trigger.
2. The system, software, interlocks, variables, and clocks must start in a safe state and must be initialized and updated to the proper status at startup and after shutdowns or other such interruptions of normal operation.
3. Software behavior must be specified for similar off-normal situations (e.g., inputs received before startup or after shutdown).
4. Input responses and maximum times for input arrival must be specified for all states (including indeterminate ones).
5. Paths from fail-safe states must be specified, and time in reduced-function states must be minimized.
6. All sensor information should be used; legal values never reached may indicate specification incompleteness.
7. Every state must have behaviors defined for every possible input and must have a finite number of output behaviors, and the behavior at any particular time should be deterministic.
8. Response to excessive inputs (load violations) must be specified, and if it involves performance degradation, the degradation must be smooth and predictable.
9. Safety-critical outputs must be checked for reasonableness and hazardous values or timing.
10. Inputs which specify outputs must only be used for a specified time before they must be updated.
11. All specified states must be reachable from the initial state, and states should not prohibit production of later required outputs.
12. There should be multiple ways to reach safe states, and multiple preventative measures to impede reaching hazardous states.
13. All hazardous states must have one or more paths to safe (or at least minimum-risk) states.
14. There are also a variety of more specific suggestions on such topics as data, timing, the human-computer interface, and other related issues.



---

## Appendix D Lutz's "Safety Checklist"

---

(From Lutz.<sup>11</sup> The first ten items refer to interface issues, the last six to robustness.)

1. Is the software's response to *out-of-range values* specified for every input?
2. Is the software's response to *not receiving an expected input* specified? (That is, are time-outs provided?) Does the software specify the length of the time-out, *when to start counting* the time-out, and the latency of the time-out (the point past which the receipt of new inputs cannot change the output result, even if they arrive before the actual output)?
3. If *input arrives when it shouldn't*, is a response specified?
4. On a given input, will the software always follow the same path through the code (that is, is the software's behavior *deterministic*)?
5. Is each input *bounded in time*? That is, does the specification include the earliest time at which the input will be accepted and the latest time at which the data will be considered valid (to avoid making control decisions based on obsolete data)?
6. Is a minimum and maximum *arrival rate* specified for each input (for example, a capacity limit on interrupts signaling an input)? For each communication path? Are checks performed in the software to avoid signal saturation?
7. If interrupts are masked or disabled, can *events be lost*?
8. Can any output be produced faster than it can be used (absorbed) by the interfacing module? Is *overload* behavior specified?
9. Is all data output to the buses from the sensors *used* by the software? If not, it is likely that some required function has been omitted from the specification.
10. Can input that is received *before startup, while off-line, or after shutdown* influence the software's startup behavior? For example, are the values of any counters, timers, or signals retained in software or hardware during shutdown? If so, is the earliest or most-recent value retained?
11. In cases where performance degradation is the chosen error response, is the degradation *predictable* (for example, lower accuracy, longer response time)?
12. Are there *sufficient delays* incorporated into the error-recovery responses, e.g., to avoid returning to the normal state too quickly?
13. Are *feedback loops* (including echoes) specified, where appropriate, to compare the actual effects of outputs on the system with the predicted effects?
14. Are all modes and modules of the specified software *reachable* (used in some path through the code)? If not, the specification may include superfluous items.

15. If a hazards analysis has been done, does every path from a hazardous state (a failure-mode) *lead to a low-risk state*?
16. Are the inputs identified which, *if not received* (for example, due to sensor failure), can lead to a hazardous state or can prevent recovery (single-point failures)?

Two questions have been suggested for addition to the checklist. One of them deals with data consistency (“Are checks for consistent data performed before control decisions are made based on that data?”); the other deals with generic structures (“Are generic structures used whenever appropriate to restrict the number of possible hazardous modes and states?”).

---

## Appendix E Sample Guidance from BTP-14

---

This appendix provides summaries of some of the guidance given in BTP-14. There are examples from each of the analysis phases (planning, implementation, and design outputs).

### Planning phase:

- \* *Software Quality Assurance Plan* -- An appropriate outline for this document can be found in IEEE Std 730.1, "Standard for Software Quality Assurance Plans." The document should conform to the requirements of 10 CFR 50, Appendix B and ASME Std NQA-1-Part II.2.7. The SQAP should contain descriptions of the following:
  - > The software quality assurance (QA) management method (including QA tasks and responsibilities)
  - > Documents subject to software QA oversight
  - > The problem reporting, tracking, and resolving process
  - > Any special software tools and methods that will be used to support the software QA effort
  - > The provisions used to ensure that software provided by suppliers will meet established project requirements
  - > The methods used for software QA records collection, maintenance, and retention.
  - > The methods and procedures used to identify, assess, monitor, and control areas of risk (especially if safety-related)
  - > All required review plans, specifications, and procedures, including review documentation requirements, evaluation criteria, error reporting, and anomaly resolution procedures (see Reg. Guide 1.1yy and IEEE Std 1028 for guidance on the review and audit process)
- \* *Software Safety Plan* -- An appropriate format for this plan is found in IEEE Std 1228, "Standard for Software Safety Plans." It should contain the following:
  - > A description of the software safety organization and a designated safety officer with responsibility for the required software safety qualities and authority to enforce the safety requirements in the specification, design, and implementation of the software
  - > A description of software safety activity management within the development organization, including responsibilities, resource requirements, staff qualifications and training, life cycle software safety actions, documentation

requirements, software safety program records, safety requirements on software quality assurance and configuration management, software safety tool support, and actions for previously developed or purchased software

- > A requirement that a safety analysis be performed and documented on each major design document, requirement, design description, and source code, and that hazards (including ACEs) and hazard reduction efforts be analyzed and documented
  - > A requirement that the safety organization be authorized to reject the re-use of existing software or tools if the tools (or use of them) cannot be shown to be adequately safe
  - > A description of safety-related tests not included in the software V&V plan
- \* *Software Verification and Validation Plan* -- An appropriate format for this plan is found in Reg. Guide 1.1yy and ANSI/IEEE Std 1012. It should contain descriptions of the following:
- > The software V&V organization, including staff capabilities, reporting channels, organizational interfaces, and training requirements
  - > The responsibilities for executing each V&V task, approval authority for them, and personnel assignments to cover them
  - > The degree of independence between the development and V&V organizations
  - > The management of the V&V effort, including reporting procedures, management reviews and audits, methods of carrying out the different V&V activities, completion criteria for the V&V activities, and methods for resolving discrepancies and anomalies
  - > The V&V activity schedule and required resources, with justifications to show that the schedule of V&V activities is sufficient to ensure the software system safety
  - > All required testing plans, specifications, procedures, and cases (including unit, integration (subsystem), system, and acceptance testing), as well as test documentation requirements, evaluation criteria, error reporting, and anomaly resolution procedures (guidance on test documentation is found in Reg. Guide 1.1vv and IEEE Std 829, and guidance on software unit testing is found in Reg. Guide 1.1xx and ANSI/IEEE Std 1008)
  - > V&V reporting requirements, including personnel involved, procedures, and results, evaluation criteria, error reporting, and anomaly procedures
  - > All tools and methods to be used in the V&V tasks

Implementation phase:

- \* V&V activities should be summarized for each activity group and should address each requirement, design element, code element, review, and test.



