RESEARCH INFORMATION LETTER 1001:
Software-Related Uncertainties in the Assurance of Digital Safety Systems—Expert Clinic Findings, Part 1

Executive Summary

This research information letter (RIL) transmits knowledge about uncertainties in assurance of digital safety systems associated with software[1] and other manifestations of complex logic (e.g., field-programmable gate array (FPGA)). The purpose of this transmittal is to support the judgment exercised in licensing reviews of complex digital safety systems. This knowledge was acquired through an expert elicitation activity[2] conducted by the U.S. Nuclear Regulatory Commission (NRC), Office of Nuclear Regulatory Research (RES), Division of Engineering (DE) (hereafter referred to as NRC/RES/DE).

Uncertainties in the assurance of digital instrumentation and control (DI&C) systems for safety functions in a variety of application domains are increasingly emanating from systemic causes, as in software. In order to learn from experiences outside of the commercial nuclear power plant (NPP) industry, such as defense, space flight, commercial aviation, medical devices, automobiles, telecommunications, and railways, NRC/RES/DE elicited knowledge from experts with safety-critical software and systems research experience in these application domains. NRC/RES/DE employed the Pacific Northwest National Laboratories (PNNL) as a neutral elicitation agent. PNNL interviewed over 30 experts spread across seven countries (United Kingdom, Sweden, Germany, Canada, United States, Australia, and New Zealand), from which a diverse group of 10 was charged with the following objectives in a 2-day clinic:

- Identify limitations in the current state of practice (i.e., sources of uncertainty) that make software assurance heavily dependent on expert judgment.
- Identify the evidence that is needed to assure software for safety more effectively, based on best practices in other application domains.
- Identify knowledge gaps to be filled (i.e., areas in need of research and development) to enable more consistent reviews and to reduce judgment-based variation.

Highlights of the expert clinic findings are summarized below.

1. Summary of Current State

DI&C system safety assessment will continue to require high-caliber judgment from a diverse team[3] commensurate with the complexity of the system and its development process and environment, as in systems containing complex software or other manifestations of complex logic. In order to exercise reasonable judgment, the team will review the types of evidence[4]

---

[1] Although the expert elicitation activity was targeted to focus on uncertainties in software assurance, the findings are applicable more broadly to potential defects in the DI&C system attributable to engineering mistakes or defects in engineering tools.

[2] This activity was part of the NRC's 2010–2014 DI&C research plan.

[3] A diverse team should be composed of individuals with complementary attributes (e.g., thought processes, communication styles, and competence) needed to perform the assigned task, including education, training, and experience in different domains and disciplines.

[4] Also see Appendices A.1–A.6.

identified in this report, integrated with reasoning to demonstrate[5] that the remaining uncertainties will not adversely affect system safety. In the absence of such demonstration, there should be diverse defensive measures[6] independent from digital safety systems using complex software, other implementations of complex logic, or products of software-intensive tools.

2. Seek Diverse Complementary Evidence to cover gaps and uncertainties

Because of the complexity[7] of the digital systems[8] being proposed in new licensing applications in the USA, claims of *complete* certitude about safety assurance would not be credible. To the extent that complete certitude of assurance cannot be obtained from product verification alone, a combination of different types of evidence[9] is recommended for complete coverage, complementing or filling the respective gaps and uncertainties, including diversity to improve the confidence in the coverage. No single approach (testing alone or process audit alone) can provide adequate assurance. It is necessary to integrate the evidence, with supporting arguments, to demonstrate reasonable assurance of safety. This RIL refers to this evidence-argument-claim integration structure as a "safety demonstration framework[10]"; some current uses of an evidence-argument-claim integration structure are known as an "assurance case" or, for a complete system, a "safety case" [1].

3. Complexity

Increasing complexity, among other issues, increases the potential for hidden or unsuspected dependencies or couplings, including interference. Establishing criteria (e.g., architectural) for avoiding unnecessary[11] complexity will support reducing uncertainty resulting from system complexity.

4. Interference

Demonstration of noninterference of system, subsystems, or elements is important; there is a need to establish criteria (e.g., architectural).

5. Change Impact Analysis

Analysis of the impact of change on safety assurance is difficult, especially as the DI&C systems and their development processes become more complex.[12] A complete understanding of all dependencies, the awareness of change in any dependency factor, and the analysis of its effect are all necessary. Architectural constraints will support such analysis.

---

[5] See Section 4 Safety Demonstration

[6] This recommendation is consistent with and supportive of current NRC diversity and defense-in-depth policy.

[7] relative to electromagnetic relay logic

[8] especially, with runtime software exhibiting dynamically altering behavior, with interconnections across elements of different degrees/levels of qualification and interconnections across redundant elements.

[9] e.g. expert group reviews to validate requirements and constraints; verifying satisfaction of design constraints or rules; model verification; analysis; simulation; coverage-based testing; process audits. See Table 2.

[10] See Section 4 Safety Demonstration

[11] "Everything should be made as simple as possible, but no simpler"—Albert Einstein.

[12] For example, as electronic hardware becomes obsolete (unavailable or unmaintainable) sooner, platform software correspondingly changes more often. The cumulative effect of a series of small or subtle changes becomes progressively more difficult to analyze.

6. Architecture

System-constraining criteria[13] and software architecture can reduce complexity, assure freedom from interference, reduce the interaction of different sources of uncertainty, reduce the impact of change, and improve verifiability. Thorough reviews can assure correctness in deriving architectural constraints and specifications from quality of service properties, such as safety, reliability, and robustness. It is recommended that reviews also examine whether assumptions about the environment are validated.