- 
- \* V&V documentation should confirm that each of the design elements above satisfies all of the functional and software development process characteristics (described earlier in this thesis).
  - \* Problems identified by the verification effort should be documented along with actions to take and actions taken in response to them.
  - \* A traceability matrix should be created that:
    - > Shows the traceability between the system level requirements and one or more SRS requirements
    - > Can be extended to design, implementation, and validation and updated after each life cycle activity group
  - \* Integration V&V activities should have, in part, the following attributes:
    - > All required unit and subsystem tests should be completed successfully and anomalies or errors found should be resolved and documented.
    - > Final integration tests should be completed and documented.
  - \* Software validation activities should have the following attributes:
    - > There should be tests (with defined test setup, input data requirements, output data expectations, completion time, and acceptance criteria) for each SRS requirement.
    - > The result of each test should *clearly show* that the associated requirement has been met [emphasis added].
    - > Correction and re-test procedures should be included to handle errors and anomalies.
    - > A final report should be made summarizing problems, errors, and corrective actions. “The report should contain a statement that the validation testing was successful and that the software tested met all of the requirements of the SRS.”
  - \* Installation activities should have, in part, the following attributes:
    - > The test configuration, required inputs, expected outputs, test execution steps, acceptance criteria, problem identification, and required problem mitigation or elimination actions should be documented.
    - > Installation problems and their resolution should be documented.
    - > An acceptance test report should be prepared and “should contain a statement that the plan was successfully executed, and the system is ready for operation.” It should also demonstrate that the system “operates correctly...”

Design Outputs:

- \* *Software Requirements Specification* -- A proper format for this can be found in Reg. Guide 1.1ww and IEEE Std 830. Some of the qualities the SRS should have and what they entail are:
  - > *Functionality* -- “.....Functions should be specified in terms of inputs to the function, transformations to be carried out by the function, and outputs generated by the function.”
  - > *Safety* -- “.....the software functions, operating procedures, input, and output [must] be classified according to their importance to safety. Requirements important to safety should be identified as such in the SRS. The identification.....should include safety analysis report requirements, as well as abnormal conditions and events as described in Reg. Guide 1.152.”
  - > *Completeness* -- “.....all actions required of the computer system be fully described for all operating modes and all possible values of input variables (including anomalous values).....should describe any actions that the software is prohibited from executing. The operational environment.....should be described. All variables..... the software must monitor and control shall be fully specified. Functional requirements should describe (1) how each function is initiated; (2) the input and output variables required of the function; (3) the task sequences, actions, and events required to carry out the function; and (4) the termination conditions and system status at the conclusion of the function. User interfaces should be fully specified for each category of user.”
  - > *Style* -- “.....requires that the contents of the SRS be *understandable*.”
- \* *Code Listings* -- These should exhibit, in part, the following characteristics:
  - > *Robustness* -- “.....corrupted data will not cause the safety system to fail. Data corruption should be avoided.”
  - > *Consistency* -- (of variable names, types, locations, array sizes, etc.) -- “The code should use mathematical models, algorithms, and numerical techniques described in or derived from the SDS [Software Design Description].”
- *System Build Documents* -- Among other characteristics, they should exhibit *completeness*, which “requires that all build procedures be *fully specified*” and that the “documents should include all required software units, including code and data, that are part of the build.”

---

## Appendix F Sample Guidance from SRP Section 7.0

---

Section 7.0 of the Standard Review Plan (SRP), an overview of the review process for digital I&C systems, briefly discusses guidelines for evaluating design certifications, construction permit applications, operating or combined license applications, and license amendments or topical reports. For *design certifications and construction permit applications*, these guidelines include evaluations of the following review points:

- \* For the *system concept*:
  - > “The overall I&C system design’s relationship to both the functions required [by] 10 CFR 50 and the functions required to support the assumptions of the plant accident analysis. (See Section 7.1)”
  - > “The adequacy of any research and development plan necessary to resolve any outstanding questions concerning the design of systems or components.”
  - > “Compliance with the technically relevant portions of 10 CFR 50. (See Section 7.1)”
  - > “Proposed resolution of technically relevant unresolved safety issues [USIs] and medium- and high-priority generic safety issues [GSIs] identified more than six months prior to the application. (See SRP Chapter 20).”
- \* For *system requirements*:
  - > “Principal design criteria with respect to the guidance of 10 CFR 50.55a(h) (ANSI/IEEE Std 279-1971) and 10 CFR 50, Appendix A. (See Section 7.1.)
  - > “The design bases and the relationship of the design bases to the principle design criteria. (See Sections 7.2 through 7.9.)”
  - > (There is additional guidance for applications under 10 CFR 52)
- \* For *system design*:
  - > “The key characteristics, performance requirements, general arrangements, and materials of construction of the systems to confirm that there is reasonable assurance the final design will conform to the design bases with adequate margin for safety. (See Sections 7.2 through 7.9)”
  - > “The identification of instrumentation and control functions and variables to be probable subjects of technical specifications for the facility. (See Sections 7.2 through 7.9)”
  - > “Proposed technical specifications.”

- > “The applicant/licensee’s analysis and technical justification to show that the instrumentation and control system design, including the underlying design bases and performance requirements, can perform appropriate safety functions.”

For *operating and combined license (CL) applications* (which generally have already passed the review criteria above), the guidelines cover the following topics:

- \* For *hardware and software requirements, detailed design, fabrication, test, and integration evaluation*:
  - > “Implementation of development plans. (See Appendix 7.0-A.)”
  - > “Conformance of design outputs with system requirements. (See Sections 7.2-7.9 and Appendix 7.0-A.)”
  - > “Evidence of design process characteristics in design outputs. (See Appendix 7.0-A.)”
  - > “The description and evaluation of the results of the applicant/licensee’s research and development to demonstrate that any safety questions identified at the [construction permit] stage have been resolved. (See Section 7.2 through 7.9 and Appendix 7.0-A.)”
- \* For the *system validation evaluation*
  - > “The applicant/licensee’s testing, analysis, and technical justification to show that I&C system design, including the underlying design bases and performance requirements, can perform appropriate safety functions. (See Section 7.2 through 7.9 and Appendix 7.0-A.)”
  - > (For combined licenses only) “The applicant/licensee’s demonstration of compliance with the interface requirements, for applications referencing a certified standard design.”
  - > (For combined licenses only) “ITAAC [Inspections, Tests, Analyses, and Acceptance Criteria] proposed to provide reasonable assurance that, if the inspections, tests, and analyses are performed, the acceptance criteria met, and a plant is built according to the design, then the plant will operate in accordance with the design certification. (Applications that reference a certified standard design must apply the certified design ITAAC to those portions of the facility covered under the DC.) (See SRP Chapter 14.)”
- \* For the *installation, operations, and maintenance evaluation*
  - > “Site visit. (See Appendix 7-B.)”
  - > (For combined licenses only) “Implementation of ITAAC. (See SRP Chapter 14.)”

The guidelines for license amendments and topical reports vary based on their particular content, but any or all of the above points may be applicable.



---

## Appendix G Digital Issues Discussed in SRP Section 7.1

---

The following is a discussion of what guidance is given in relation to the seven *digital* safety issues considered to be of particular concern by SRP Section 7.1

1. In relation to *electromagnetic capability*, the reviewer (or designer) is referred to EPRI TR [Topical Report] -102323, "Guidelines for Electromagnetic Interference Testing in Power Plants" and to the guidelines concerning lightning in NFPA Std 78 and ANSI/IEEE Std 665.
2. Requirements for *computer system quality* and reliability come from IEEE 279, GDC 1, GDC 21, GDC 29, and 10 CFR 50 Appendix B ("Quality Assurance Criteria for Nuclear Power Plants and Fuel Reprocessing Plants"). This area is broken into five categories:
  - a) *Software development and hardware/software integration* involves performing system integration with a "well-structured and well-executed software engineering process" -- the user is referred to the related requirements of 10 CFR Appendix B, ASME Std NQA-2a Part 2.7, Reg. Guide 1.1zz (on digital safety system software life cycles), and the software characteristics of BTP-14 (at which point we seem to have come full circle, as we started with BTP-14, which led us to this document, which leads us back to BTP-14!).
  - b) The section on *qualification of existing commercial computers (including pre-existing software products)* refers the user to guidance on an acceptable way of doing such a qualification (possibly with the aid of engineering judgment) in IEEE Std 7-4.3.2 Section 5.3.2, EPRI TR-106439, NUREG/CR-6421, and BTP-18.
  - c) *Software tools* requirements are referenced to IEEE Std 7-4.3.2 Section 5.3.3, the EPRI TR-106439 qualification process, and the BTP-14 development process.
  - d) *Verification and validation* requirements are referenced to IEEE Std 7-4.3.2 Section 5.3.4, the software engineering process described in BTP-14, and Reg. Guides 1.1yy, 1.1vv, and 1.1xx (on V&V, test documentation, and unit testing, respectively).
  - e) *Software configuration management* requirements are referenced to IEEE Std 7-4.3.2 Section 5.3.5, ASME NQA-2a Part 2.7, BTP-14, and Reg. Guide 1.1uu.
3. *Equipment qualification* involves the I&C system being able to withstand both normal and adverse environmental conditions. Guidance to fulfill the requirements on this topic (from 10 CFR Appendix A, GDC 4, 10 CFR 50.49, and IEEE Std 279 Sections 3.7, 4.4, and 4.5) can be found in Reg. Guide 1.89, which endorses IEEE Std 323.

4. *System integrity* requirements for functional capabilities under various extremes (of the environment, power supply, accidents, etc.) come from IEEE Std 279 Section 4.5, GDC 21, and IEEE Std 603 Section 5.5. This topic includes two subtopics:
  - a) *Design for computer integrity* requirements from IEEE Std 7-4.3.2 Section 5.5.1 (with guidance in BTP-21) involve using designs and architectures that support predictable real-time performance within design requirements.
  - b) *Design for test and calibration* requirements that support failure detection (when fail-safe designs are not possible) come from IEEE Std 603 Section 5.7 with guidance in BTP-17.
5. *Communications independence* requirements for independence among redundant protection system channels and among safety and non-safety systems come from IEEE Std 279 Sections 4.6 and 4.7, IEEE Std 603 Section 5.6, GDC 21, GDC 22, and GDC 24. IEEE 7-4.3.2 (especially Annex G) offers guidance on creating such designs.
6. *Reliability* requirements can address software and hardware either alone, together but as separate entities, or combined as a system. Basic reliability criteria are found in GDC 21, IEEE Std 279, and IEEE Std 603. Also, IEEE Std 7-4.3.2 Section 5.15 mentions specifically that proof of meeting system-level reliability requirements must account for software reliability. Because software errors do not follow random failure behavior like hardware errors, *quantitative reliability goals and data are not supported as a sole criterion of quality*, only as supplementary information to increase confidence (from Reg. Guide 1.152, which endorses IEEE Std 7-4.3.2).
7. Requirements for *defense against common-mode failures* come from the Staff Requirements Memorandum on SECY-93-087, with guidance from BTP-19.



---

## Appendix H Sample Guidance from SRP Appendix 7.1-A

---

The following are examples of the guidance given in Appendix 7.1-A. Brief descriptions of the criteria are followed by applicability statements (how the criteria apply to I&C reviews) and review methods (how to ensure the design conforms to the criteria).

### *Requirements from 10 CFR 50 & 52*

1. *50.55a(a)(1) Quality Standards for Systems Important to Safety* -- “Structures, systems, and components must be designed, fabricated, erected, constructed, tested, and inspected to quality standards commensurate with the importance of the safety function to be performed.” -- This applies to all I&C systems. The review methods simply say “the licensee should commit to conformance with the regulatory guides and standards referenced in Sections 7.1 through 7.9 and Chapter 7 Appendix A.”
2. *52.47(a)(2) Level of Detail* -- “The application must contain a level of detail sufficient to.....judge the.....licensee’s proposed means of assuring that construction conforms to the design and to reach a final conclusion on all safety questions.....[and] must include performance requirements and design information sufficiently detailed to permit the preparation of acceptance and inspection requirements by the NRC.....” -- This applies to “all I&C systems that are part of applications for design certification under 10 CFR 52, Subpart B, or combined licenses under 10 CFR 52, Subpart C.” The review methods information says that “sufficient information for an NRC safety determination should be provided for each I&C system,” and that BTP-16 can aid in determining what is sufficient for application under 10 CFR 52, Subpart B.

### *General Design Criteria (from 10 CFR 50 Appendix A)*

1. *Criterion 1 -- Quality Standards and Records* -- “Structures, systems, and components important to safety shall be designed, fabricated, erected, and tested to quality standards commensurate with the importance of the safety functions to be performed.....A quality assurance program shall be established and implemented in order to provide adequate assurance that these structures, systems, and components will satisfactorily perform their safety functions.....” -- This applies to “all I&C systems and components important to safety.” The review methods information says that the applicable regulatory guides and endorsed codes and standards (all of which are identified in Section 3 of this appendix) should be selected as part of the process of SRP Section 7.1.
2. *Criterion 22 -- Protection System Independence* -- “The protection system shall be designed to assure that the effects of natural phenomena and of normal operating, maintenance, testing, and postulated accident conditions on redundant channels do not result in loss of the protection function.....Design techniques, such as functional diversity or diversity in.....design and.....operation, shall be used to the extent practical.....” -- This applies to protection systems (RTS, ESFAS, and supporting

data communication systems). The review methods information refers the user to a wide array of other documents (depending on the particular topic): Reg. Guide 1.75, IEEE Standards 384 and 7-4.3.2, BTPs 11 and 19, SRP Sections 7.2, 7.3, and 7.9, Appendix 7.1-B items 3 and 7, and Appendix 7.1-C items 6, 11, and 24.

#### *Regulatory Guides and Branch Technical Positions*

1. *Regulatory Guide 1.152 (endorses IEEE 7-4.3.2)* -- This applies to all I&C safety systems and supporting data communication systems. The review methods information says only that this Reg. Guide allows for evaluation of conformance with GDC 21 and that “additional guidance” to “supplement” it is provided by BTPs 14, 17, 18, 19, and 21.
2. *Regulatory Guide 1.1uu-1.1zz* -- All of these apply to all I&C systems and components important to safety. The review methods information is similar for all of them. As an example, for Reg. Guide 1.1yy (on V&V) the user is referred to 10 CFR 50.55a(a)(1), 50.55a(h), GDC 1, and Criteria I, II, III, XI, and XVIII of 10 CFR Appendix B. The reader is also referred to ANSI/IEEE Std 1012 (“IEEE Standard for Software Verification and Validation Plans”) for guidance on planning the V&V of safety system software and to IEEE Std 1028 (“IEEE Standard for Software Reviews and Audits”) for acceptable approaches for carrying out software reviews, inspections, walkthroughs, and audits. In addition, the user is referred back to BTP-14.
3. *Branch Technical Positions* -- Their applicability is as noted in them individually. The only review methods information is that “the BTPs provide bases for evaluating specific review areas.”