7. Tool-Automated Processes

Software development technology is headed toward automation of routine, well-defined, repetitious, or tedious tasks through tools such as for automatic generation of code, automatic generation of test cases, and automated testing. However, any such tool, by itself, should not be accepted[14] as the sole contributor of evidence for safety assurance, without published formally defined semantics and certification and qualification of their correctness and fitness for purpose. Although such tools can eliminate the possibility of mistakes that people make in routine tasks, tools can also add new defects[15] in ways that are outside the intuition and experience of the designers and regulators. When tools are used in a chain of transformations from various levels of abstract representations to executable code, evaluation of the chain, as a set, for semantic consistencies across transformations is recommended.In the current state of the art, in addition to certification and qualification of tools, diverse evidence, such as independent checks of the tool outputs or complementary evidence from various process activities[16], supports providing adequate coverage.

8. Follow-On Expert Elicitation Activities

Recognizing that the scope of these problems is large and their resolution a long-term endeavor, it is recommended the NRC undertake similar expert elicitation activities with more domain-specific information,[17] engaging specific expert groups for topics such as assessment and audit of tool-automated, tool-assisted processes for development, verification, and assurance. In order to focus the expert knowledge on problems being experienced or foreseen in the NRC (see [2]) and for regulatory improvement, these activities would benefit from including experienced NRC licensing reviewers. Thus, the resolution of these problems will be an iterative, evolutionary process, with commensurate growth in knowledge and active engagement of the stakeholders

---

[13] In scope of the NRC DI&C research plan [2], Section 3.1.5.

[14] In contrast, consider the example of tools used in creating logic targeted for FPGAs. Tools developed for non-safety consumer products are being used for safety applications without commensurate qualification guidance. On the other hand, it is economically prohibitive to develop tools for safety-critical applications from scratch.

[15] New defects may possibly result in new system failure modes.

[16] e.g. regression tests; verification of compliance with design constraints, implementation constraints, coding standards, etc..

[17] Domain-specific information might include representative system configurations, platforms, and applications.

Contents

Figures

Tables

1. Introduction

This RIL is the first in a series of three (RIL-1001, RIL-1002, and RIL-1003) that collectively respond to the DI&C-relevant part of the Commission's staff requirements memorandum (SRM) M080605B, "Meeting with Advisory Committee on Reactor Safeguards," dated June 26, 2008[18] [3], which asked the staff to do the following:

- Report progress in identifying and analyzing DI&C failure modes (to be addressed in RIL-1002, "Identification of Failure Modes in Digital Safety Systems and Analysis for Systemic Causes—Expert Clinic Findings, Part 2" [4]).
- Discuss the feasibility of applying failure mode analysis to quantification of risk associated with DI&C (to be addressed in RIL-1003, "Feasibility of Applying Failure Mode Analysis to Quantification of Risk Associated with Digital Safety Systems—Expert Clinic Findings, Part 3" [5]).

1.1 Context: State of Maturity of Knowledge in These Topics

These challenging issues can be addressed in long-term research, in collaboration with international regulatory researchers from other safety-critical application domains. The "NRC Digital System Research Plan FY 2010–FY 2014," dated February 26, 2010 (referred to hereafter as "DI&C research plan") [2], is structured to address these long-term research issues in an iterative, evolutionary manner, with several interrelated research activities, such as operational experience, analytical assessment, knowledge management, and probabilistic risk analysis. Figure 1 Lower half illustrates these inter-relationships, including the expert elicitation activities, being reported herein, and addressing the requirement from the Commission, depicted in the upper half of Figure 1.

[18] SRM M080605B resulted from the June 5, 2008, Advisory Committee on Reactor Safeguards (ACRS) presentation to the Commission about its review on DI&C issues.