---

## Appendix I Guidance Topics from SRP Appendix 7.1-B

---

SRP Appendix 7.1-B summarizes guidance from IEEE 279 Sections 3 and 4. IEEE Section 3 addresses such topics as:

- \* Completeness in addressing GDC 20 and describing the functional requirements and operational environment for the I&C system; also consistency, correctness, traceability, unambiguity, and verifiability (the definitions for these terms are of the same type as those in BTP-14)
- \* Identification of all conditions (and corresponding monitored variables) that require protective action (and identifying where to measure those variables)
- \* Identification of operational limits, margins between them, setpoints (where unsafe conditions begin), and limits requiring protective action
- \* Identification of transient and steady-state conditions for the energy supply and the environment during normal, abnormal, and accident conditions, as well as identification of and provisions for the initiators that could damage the protective system or change the environment
- \* Identification of (digital I&C) system performance requirements (response times, accuracy, variable monitoring rates, etc.) to allow for completion of protective actions

IEEE 279 Section 4 addresses more specific topics, such as:

- \* *General functional requirements* -- “The.....analysis should show that the protection system has been qualified to demonstrate that the performance requirements are met.” This includes proper software/hardware task allocation, “deterministic and known” real-time performance, and automatic *and* manual initiation capability for protective functions. “The evaluation of *precision* is addressed to the extent that setpoint, margins, errors, and response times are factored into the analysis.” “.....*acceptance of system reliability is based on deterministic criteria for both the hardware and software rather than on quantitative reliability goals. .... the methods to be used to confirm that these deterministic criteria have been met [should be discussed].*” The use of quantitative measures to increase confidence in reliability but *not* as a sole means of reliability determination is discussed. “The applicant.....should justify that the degree of *redundancy, diversity, testability, and quality.....*is adequate to achieve functional reliability commensurate with the safety functions to be performed.” The user is referenced *back* to BTP-14 and Reg. Guide 1.152 for guidance on designing for and determining software reliability. “The assessment of reliability should evaluate the effect of possible hardware and software failures and the design features provided to prevent or limit the effects of these failures.” Some examples of failure types to look for are given with a reference to NUREG/CR-6101 for more information. [Emphasis added.]

- \* *Single-failure criterion* -- No single protection system failure should prevent necessary protective actions under any circumstances (i.e., defense-in-depth).
- \* *Quality of components and modules* -- The appropriate quality standards should be followed for the protection system (10 CFR 50 Appendix B), including original and pre-existing or COTS software (BTP-14; EPRI TR-106439 and NUREG/CR-6421).
- \* *Capability for test and calibration* -- Testing should simulate as realistically as possible the required performance of the protection system and should test both automatic and manual circuitry.
- \* *Multiple setpoints* -- The more restrictive setpoint should be used automatically when required.
- \* *Completion of a protective action once it is initiated* -- Features should be provided to ensure this happens.
- \* *Manual initiation* -- Design for this capability should be coordinated with the guidance of Reg. Guide 1.62 and of the Human Factors Assessment Branch of the review.
- \* *Access to setpoint adjustments, calibrations, and test points* -- This should be properly restricted (for either physical or electronic access).
- \* *System repair* -- Guidance on self-diagnostics is found in BTP-17.
- \* *Identification* -- In regard to software, this involves proper configuration management according to Reg. Guide 1.1uu and IEEE Std 828.

---

## Appendix J Digital Issues Discussed in SRP Appendix 7.0-A

---

This appendix summarizes the guidance provided by SRP Appendix 7.0-A on seven areas of clarification for *digital* I&C reviews. These areas are as follows:

1. *Adequacy of design criteria* (as evidenced by a commitment to Regulatory (Reg.) Guide 1.152, “Criteria for Digital Computers in Safety Systems of Nuclear Power Plants,” which endorses IEEE 7-4.3.2, “IEEE Standard for Digital Computers in Safety Systems of Nuclear Power Generating Stations,” and a set of software engineering standards such as Reg. Guides 1.1uu-1.1zz (on configuration management, test documentation, requirements specification, unit testing, V&V/reviews/ audits, and life cycle processes, respectively) sufficient to describe the software development process).
2. *Identification of review topics* based on the safety significance of the system being considered.
3. *Defense-in-depth & diversity (D-in-D&D)* in compliance with BTP-19 and the Staff Requirements Memorandum to SECY-93-087.
4. *Life cycle process planning* as addressed by BTP-14, Section B.2.1 (e.g. check that plant I&C system requirements are correctly decomposed to the digital system level; that the development process is specified and documented, yielding a high degree of confidence that functional requirements are implemented; that the development process and products can be inspected; and that the installed system functions as designed based on validation and integration tests, acceptance tests, and on-site pre-operational and start-up functional tests).
5. *Adequacy of system functional requirements* (requirements from IEEE 603, “IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations, and General Design Criteria [GDC] from 10 CFR 50 are complicated by the addition of digital computers to a system, because digital systems behave differently than analog systems in the areas of equipment qualification, real-time performance, on-line and periodic testing, and communications independence).
6. *Adequacy of life cycle process implementation* per BTP-14, Section B.2.2.
7. *Adequacy of design outputs* per BTP-14, Section B.2.3 (as well as Appendix 7-B and BTPs 17 and 21). This is accomplished by inspection of a “representative sample of the design outputs” [emphasis added] as well as V&V analyses and test reports to confirm proper implementation of functional characteristics. An accompanying confirmation of the development process characteristics builds confidence that the results of the functional inspections are indicative of all parts of the development process and products. Conclusions of these inspections are further strengthened by positive reviews of the development plans and process audits.



---

## Appendix K BTP-14 Software Characteristic Definitions

---

This appendix contains a summary of the definitions provided by BTP-14 for the two different types of software characteristics (functional and development process).

### Functional characteristics:

1. *Accuracy* -- degree of freedom from error in input, calculations, and output
2. *Functionality* -- the necessary software operations
3. *Reliability* -- degree of operation without failure
4. *Robustness* -- ability to function with incorrect inputs and/or a stressful environment
5. *Safety* -- degree of operation without *catastrophic* failure
6. *Security* -- ability to prevent unauthorized intrusions
7. *Timing* -- ability of software to meet timing goals within hardware constraints

### Development process characteristics:

1. *Completeness* -- full implementation of required software functions
2. *Consistency* -- no contradictions (within or between software components or documents)
3. *Correctness* -- no faults in specifications, design, or implementation
4. *Style* -- design output form/structure (i.e., understandability)
5. *Traceability* -- degree to which you can see specifications reflected in design outputs and vice versa
6. *Unambiguity* -- only one interpretation
7. *Verifiability* -- ability to facilitate criteria and tests to see if the criteria have been met





## Appendix L Safety Impact of Software Qualities from a Regulator Viewpoint

(From NUREG-CR/6421, p. 47)

	Impact on Operational Safety		
	Primary Impact	Secondary Impact	Little Impact
External (Functional) Qualities	Accuracy Acceptability Availability Completeness Correctness Interface Consistency Performance (Efficiency, Timing) Preciseness Reliability Robustness Security Usability	User Friendliness	
Internal (Engineering) Qualities	Integrity Internal Consistency Testability Validity	Clarity Interoperability Simplicity Understandability	Accountability Adaptability Generality Inexpensiveness Manageability Modularity Self-Descriptiveness Structuredness Uniformity
Future Qualities			Accessibility Augmentability Convertibility Extendibility Maintainability Modifiability Portability Reparability Reusability Serviceability

Note that qualities associated with modifications that might be made in the operations phase have been listed in the "Little Impact" category because an assumption is made here that, in typical safety-related reactor applications, changes will be infrequent. To the extent that such software might be used in an environment with regularly changing requirements, these qualities assume more importance. It should also be noted that, in some cases, listed qualities have essentially the same meaning but may have slightly different interpretations depending on the context. Since they all appear in the literature, no attempt has been made to group them. They are, however, categorized consistently.



## Appendix M    Testing Strategies Appropriate to Software Qualities

(From NUREG-CR/6421, p. 48)

Software Quality	Static Analysis	Structural Testing	Functional Testing	Statistical Testing	Stress Testing
Acceptability			X		O
Accuracy	O	X	X		
Availability				X	X
Clarity	X				
Completeness	X		X		O
Correctness	X	X	X		X
Integrity	O	X	X		
Interface Consistency	X		X		
Internal Consistency	X	X	O		
Interoperability	X		X		
Performance (efficiency & timing)		O	X		X
Preciseness	O	X	X		
Reliability				X	
Robustness	O		X		X
Security	O	X	X		
Simplicity	X				
Testability	X				
Understandability	X				
Usability	X		X		
User Friendliness			X		
Validity	X		X		
Regression Testing		O	X	O	X

X = Strategy should be used for the specified quality

O = Strategy may be used for the specified quality



## Appendix N Sample Prerequisites for and Extent of Testing

(From NUREG-CR/6421, pp. 53-54)

Strategy: Technique	Goal	Minimum Information Required	Suggested Extent of Testing/Analysis
<b>Static:</b>			
Inspection (I0)	Examine architectural design with requirements as reference	Software requirements; architectural design	One or more inspections. Group decision on re-inspection based on inspection results.
Inspection (I1)	Examine detailed design with architectural design as reference	Architectural & detailed design	One or more inspections. Group decision on re-inspection based on inspection results.
Inspection (I2)	Examine source code with detailed design as reference	Source code & detailed design	One or more inspections. Group decision on re-inspection based on inspection results.
Inspection (other)	Check code for specific qualities, properties, or standards adherence (can be part of I2)	Source code	One or more inspections. Group decision on re-inspection based on inspection results.
Inspection (other)	Verify allocation of software requirements	System requirements & software requirements	One or more inspections. Group decision on re-inspection based on inspection results.
Inspection (other)	Check application-specific safety requirements	System & software safety requirements; hazard/risk analyses	One or more inspections. Group decision on re-inspection based on inspection results.
Desk checking	Verify key algorithms & constructs	Source code	One pass per revision; continue until no new faults are found.
Automated structural analysis	Produce general/descriptive information; compute metrics values	Source code	One pass per revision
Automated structural analysis	Fault detection	Source code	One pass per revision; continue until no new faults are found.
Automated structural analysis	Standards violations	Source code	One pass per revision; continue until no new faults are found.

Strategy: Technique	Goal	Minimum Information Required	Suggested Extent of Testing/Analysis
<b>Structural:</b>			
Path	Verify internal control flow	Source code; module design specification	Branch coverage
Loop	Verify internal loop controls	Source code; module design specification	Focus on loop boundaries
Data flow	Verify data usage	Source code; module design specification	All-'definition-usage'-pairs
Domain (structural)	Verify internal controls/computations over input domains	Source code; module design specification	Focus on boundaries
Logic (structural)	Verify internal logic (implementation mechanisms)	Source code; module design specification	All combinations of conditions
<b>Functional:</b>			
Transaction	Verify implementation of application functions	Executable, software requirements	All transactions
Domain	Verify functional controls/computations over input domains	Executable, software requirements	Representative domain values including boundary and illegal values
Syntax	Verify user interface and message/signal constructs	Executable, software requirements	All input/message constructs
Logic	Verify implementation of the logic of the real-world application	Executable, software requirements	All combinations of real-world conditions
State	Verify implementation of states associated with the real-world application	Executable, software requirements	All states/transitions
Statistical	Estimate reliability	Executable, software requirements, operational profiles	Predetermined reliability target
Stress	Examine robustness; characterize degradation with increasing loads on resources	Executable, software requirements	One pass per resource per revision per operating mode; sampling of combinations of resource loads
Stress	Find breaking points; check recovery mechanisms	Executable, software requirements	Continue testing a resource until failure & recovery modes are well understood
Regression	Verify that changes have not impacted the software in unexpected ways	Various input needed depending on test strategies used in the regression test suite	Continue until no new failures are detected

## Appendix O Typical Testing Strategies for Investigating Software Qualities

(From NUREG-CR/6421, pp. 55-57)

Software Quality	Also see:	Question to be Answered	Applicable Testing Strategies
Acceptability	Validity	Are real-world events handled properly? How does the product perform in realistic, heavy load situations?	Functional (T,D,L,Se) Stress
Accuracy	Preciseness	Are internal calculations accurate? Are results accurate? Is there confidence that important calculations are accurate?	Structural (DF) Functional (T) Static analysis (L,DC)
Availability	Reliability	Will the software be unavailable due to poor reliability? Will functions be available during heavy load situations?	Statistical Stress
Clarity	Understandability	Is the implementation sufficiently clear to a knowledgeable reviewer?	Static analysis (L,DC)
Completeness		Are all requirements expressed in the design? Are all design elements implemented in the code? Are internals complete? (no missing logic, undefined variables, etc.) Are all aspects of real-world transactions implemented? Are boundary values and all combinations of conditions accounted for? Are recovery mechanisms implemented?	Static analysis (I) Static analysis (I) Static analysis (ASA,I) Functional (T) Functional (D,L,Se) Stress
Correctness		Does the product have statically detectable faults? Is the implementation/modification structurally correct? Is the implementation/modification functionally correct? Does the product perform correctly in heavy load situations? Have modifications had unintended effects on the behavior of the software?	Static analysis (All) Structural (All) Functional (All) Stress Regression
Integrity	Security	Are access control schemes appropriate? Are access controls and internal protections correctly implemented? Is end-user access management correct? Are access-related boundary values, logic, states, & syntax correctly implemented?	Static analysis (I) Structural (All) Functional (T) Functional (D,Sx,L,Se)

**Legend:**

ASA Automated Structural Analysis	I Inspection	Se State Testing
D Domain Testing	L Logic Testing	Sx Syntax Testing
DC Desk Checking	Lp Loop Testing	T Transaction Testing
DF Data Flow Testing	P Path Testing	

Software Quality	Also see:	Question to be Answered	Applicable Testing Strategies
Interface Consistency	Internal Consistency	Have interface standards & style been followed?	Static analysis (ASA,I)
		Is parameter & variable usage consistent across interfaces?	Static analysis (ASA,I)
		Is transaction data handled consistently among modules?	Functional (T)
		Are boundary conditions treated consistently?	Functional (D)
		Is message syntax consistent?	Functional (Sx)
		Is decision logic consistent among modules?	Functional (L)
		Are system states consistently treated among modules?	Functional (Se)
Internal Consistency	Interface Consistency	Have standards & style been followed?	Static analysis (ASA,I)
		Is parameter & variable usage consistent?	Static analysis (ASA,I)
		Are conditions handled consistently with respect to control flows?	Structural (P, Lp,D,L)
		Are there inconsistencies in data handling? (typing, mixed mode, I/O compatibilities, etc.)	Structural (DF)
Inter-operability		Are real-world events and logic handled consistently?	Functional (L, Se)
		Does the architecture facilitate interoperability?	Static analysis (I)
Performance		Do modules used in transactions exchange & use information properly?	Functional (T,D,Se)
		Is intra-module timing within specification?	Structural (P,Lp)
		Are transactions performed within required times?	Functional (T)
		Are timing requirements met when boundary values are input?	Functional (D)
Preciseness	Accuracy	Is system performance adequate under heavy load conditions?	Stress
		Will internal representations yield required precision?	Static analysis (DC)
		Are internal calculations sufficiently exact?	Structural (DF)
Reliability	Availability	Are real-world transaction results sufficiently exact?	Functional (T)
		What is the probability of running without failure for a given amount of time?	Statistical

**Legend:**

ASA Automated Structural Analysis  
D Domain Testing  
DC Desk Checking  
DF Data Flow Testing