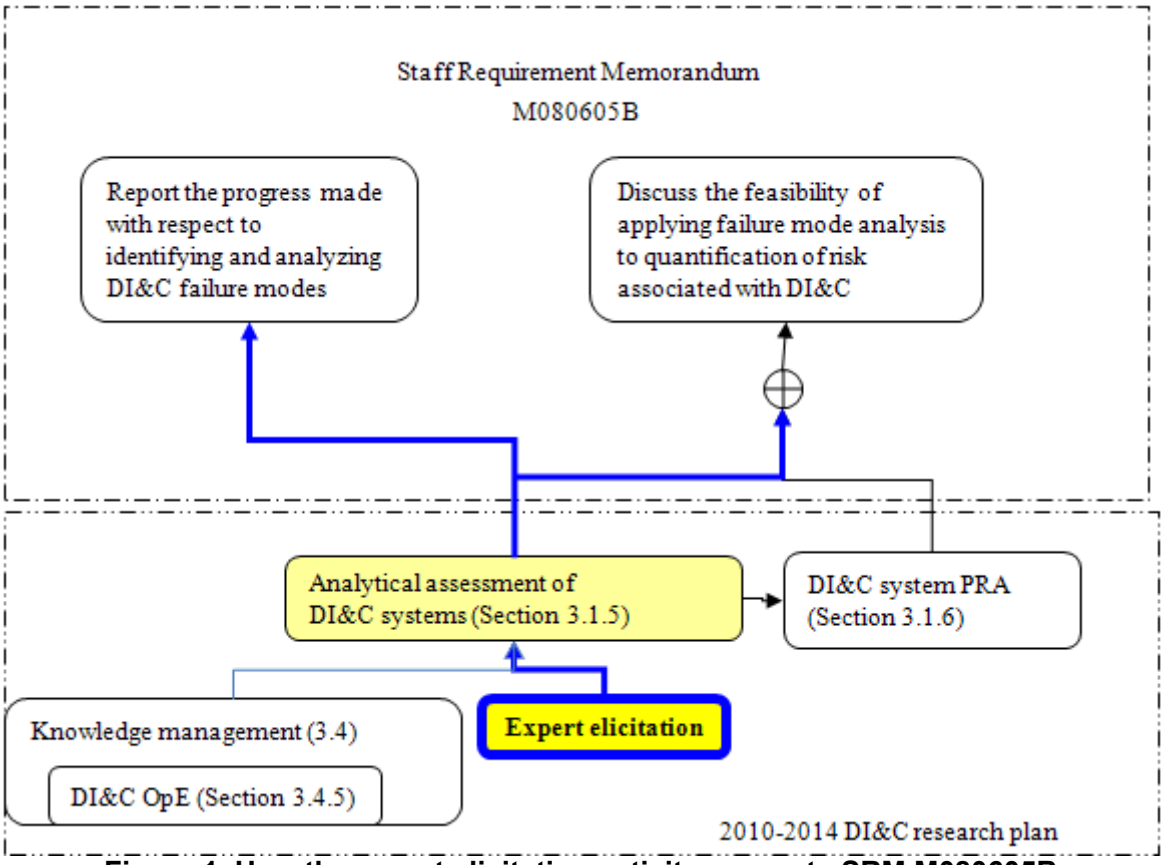


Figure 1 How the expert elicitation activity supports SRM M080605B

1.2 Organization of This Letter

Section 2, "Background," summarizes the generic issues that led to the SRM and this research. The main part of this RIL focuses on understanding the nature and sources of the uncertainties that make software assurance difficult even in a high-quality environment (e.g., process, architecture, organization, culture, and competence). Section 2 notes that, in this high-quality context, additional evidence would support reducing the extent of these uncertainties, and it connects some of these topics to corresponding International Electrotechnical Commission/International Organization for Standardization (IEC/ISO) and Institute of Electrical and Electronics Engineers (IEEE) standards.[19]

The understanding of the nature of uncertainties provided by this RIL establishes the knowledge platform for RIL-1002 [4], which will discuss failure mode analysis in the presence of such uncertainties.

Building on the information in these two RILs, RIL-1003 [5] will discuss the feasibility of applying failure mode analysis to quantification of DI&C system-failure likelihood resulting from software, other implementations of logic, other types of mistakes in the engineering of DI&C systems, defects in the tools used to produce the work products, and such other systemic causes.

[19] NRC regulatory guides reference many of these IEEE standards.

Baseline for discussion of uncertainties: The experts were focused on discussing uncertainties that are not well understood. Well-known sources of defects or uncertainties about the presence of defects were excluded from the scope of their discussion. For that purpose, this discussion of uncertainties is predicated on the assumption that DI&C systems and their elements satisfy the conditions and criteria outlined in Appendices A.1 Baseline Process–A.6 Safety Demonstration Framework to this RIL. It is not implied that the assumption holds for current DI&C systems, their elements, or their development environments.

Even in a high-quality environment as outlined in Appendix A to this RIL, there are many uncertainties in the verification and validation (V&V) of a DI&C system. Section 3, "Uncertainties in Verification and Validation of DI&C Systems," identifies these uncertainties, the limitations to overcoming these uncertainties, and recommendations to address these limitations. One of these recommendations is a logically structured organization of the claims, evidence, and arguments that establish their relationships, as introduced in Section 4, "Safety Demonstration." With the recommended evidence-integration R&D, it is the kernel of the approach to evaluate the resultant effect of the uncertainties identified in this RIL. Uncertainties in the V&V of a system arising from the use of tool-automated processes are addressed in Section 5, "Uncertainties Associated with Tool-Automated Processes." Uncertainties in the V&V of a system arising from change of any kind are addressed in Section 6, "Unknown Effects of Change." The limitations of the state of the art discussed in Sections 2–6 are also treated as knowledge gaps to be filled with further research. Section 7, "Miscellaneous Research Recommendations," includes additional long-term research recommendations. Section 8, "Next Steps," identifies some immediate next steps.

The glossary includes definitions of terms as used in this document. Supporting references provide more detail and contextual information about the conclusions and recommendations.

Appendix B outlines the elicitation process employed in this study.

2. Background

SRM M080605B [3] was triggered by ACRS concern [6] that attempts to quantify the likelihood of DI&C system failures might not yield useful information for safety assurance reviews because little is known about how digital systems fail. The background of this controversy is provided below, explaining the genesis of this study through the expert elicitation activity.

2.1 Differences in Judgment-Based Evaluation

In the safety review of software in a DI&C system, there are differences among various stakeholders' evaluations, partly driven by subjective differences in judgment. For example, in DI&C safety-related protection systems with four redundant trains to protect against hardware failure, but with identical software in each train, some stakeholders claim that the software is not a safety-significant common cause of failure because it has been developed with defensive measures, such as static allocation of computing platform resources. In contrast, the regulatory staff holds the position that the evidence to support this claim is not sufficient to provide reasonable assurance that these systems can perform adequately in service, leading to an expectation for a diverse defensive alternative.[20]

[20] For example, Finland's regulatory authority (STUK) seeks an actuation system not involving software.

2.2 Research on DI&C System Probabilistic Risk Assessment

The NRC's digital system research plan [2] addresses probabilistic risk assessment (PRA) methods and supporting data for digital systems. An important research need is to establish a commonly accepted basis for incorporating the behavior of software into digital I&C system reliability models for use in PRAs[21]. To address this need, the NRC sponsored a review of quantitative software reliability methods that might be used to support reliability modeling of digital systems of nuclear power plants [8]. Current NRC work in this area is focused on applying one or two candidate methods to an example digital protection system in a proof-of-concept study.

2.3 Challenges with Incorporating Software Behavior into Reliability Models

Two principle challenges to incorporating software behavior into digital I&C system reliability models are (1) the limited data from operational experience in the NPP industry and (2) the expected large uncertainty associated with the use of expert judgment. While the NRC's digital system research plan includes projects that attempt to address these challenges for the purposes of PRA, NRC/RES/DE staff believe that these challenges limit the usefulness of quantifying software reliability for the express purpose of supporting licensing reviews of complex digital safety systems (i.e., in reaching a reasonable assurance conclusion), therefore complementary NRC/RES/DE research addresses the question, "How can we extend the knowledge needed in reviewing software for likelihood of fault?"

2.4 Search for Alternative Qualitative Approaches for Software Assurance

Related to the challenges identified above, as they pertain to performance of licensing reviews, the vastness and nature of the uncertainty space, as identified in this study, precludes the possibility of identifying a relatively complete set of credible failure modes in software and in obtaining a statistically significant amount of observational data. This points to an analytical approach to support licensing reviews of safety system software. Accordingly, NRC/RES/DE sought other perspectives, focusing on the following question: "Are there systematized, analytical, qualitative approaches to informing the assurance process for systemic causes potentially leading to DI&C system failure?" Because experience with digital safety systems in commercial U.S. NPPs is limited relative to the issues underlying the concerns in SRM M080605B [3], NRC/RES/DE sought knowledge from other application domains through an expert elicitation activity (described in Appendix B to this RIL). The process was designed for the elicitation to do the following:

- Be supported with evidence, appropriately qualified for its degree of validity.[22]
- Synthesize complementary individual knowledge and apply qualitative reasoning.
- Use collective judgment to integrate and apply this knowledge to the NPP domain.
- Identify reasons for differences across experts (e.g., different underlying assumptions).

[21] See the relevant discussion in the Ref. 7 chapter titled, "Safety and Reliability Assessment Methods."

[22] Appendix B Expert Elicitation Process," establishes the integrity of the process, thereby supporting the validity of its results.

3. Uncertainties in Verification and Validation of DI&C Systems

Baseline for discussion of uncertainties: This discussion of uncertainties is predicated on the assumption that DI&C systems and their elements satisfy the conditions and criteria outlined in Appendices A.1–A.6 to this RIL. It is not implied that the assumption holds for current DI&C systems, their elements, or their development environments.

This section identifies sources of uncertainty in the verification and validation of DI&C systems, limitations to the state of the art, and recommendations to help reduce the uncertainties.

It is not credible that DI&C systems being considered for NPP safety functions are fully verified and validated because the combination of input space[23] and system-internal state space is too large (i.e., the systems are too complex) [9].[24] The uncertainty space is too large; to provide reasonable assurance of safety requires verification closer to certitude, in a manner that is based on safety goals[25] and oriented objectively.[26] Even a single defect[27] in software could have unforeseen consequences. All such individual defects, latent in the operational software, can combine in a large number of unforeseen ways and increase the potential for significant system failure.

Keeping in mind that preconditions identified in Appendices A.1–A.6 to this RIL will help reduce the confounding effect of interacting process variables, the rest of this discussion identifies additional conditions under which DI&C system verifiability can be improved to reduce complexity, improve analyzability, and reduce uncertainties.

Figure 2 depicts (with "?" symbols) that uncertainties exist in each phase of the system and software development lifecycles, including uncertainties associated with the use of tools and with any change affecting any of the process variables mentioned in Appendix A.1, "Baseline Process," to this RIL. In the figure, the size of the "?" symbol is a broad-brush indicator of relative uncertainty at this time. The statement, "each anomaly or uncertainty by itself seems to be insignificant," is applicable in general to any source, but the combined effect can be unexpectedly significant. Safety Demonstration in the presence of the various uncertainties is difficult.

---

[23] The size of the input space precludes coverage through testing alone. Analytical verification techniques at the design phase may provide coverage of the input space, but not coverage of uncertainties in the transformation of the design into operational code – see Section 5 and Appendix A.5.

[24] In a concurrent execution of multiple software components, a particular interleaving of events may produce the conditions under which a [fault] is uncovered, but it may be extremely difficult or impossible to reproduce that particular interleaving in a testing environment [RK, CW, DJ].

[25] It is assumed that the goal of NPP safety (prevention of unwanted release of radioactivity into the environment) is fully allocated to the DI&C safety system for the following reasons. The DI&C safety system in a NPP is an independent layer of defense, with the intent to cover for uncertainties and lack of certitude in the other layers. Thus, no credit is assumed from any other layer of defense. Although the NPP has four redundant trains of equipment, the DI&C systems are identical; the redundancy is ineffective in the software. Because design certification for DI&C platforms, tools, and processes is given without any constraint on the application environment, it is assumed that the same elements could be replicated for different NPP functions, entailing vulnerability to the same defects, weaknesses, or deficiencies.

[26] Rather than in a prescriptive manner.