I Inspection  
L Logic Testing  
Lp Loop Testing  
P Path Testing

Se State Testing  
Sx Syntax Testing  
T Transaction Testing



Software Quality	Also see:	Question to be Answered	Applicable Testing Strategies
Robustness		Has appropriate recovery logic been implemented?	Static analysis (I)
		Are poorly specified/invalid transactions handled correctly?	Functional (T,Sx)
		Are marginal/illegal inputs handled correctly?	Functional (D,Sx)
		Are unexpected combinations of conditions/states handled correctly?	Functional (L,Se)
		Can the system continue operating outside of normal operating parameters?	Stress
Security	Integrity	Are access controls properly designed/implemented?	Static analysis (I)
		Are access controls consistent with the operating environment?	Static analysis (I)
		Are the structural aspects of access control mechanisms correct?	Structural (All)
Security (continued)		Do access management functions work correctly?	Functional (T)
		Do access management functions work correctly in the presence of marginal or illegal values and constructs?	Functional (D,Sx,L,Se)
Simplicity		Are implementation solutions overly complex?	Static analysis (I)
		Are complexity-related metric values reasonable for a given situation?	Static analysis (ASA,I)
Testability		How can aspects of the software be tested?	Static analysis (DC,I)
Understandability	Clarity	Is the designer/implementer intent clear?	Static analysis (DC,I)
		Does information characterizing the software make sense?	Static analysis (ASA,I)
Usability	User friendliness	Can the user correctly form, conduct, & interpret results of transactions?	Functional (T,D,Sx)
		Does the user interface design support operational procedures?	Static analysis (I)
User friendliness	Usability	Is the user comfortable in forming, conducting, and interpreting results of transactions?	Functional (T,Sx)
Validity	Acceptability	Are requirements traceable?	Static analysis (I)
		Are implementation solutions appropriate?	Static analysis (DC,I)
		Is the real world appropriately represented?	Functional (All)
		Is the implementation/modification structurally correct?	Structural (All)
		Is the implementation/modification functionally correct?	Functional (All)

**Legend:**

ASA	Automated Structural Analysis	I	Inspection	Se	State Testing
D	Domain Testing	L	Logic Testing	Sx	Syntax Testing
DC	Desk Checking	Lp	Loop Testing	T	Transaction Testing
DF	Data Flow Testing	P	Path Testing		



---

## Appendix P Human Factors

---

Human factors (HF) is an important subtopic of the greater issue of software and digital I&C safety and reliability. The field of HF (in regard to NPP applications) is quite nebulous at this point, so there was little pertinent literature from which to develop a focused work solely on this topic which could lead to any substantive conclusions. However, there has certainly been enough exploration of the HF topic that it should not be glossed over either.

HF factors issues are considered in the BTP-14 software review process, but they do not come across as integral to the process. There is an HF organization that is separate from the HICB, the organization that deals with the software issues we have discussed so far (as in BTP-*HICB* 14). The HF branch is primarily responsible for evaluating HF issues in the overall scheme of SRP Chapter 7 (the I&C review process). Their efforts are supposed to be coordinated with those of the HICB (and numerous other branches), but human factors have too long been relegated to the position of an *additional* topic in software design and review. They must be made *integral* to the process. As mentioned before, many of the problems with BTP-14 are simply in its organization and presentation rather than in content. Likewise, BTP-14 does not ignore HF issues, but it needs to incorporate them more directly into its guiding principles.

### Background

“At this time, there does not seem to be an agreed-upon, effective methodology for designers, owner-operators, maintainers, and regulators to assess the overall impact of computer-based, human-machine interfaces on human performance in nuclear power plants. What methodology and approach should be used to assure proper consideration of HF and human-machine interfaces?”<sup>P1</sup> (References for this Appendix are listed at the end of the appendix as P1 through P12.) The preceding quote comes from a presentation on the safety and reliability issues of digital instrumentation and control systems in nuclear power plants by Douglas M. Chapin, Committee Chairman of the National Academy of Sciences group that was formed to look at such issues. Unfortunately, this statement is a fair summary of the current lack of understanding and agreement on this topic. A survey of some of the theoretical and experimental research that has been conducted in this area will show that, at least at this point in time, there are no “right” or “wrong” answers regarding HF design considerations. There are many different solutions, each of which hopefully contributes something that brings us closer and closer to an ideal HF-designed system.

The importance of a proper understanding of HF issues from an engineering and design perspective (e.g., the impact of I&C failures on human error probabilities) can be readily observed in the

relatively recent accidents involving the Therac-25. HF problems in the case of this medical radiation administration device included the following:<sup>P2</sup>

- \* Error messages displayed to the users were cryptic,
- \* Conveniences added to make the keypunch procedure easier for the operators led them to carelessly proceed after an error rather than to determine the cause of the error and start over, and
- \* Hardware interlocks (from previous Therac designs) designed to prevent catastrophe if the user set the system up incorrectly were removed in the false thought that the new software design could accomplish the same task.

### Theory and Research

To understand the importance of understanding HF, consider the Therac-25 case and the HF tie-in. The Therac-25 is a medical device used for chemotherapy administrations. Over twenty separate incidents with this machine led to a range of injuries in 172 patients (from minor injuries to, in one case, death). A variety of causes, most revolving around poor HF design issues, were identified.<sup>P2</sup>

Sheng et al. discuss the concept of “experimental frame,” one of three fundamental elements of an experiment (the other two being the “object” -- either a system or a system model -- and the generated/collected/ observed “data”). The experimental frame consists of five entities which are manifested slightly differently in model simulation versus real-world experimentation. These five entities are: observational variables, input schedules, initialization, termination conditions, and data collection/compression specifications (see Figure H1). Each entity has constraints imposed upon it by the real world (or by the model), so that a model may be valid with respect to some experimental frames but not to others.

Now, what is the connection between all of this and HF? “Experimental frame” touches on the issue of context, which is often overlooked in creating the requirements, specifications, and designs for software. As can be seen in the Therac-25 case, the context and conditions under which software is being used can be an important factor in determining whether the software will operate reliably or not.

In the case of the Therac-25, it was an unforeseen combination of the following conditions which led to the radiation overdose incidents:<sup>P2</sup>

- \* Poor man-machine interface (MMI) in the form of unclear error messages,
- \* Operator disregard for error messages,
- \* Operator impatience,
- \* Misplaced faith in previous procedures by the operator, and

- \* *Most importantly*, improper application of software technology to replace hardware interlocks on the Therac-25 (which would have prevented the accident from occurring in the first place).

It is helpful to look at a cross-section of the work that has been done in the HF area to understand its current state and to determine what work still needs to be done. There is a lot of lingo used in this field, and even basic definitions in the HF field are not agreed upon by everyone. Error can be defined as “an action that fails to meet some implicit or explicit standard of the actor or of an observer.” In many models of error there is either error or no error -- nothing in between. Yet many support the use of degrees of error, as this is more reflective of real life. An error may or may not lead to an accident (defined as an “unwanted and unwonted exchange of energy”) Classes of errors and error causes are also vague areas, where the appropriate classification may depend on the particular application. Taxonomies include (among many others):

- \* Errors of *omission* vs. errors of *commission*,
- \* *Intended* errors (mistakes) vs. *unintended* errors (slips, lapses), and
- \* *Forced* errors (tasks greater than capabilities) vs. *random* errors.

Likewise, there are many different classifications of error *causes*. These include:

- \* *Capture* (where the operator does an unintended sequence of actions almost automatically before he can catch himself because it is a common sequence),
- \* *Hypothesis verification* (looking only for evidence which supports one’s hypothesis), and
- \* *Risk (error) homeostasis* (i.e. take riskier actions in safer systems and vice versa so that the overall risk level stays the same).