[27] no matter how "small"

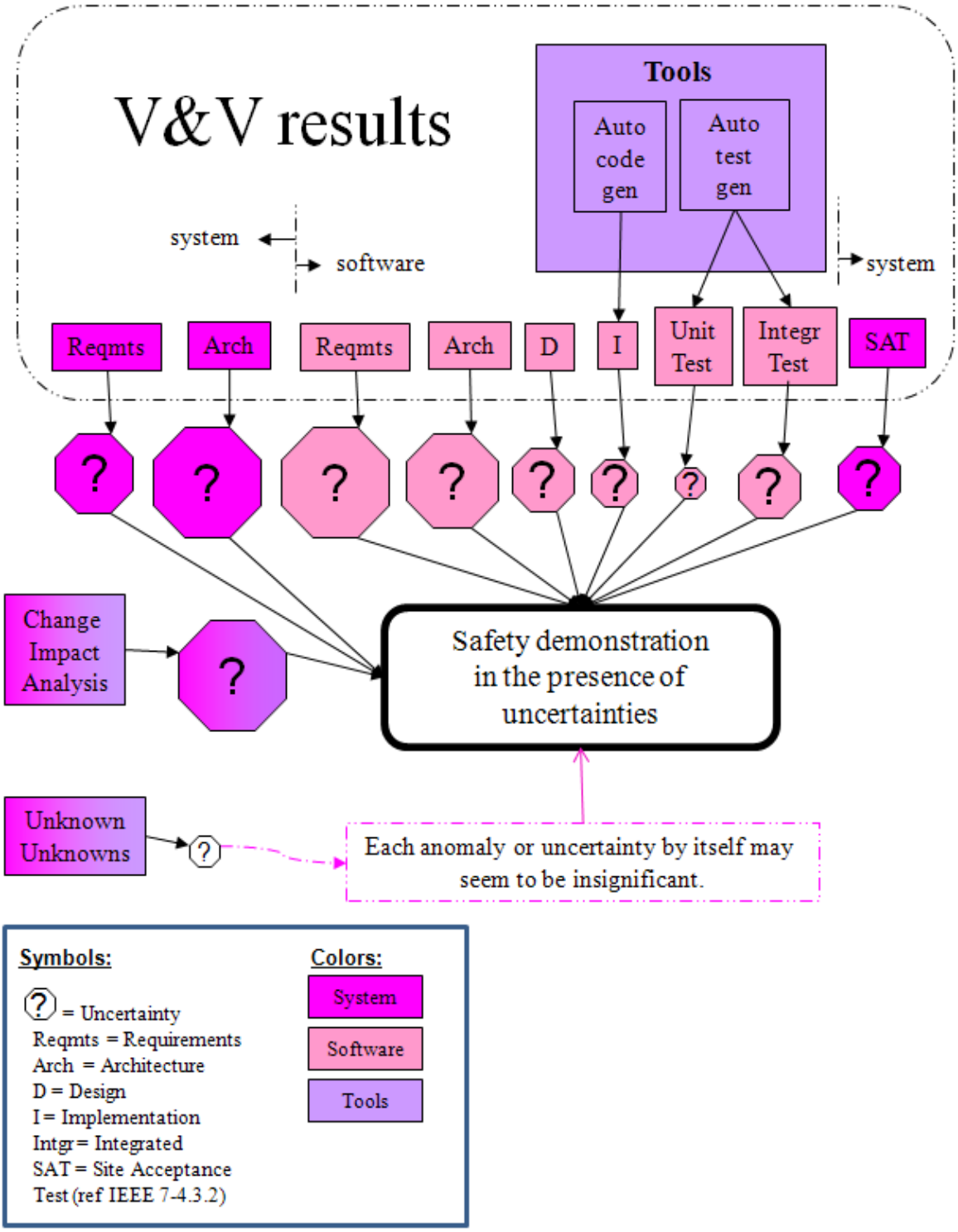


Figure 2 Integrating the effect of uncertainties in software assurance is difficult

Figure 3 shows some major sources of uncertainties, elaborated in Table 1, and the evidence that can reduce their impact, partly elaborated in Table 2.

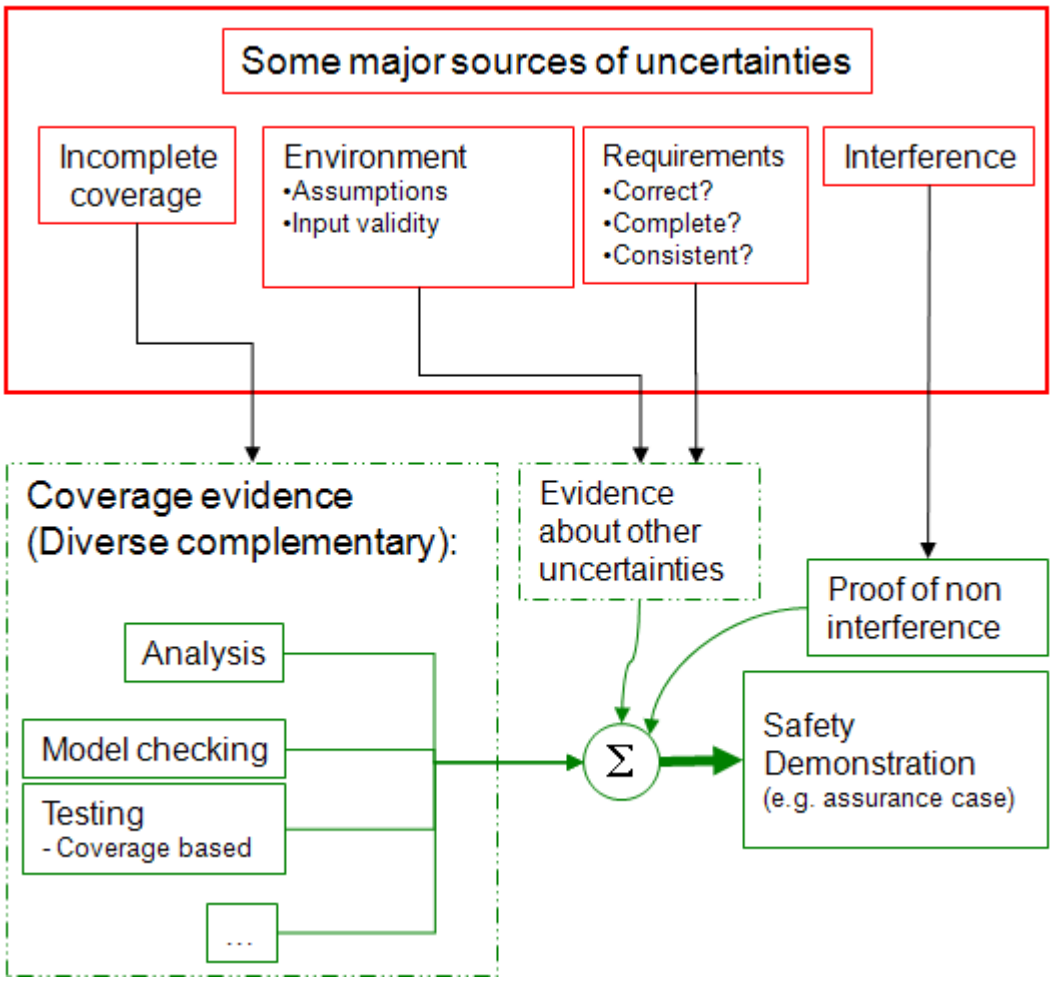


Figure 3 Uncertainties in V&V and evidence needed to reduce their impact

Traditional verification techniques do not address uncertainties about the environment, requirements, or interference—these uncertainties have to be addressed through better validation of the requirements, including derived requirements and constraints, and through assumptions about the environment.

Evidence proving that a system element does not interfere with another, for example, through an architecture that avoids unnecessary complexity and allows only well-behaved interactions, is conducive to demonstration of correct integration (see A.4 Baseline Architecture").

For those uncertainties addressable through verification, the issue boils down to coverage. Traditional testing of the final code is not enough by itself. Complementary forms of verification of earlier work products in the development lifecycle support improved coverage .

Table 1 Major Sources of Uncertainties in the V&V of a DI&C System and Its Software

ID#	Source of Uncertainty[a]	Remark
1	Assumptions and constraints concerning the application and its environment, including inputs	Better validation will reduce uncertainty, e.g., through reviews by diverse groups of experts Research is needed in domain modeling to include the environment and the assumptions.
1.1	If the object of verification is the DI&C system, then its environment includes the input and output devices and their valid signal space (value; timing) and their valid combinations.	
1.2	If the object of verification is the platform or system software, then its environment includes the hardware it is dependent on or with which it is interacting.	
1.3	If the object of verification is the application software, then its environment includes the platform it is dependent on the following, for example: 1.3.1 the timely availability of needed resources 1.3.2 assurance of freedom from interference with other components 1.3.3 correctness of operation in other ways	
2	Lack of completeness, consistency, correctness, and adequate validation of the requirements, including quality-of-service[b] requirements, derived requirements, and service requirements	Also, see A.3 Baseline Criteria for Specifying Requirements." Better validation will reduce uncertainty, e.g., through reviews by diverse groups of experts.
3	Incomplete coverage of verification. Combinations of individual defects (each, by itself, thought of as being small and insignificant) are rarely, if ever, considered exhaustively.	Testing can cover only a small portion of the input space in a feasible amount of time.
4	Defective performance of other resources employed in the process, such as the following: 4.1 humans 4.2 tools 4.3 information	Also, see A.5 Baseline Criteria for Evaluating Tool Automation."

[a] Predicated on satisfaction of the preconditions given in Appendices A.1–A.6.

[b] Commonly known as "non-functional"

Given the diversity in the sources of uncertainties discussed above, commensurate forms of evidence will provide adequate coverage support assurance (see Table 2). A range of diverse techniques are recommended, e.g. preventative approaches such as constraining the design space, exploiting abstraction, mechanized reasoning in the abstract space, and independent expert team review.

Table 2 Coverage of uncertainties through diverse, complementary forms of evidence

ID#	Evidence	Remark
1	Review, walk-through, and inspection (RWI)—particularly effective in validating[a] requirements or evaluating architectures	Also see: A.3 Baseline Criteria for Specifying Requirements" A.4 Baseline Architecture"
2	Bounding the space to be verified, such as the following.	
2.1	Design constraints, e.g. architectural	A.4 Baseline Architecture"
2.2	Exploiting abstraction through the use of appropriate modeling languages	
2.3	Disciplined use of a well-proven, "safe" subset of the various languages employed in each phase of the development lifecycle	See A.5 Baseline Criteria for Evaluating Tool Automation." Table 8, ID#s 3–4
2.4	Assurance that the respective language subsets (e.g., for requirements specification, architectural design, detailed design, implementation,[b] target hardware instruction set) are semantically consistent and the corresponding transformations preserve the source semantics[c]	See Table 8, ID#s 2–8
3	Reasoning—combination of machine-aided and human: logical; probabilistic	For key enablers, see Appendices A.2–A.5
4	Analysis—static, dynamic: mathematics-based	See A.4 Baseline Architecture" By using best in class analysis, which improves over time, the coverage can be improved and residual uncertainties, reduced [GH].
5	Use of insights gained from verification activity at an earlier stage for improvements, including downstream V&V plan revision. The earlier in the lifecycle that an issue is discovered, the more effective and efficient is the V&V.[d]	
6	Model checking, with coverage comparable to items 8.0–0 below	For key enablers, see Appendices A.2–A.5
7	Simulation, including respective stakeholders (e.g., developers and users)	
8	Testing (reduced with reasoning for coverage achieved), e.g.: 8.1 Covering all states, including fault states or abnormal conditions 8.2 Covering all paths 8.3 Covering all combinations of inputs (values; timing) 8.4 Covering specific scenarios of concern, including effects of failures 8.5 "Corner" cases 8.6 Stress testing 8.7 Retesting after each change[e] or corrective action	The NRC has a related research project on fault injection techniques and tools