Some basic HF issues are addressed in an article by Sheridan entitled “Understanding Human Error and Aiding Human Diagnostic Behavior in Nuclear Power Plants.” He addresses the inherently contradictory goals and purposes of a reactor operator which make his job so difficult: for example, protecting the public from dangerous radiation exposure while also avoiding the economic impacts of shutting the plant down, and following prescribed rules but also being able to think resourcefully if need be.<sup>P3</sup> Unfortunately, most simulator training that reactor operators receive addresses situations for which there are prescribed procedures, but not unanticipated events that call for resourcefulness. The article also points out that, although many reliability analysts try to treat human error and machine error in the same manner, there are significant differences between them which must be considered, including:<sup>P3</sup>

- \* Defining what constitutes an error (e.g. the action itself or the consequences of the action? -- In other words, if an error is made but also caught and corrected before there are any negative consequences, is it officially counted as an error?)

- \* Explaining causes of human error
- \* Classifying types of errors
- \* Common-mode errors and related follow-on errors (e.g., more errors may be made once discovering that a redundant system intended to backup a failed system has also failed)
- \* Opportunity for error harder to define for humans than for machines (e.g., for a machine it could just be the number of times a switch was activated)
- \* Correction of error (humans do correct errors; machines do not)
- \* Defining criteria for human takeover from automatic systems
- \* Error data base -- little data for humans; often not conclusive

Various methods have been attempted to model HF and human errors in accident scenarios. One example is the “dynamic logical analytical methodology” (‘DYLAM’), which treats slips and mistakes deterministically (based on empirical data) and treats lapses stochastically (a more Bayesian probabilistic method). It is designed more for hardware though, so its stochastic treatment of the human operators is somewhat weak. Another system, called the “dynamic event tree analysis method” (‘DETAM’), is better at addressing human states and interactions with the rest of the system. The human-related information that this method incorporates is “diagnosis state” (the operating crew’s accident situation assessment), “quality state” (internal state related to stress and other performance shaping factors), and “planning state” (based on what procedure the crew is planning).<sup>P4</sup>

Many studies have been conducted to establish guidelines for properly designing systems with HF issues in mind, the lowest (most specific) level of these studies focusing on how to create a good human-computer interface (e.g. the screen display, the mouse, other input devices, etc.).

Sheridan summarizes some recommended strategies for prevent “bad errors.” These strategies include:<sup>P3</sup>

- \* Error-preventative designs (e.g., feedback on both immediate actions/errors and longer time-scale plant state issues)
- \* Illustrative computer aids and system displays
- \* Attention to personal and cultural issues affecting operator performance (e.g., in Europe, a switch is “on” if it is “down” and “off” if it is “up,” opposite the American convention)
- \* Information redundancy
- \* “Fail safe” or “fail soft” designs (with built-in mechanisms to prevent catastrophic failures and to continue operation under degraded conditions)

- \* Operator training that encourages self analysis of cognitive errors, coping with new situations, and maintaining seldom-used skills, and
- \* Appropriate warnings and alarms.

Sheridan also provides recommendations for control panel displays. These recommendations include:<sup>P3</sup>

- \* Consolidating correlated components, signals, and alarms into fewer integrative displays (with optional access to more detail) to cut down on information overload
- \* Easy and logical methods to find “hidden” information and bring it to the viewscreen
- \* Overview portions of the display with important whole-plant-state conditions
- \* Adaptive display formats for changing situations, and
- \* Decision-aiding capabilities, which actually suggest options to the operator.

#### **Situational Awareness, Control, and Operator Aid**

There is a lot of debate over just how much automation should be used in the nuclear power plant control process. This debate stems again from the issue of situational awareness. One experiment in this area results showed that for low workload situations involving few and/or simple displays, control should be manual with the operator fully ‘in the loop.’ However, for high workload events involving complex displays, automatic control is preferable.<sup>P5</sup> A similar experiment reached a conclusion that does not involve the issue of high or low workload (and the difficulty of defining those terms precisely). It was found that, since manual control involves selecting and executing manual tasks, and since motor activities interfere with such processes, heavily manually-intensive scenarios would be controlled more effectively with automation (than with manual control). Conversely, since automatic control involves perception and central processing, mentally taxing situations should be controlled manually.<sup>P6</sup>

A variety of research and experimental work has been done with HF issues in control panel display design and in design of decision-aiding systems, which are often referred to as “Intelligent Decision Support Systems” (‘IDSS’). IDSS has both advantages and disadvantages. On the positive side, IDSS can help to correct misdiagnoses and cognitive lockup by continually offering relevant alternative hypotheses. The IDSS can also help the operator to maintain an accurate system mental model by providing appropriate parameter information and graphical displays.<sup>P7</sup> Human operators tend to focus on a few select gauges or variables and develop ‘tunnel vision’ around them. IDSS can help them to follow a more logical system of display monitoring. Under normal operations, the operator should scan a wide array of generally uncorrelated variables. As soon as an indication of a problem arises, the operator should start to monitor variables correlated more closely to the abnormal variable in order to determine a cause or to find a trend.<sup>P8</sup>

These first two functions (correcting misdiagnoses; maintaining a mental model) keep the operator in the loop so that he does not lose his situational awareness. The benefit of all of this aid is to make the operator more efficient so that he will be effective in time-critical crises. The foremost drawback of such diagnostic systems is their potential to do too much and thus to take the operator out of the loop, possibly causing him to lose his own skills.<sup>P7</sup>

There are many examples of such diagnostic aiding systems from the past few years which shed some light on what functions prove to be truly useful to reactor operators. One popular option is an "expert system" based on some type of 'library' of accident scenarios and appropriate responses. Experts in the field create these "plan libraries" which are then used to guide a reactor operator through an accident situation by comparing his actions to those stored in the library.<sup>P9</sup>

A unique decision-support experiment relating to control panel design was performed on the MIT Research Reactor (MITR). Researchers compared displays of various combinations of derivative, current, and predictive plant state information. This research was based on the notion that people make control decisions and implement control actions based on a comparison of their anticipation of the system behavior and the desired system response. The display layouts compared were: 1) current information only (in numeric form), and (in graphical form) 2) current and derivative [past] information, 3) current and predictive information, 4) derivative, current, and predictive information, and 5) limited-derivative, current, and predictive information. (Limited derivative information consists of only a set amount of the reactor's past power profile rather than the entire profile since startup.) Experimental subjects were asked to take the reactor from some initial power to a different final power by either moving control rods in or out or leaving them stationary.<sup>P10</sup>

Scheme 4 with all three types of information gave the best results, followed closely by the current and derivative combination (scheme 2). The other three schemes showed significantly worse results. Once trained on the usage of the predictive information, all of the operators agreed that it was useful, especially during the final stages of a transient or during a particularly unusual situation. Displays including predictive information have been suggested for such operations as control of the water level in pressurized water reactor steam generators and other control situations where the system behaves in a non-linear, time-delayed, or counter-intuitive manner.<sup>P10</sup>

This system and others like it can lead to a distillation of some of the major points to be considered in determining the effectiveness of an operator-aiding or expert system. The system must provide the operator with the necessary process information in such a way that it does not contradict his own thought and logic process. The operator must be able to trust the expert system and believe that it has the necessary and sufficient resources to provide competent help. At the more concrete level of the man-machine interface, the display should be clear and understandable to all of its possible users.<sup>P11</sup>