ID#	Evidence	Remark
[a] Mathematical proof systems today are very powerful, but they can prove only according to the input they get. Therefore, we still have the validation problem [MB]. [b] [Even if an organization has a coding standard,] don't just assume that it is being adhered to [DD]. Compliance should be verified, e.g. through tool automation. [c] …Systems…today…rely on...many different…implicit assumptions, often about how you produce a piece of software…[e.g.,] about the semantics.… Some of these levels are inherently difficult and complex, and error prone…. It is very difficult to assure that an argument which you can formulate on a particular level of the stack can be transmitted to the next level…arguments [from one level of abstraction to another] don't fit together properly [MB]. [d] Conversely, if the root cause of a defect is in an earlier stage of the lifecycle, the later the discovery, the more difficult it is to find the root cause and to understand its impact on system safety [JC]. [e] Try to make regression test/verification suite sufficiently complete to detect new errors introduced by change.		

The types of evidence listed in Table 2, esp. for ID# 8, are not exhaustive. The types of evidence used for a particular system will depend upon its specific requirements and design choices; not all types listed in Table 2 ID# 8 may be needed. V&V continues to depend upon expert judgment because of limitations in the state of the art (see Table 3 for examples).[28] In particular, expertise is needed to review the V&V plan to examine whether the plan provides adequate coverage in relation to the requirements, environment, design choices, and associated uncertainties.

Table 3 Limitation in the State of the Art for V&V

ID#	Limitation in the State of the Art	Remark
1	Traditional verification is limited to finding defects, but it cannot assure the absence of defects, especially those rooted in systemic causes such as engineering mistakes.	Example limitation: Shortcomings in specifications, esp. for quality of service and derived requirements
2	Independent V&V personnel might not have an adequate understanding of the application domain.	The IV&V team needs domain expertise because the team must have equivalent or better expertise than the developers of the system (see Appendix B to 10 CFR Part 50). However, fulfillment of that requirement is not easy.
3	Methods for combining or integrating different types of evidence mentioned above and determining adequacy or sufficiency	Further discussed in Section 4 "Safety Demonstration"
4	Methods for modeling the operating context or environment that could expose sensitivity to unfortunate combinations of defects with especially severe consequences	
5	Validation of requirements	See Appendix A.3, "Requirements"
6	In architectures, identifying the "right" level of separation of concerns or isolation to avoid unexpected coupling, dependency, or side effects. (Also see discussion in	See A.4 Baseline Architecture," condition 6

[28] Further recommendations are given in Section 7, "Miscellaneous Research Recommendations—V&V."

ID#	Limitation in the State of the Art	Remark
	Appendix C)	
7	In architectures, specifying the "quality of service"[a] properties or requirements and identifying dependencies	
8	Systematizing the identification of dependencies	
9	Methods and tools for compositional reasoning about interacting components	
[a] Commonly known as nonfunctional requirements		

4. Safety Demonstration

Baseline for discussion of uncertainties: This discussion of uncertainties is predicated on the assumption that DI&C systems and their elements satisfy the conditions and criteria outlined in Appendices A.1–A.6. It is not implied that the assumption holds for current DI&C systems, their elements, or their development environments.

As Figure 3 shows, when V&V activities indicate various uncertainties about whether the safety system or a component meets its requirements, it is difficult to ensure that the system safety goals are met. Referring to the baseline given in A.6 Safety Demonstration Framework," and depicted in Figure 6 Evidence-argument-claim structure," the assertion claim would be substantiated through a logical (argument-based or reasoning-based) organization and integration of the evidence. It is not enough to show only a clause-by-clause compliance with the applicable requirements, guidelines, and standards. The evidence-argument-claim chain or structure demonstrates that the effect of uncertainties and known limitations is reduced to a level and manner that does not compromise the safety goal. In particular, the safety demonstration would include the following:

- diverse, complementary evidence, as shown in Figure 3
- explicit evaluation of the sufficiency of evidence and argument or reasoning to expose weaknesses, fallacies, and limitations
- explicit reasoning about the uncertainties in the evidence[29] and how these have been managed and mitigated
- evidence of the level of effort and rigor in analysis and proof, and that it is commensurate with the strength of the claim made
- explicit identification of system aspects, features, characteristics, or other items or of process activities or competencies upon which the safety argument depends, in order to identify whether a change impacts the argument
- modular structure (see Appendix A.4, especially Sections 4–6) with modular evidence[30]

[29] Objective evaluation of safety evidence and arguments is not always feasible…. Thus, evaluation includes subjectivity, as in law and other disciplines. Good scientific evidence or good mathematical evidence is better but it is a rarity. And bad "scientific" evidence and bad "mathematical" evidence is worse than no evidence because it produces unjustified confidence [MH].