### Recent Human Factors Developments

In the U.S., developments involve system upgrades (analog to digital) rather than construction of entirely digital systems. All operating U.S. plants and many others worldwide have installed the Safety Parameter Display System (SPDS), which (along with other plant safety-related instruments) helps the control room crew to determine plant safety status during any plant condition. The SPDS presents the status of various critical safety functions symbolically and graphically. The SPDS, since it is so important to plant safety, is isolated from other parts of the plant's computer system to increase its reliability (i.e. so there is less chance of some type of common-mode failure). Because the SPDS systems have been *backfitted* into U.S. plants, they are not blended into the control room from the design stage, violating one tenet of HF engineering. This, however, is the best that can be achieved under the circumstances. Many other countries have made similar efforts in digitizing their control rooms, including Hungary, Germany, Slovenia, the UK, France, and Italy. They have incorporated a variety of concepts, some of them already described: color-coded graphic displays, hierarchical display screens, and a large wall display for the common viewing of control room personnel.<sup>P8</sup>

Alarm processing is an area that has been being researched in many different countries. Some common HF themes have developed in this area. These themes include intelligent filtering and logic to prevent unneeded alarms (and to suppress those alarms that are related to previous alarms), prioritization of related alarms, and alarm sequence recording for future analysis. A color coding system is also sometimes used for symbolic panel displays.<sup>P8</sup>

Another area of great research is task allocation between man and machine, a critical step in the design process that, if done improperly, can ruin an otherwise good HF-based approach. In general, humans should be allocated tasks involving "inferential knowledge" (computers cannot infer anything that is not there), changing circumstances (requiring creativity and flexibility), or inordinately large automation costs. Machines should be allocated tasks requiring large amounts of repeated, rapid, accurate data processing or tasks done in high risk areas (i.e. highly radioactive areas). Italy and some other countries have developed passive plant safety controls which can protect the plant itself and prevent radioactive releases within the first 24 to 72 hours after an initial emergency with no operator actions. This takes the time pressure off of the operators and allows them to function as "intelligent supervisors."<sup>P8</sup>

One of the most difficult HF issues being dealt with is how to account for human and team interactions and interdependencies in making decisions and taking actions. One method that may be able to do this is the Dynamic Flowgraph Methodology, or DFM. DFM is a digraph model-based technique for expressing logical and temporal characteristics of a system (both its software and its hardware) in order to construct fault trees that identify paths leading to potentially critical system events.<sup>P12</sup>

DFM has been proposed to incorporate "team process variables within a dynamic evolution" into the human reliability analysis (HRA) portion of a probabilistic risk assessment (PRA). Most models used previously take into account only "one operator [at a time] performing an isolated set of tasks." *Team*

*interdependencies* (e.g., communication, coordination, and adaptability) are not often considered, but they can be a big factor not only in team actions but also in team decisions. Such dependencies can be considered in DFM. Dependencies between people are not the only dependencies considered; those between people and hardware can also be evaluated.<sup>P12</sup>

Causal relationships are established between physical variables and human relationships. Classes of operator actions with risk significance are determined (e.g., routine actions before an initiating event, actions that can cause initiating events, recovery actions), and then various specific operator actions are postulated. Quantitative error frequencies for these actions are determined from operator input through such methods as the Success Likelihood Index Methodology (SLIM). Various cognitive problems (e.g., confirmation bias, anchoring bias, and others previously discussed) are considered due to the tendency for operation of a system as complex as a nuclear reactor to lead to cognition errors. Ultimately, decision tables are created which show how the values of different process variables effect the outcome of team performance. An example is how the (a) task complexity, (b) training level, and (c) technical knowledge of the team (all three of which are divided into only a few possible states, or values) combine to form the “knowledge, skills, and abilities” value, which in turn has an effect on such things as the team’s “formation of intent” (i.e., intent to perform an action).<sup>P12</sup>

## Conclusions

As can be seen, there has been a lot of research into both theoretical aspects of HF design and practical implementations of it. Human factors have long been neglected in engineering design processes. When they have been included, it has usually been as an afterthought to a design project, at which point it is too late to properly incorporate HF principles into the design. Much more attention has been paid to human-machine interactions in recent years, and advances have been made.

Unfortunately, most aspects of human factors are so subjective and debatable that it is difficult to define a ‘best’ HF design. Nonetheless, a more detailed treatment of HF issues (such as the DFM version just discussed) needs to be built into the SRP Chapter 7/BTP-14 review process. Human factors include much deeper and more profound issues than just graphical user interfaces and control knob layouts. (Because HF is just one sub-issue of the greater topic of this work and because it is still very hazy compared to many of the other issues involved, no more conclusive recommendations will be given on this topic in the conclusions.)

## Human Factors References

- P1. Chapin, Douglas M., “Digital Instrumentation and Control Systems in Nuclear Power Plant and Safety: Safety and Reliability Issues,” presentation on 10/4/95.

- 
- P2. Leveson, Nancy G. and Clark S. Turner, "An Investigation of the Therac-25 Accidents," *IEEE Transactions on Software Engineering*, July 1993: 18-41.
- P3. Sheridan, Thomas B., "Understanding Human Error and Aiding Human Diagnostic Behaviour in Nuclear Power Plants," *Human Detection and Diagnosis of System Failures* (Proceedings of a NATO Symposium on Human Detection and Diagnosis of System Failures held August 4-8, 1980, ed. by Jens Rasmussen and William B. Rouse). New York City: Plenum Press, 1981.
- P4. Acosta, C. and N. Siu, "Dynamic Event Trees in Accident Sequence Analysis: Applications to Steam Generator Tube Rupture," *Reliability Engineering and System Safety*, vol. 36, no. 3, 1992.
- P5. Ephrath, Arye R. and Laurence R. Young, "Monitoring vs. Man-in-the-Loop Detection of Aircraft Control Failures," *Human Detection and Diagnosis of System Failures* (Proceedings of a NATO Symposium on Human Detection and Diagnosis of System Failures held August 4-8, 1980, ed. by Jens Rasmussen and William B. Rouse). New York City: Plenum Press, 1981.
- P6. Wickens, Christopher D. and Colin Kessel, "Failure Detection in Dynamic Systems," *Human Detection and Diagnosis of System Failures* (Proceedings of a NATO Symposium on Human Detection and Diagnosis of System Failures held August 4-8, 1980, ed. by Jens Rasmussen and William B. Rouse). New York City: Plenum Press, 1981.
- P7. Norman, Donald A., "New Views of Information Processing: Implications for Intelligent Decision Support Decisions," *Intelligent Decision Support in Process Environments* (Proceedings of a NATO Advanced Study Institute on Intelligent Decision Support in Process Environments held September 16-27, 1985 ed. by E. Hollnagel, G. Mancini, and D.D. Woods). New York City: Springer-Verlag, 1986.
- P8. *New Man-Machine Interfaces in Nuclear Power Plants*. Paris: Nuclear Energy Agency Organisation for Economic Co-operation and Development, 1994.
- P9. Hollnagel, Erik, "The Design of Fault Tolerant Systems: Prevention is Better than Cure," *Reliability Engineering and System Safety*, vol. 36, no. 3, 1992.
- P10. Lau, Shing Hei, John A. Bernard, Kwan S. Kwok, and David D. Lanning, "Experimental Evaluation of Predictive Information as an Operator Aid in the Control of Research Reactor Power," presented at the 1988 American Control Conference, Atlanta, GA, June 15-17, 1988.
- P11. Bernard, John A. and Takashi Washio, "The Utilization of Expert Systems within the Nuclear Industry," presented at the 1989 American Control Conference, Pittsburgh, PA, June 21-23, 1989.
- P12. Campbell, S.D., G. Apostolakis, and J.-S. Wu. "Modeling Operator Teams in Probabilistic Safety Assessment."