The state of the art in the application of a logical claim-argument-evidence structure (see Figure 6) to DI&C safety systems has significant limitations. There is too little theory and too little empirical knowledge of how things work in practice. Therefore, highly skilled judgment is recommended in conjunction with multidisciplinary, multidimensional expertise and with diversity of perspectives and thought processes. The application of judgment could be improved through additional knowledge, such as the following:

- understanding, principles,[31] and techniques drawn from other fields, e.g., philosophy, law, linguistics[32], for evaluating arguments and reviewing safety demonstrations for the quality of arguments and evidence
- understanding of the limitations in evidence and how to combine different types of evidence,[33,34] such as testing, model-checking and analysis, including a theory of coverage
- understanding of where in a process uncertainties can arise (e.g., when creators of the architecture misunderstand the requirements)
- integrating the contribution of interdependent factors, such as the complexity⇔competence[35] nexus

Further recommendations appear in Section 7, "Miscellaneous Research Recommendations."

5. Uncertainties Associated with Tool-Automated Processes

Baseline for discussion of uncertainties: This discussion of uncertainties is predicated on the assumption that DI&C systems and their elements satisfy the conditions and criteria outlined in Appendices A.1–A.6. It is not implied that the assumption holds for current DI&C systems, their elements, or their development environments.

There is a trend to automate[36] labor-intensive tasks, under the premise that automation will preclude the occurrence of mistakes made by programmers or testers.[37] Although this premise applies well to automating tedious tasks at which humans are typically not very good,

---

[30] Modular evidence is evidence of verification about each module or element, which can be reused wherever the module is reused and which can be composed in a claim-argument-evidence hierarchy corresponding to the system architecture.

[31] Strive for a scientific foundation, e.g., devise a calculus for reasoning about uncertainties, degrees of validity, and degrees of confidence [MB].

[32] in combination with those from basic "hard" sciences, e.g. logic, inference, completeness of the proof

[33] …be aware that there is always some uncertainty left, and so…the interesting question is: What is a good approach to deal with all of the things we have achieved to come as close as possible to 100 percent, and to be aware of the uncertainties and make sure that these uncertainties do no harm [MB]

[34] One of the issues is how we can put our hands around what is needed and how to combine evidence. This is important when we ask the question "If we are going to accept something different, what is the scope and span of evidence that is needed?" If we replace one thing with something else, will we be missing anything? Mapping the areas of knowledge needed to make the safety decision is important [SA].

[35] There is limited understanding of the complexity⇔competence relationship. As the cognitive load increases and the simultaneous use of multiple cognitive faculties increases with complexity, it becomes more difficult to identify the commensurate performance indicators or to assess the competence of a performer.

[36] Qualified tools are invaluable if used correctly. Even some "trivial" tools have been instrumental in significant time savings and higher quality.

[37] Although automation can reduce or eliminate a class of defects, it also changes the space of possible new defects. This could result in systems exhibiting new failure modes that are outside the intuition and experience of the designers and regulators.

automating judgment-based tasks at which humans are typically good entails potential hazards about which there is little understanding or experience.[38]

Early experience with automation in various engineering domains has shown that the task performed by the tool is not fully automated but rather entails a new set of human competenciesfor human-tool interaction, often requiring cognitive capabilities that are not well understood or well documented. This introduces new sources of uncertainty. For example, the semantics and assumptions built into the tool or the limitations of its application are not explicit and not well understood. Use of a tool beyond its valid range has led to disastrous consequences.

Even when engineering tools such as code generators and verifiers are used in their valid range, there is uncertainty about the correctness of their use. Qualification standards and capabilities for such tools are not mature [PM].

Table 4 identifies some of the known limitations and challenges experienced in tool-automated development or verification of complex logic, esp. software.

Because many automated tools can be used across many products, if a tool has a hidden defect or limitation, an adverse effect can occur in different parts of a system and even in many different systems, multiplying the effect of the tool's defect or limitation.

Some sources of uncertainty are worth specific scrutiny when evaluating tools:

- code generation and testing from same model (see Table 8. criterion 1)
- intellectual grasp of tool operation and results (see Table 8, criteria 11,12, and 14)
- completeness of tool coverage
- applicability of tool to specific project and product
- pedigree and integrity of tools
- assumptions made by tool designers
- algorithms used by tools

As an approach to manage complexity, software is developed by using a sequence of abstractions from architectural specifications to design and then to implementation, and abstraction-to-concretion transformations are performed with the assistance of tools, intended to "hide the complexity." However, new uncertainties result from semantic inconsistencies across different levels of abstractions. Evaluation of a tool[39] is performed in the context of the whole process, including other information[40] on which the performance of the tool under evaluation depends[41] (also see Appendix A.5,

Table 8). One type of interdependency, as mentioned above, is language[42] semantics and interpretations by respective tools (see Appendix A.5,

---

[38] Potential fault modes are outside the intuition and experience of the designers and regulators.

[39] and the chain, as a set,

[40] For example, the 1994 Intel Floating-Point Divide (FDIV) defect was introduced by a simple bug in a script used to generate table entries for the quotient selection step of the FDIV algorithm.

[41] This recommendation is consistent with and an inherent part of the safety demonstration concept.

[42] Standardization in itself is not sufficient; the Unified Modeling Language is an example of such a weak standard.

Table 8, criteria 2–8). If semantic compatibility is claimed through the use of restricted subsets of the respective languages[43] [10, 11, 12, and 13], definitions of these restricted subsets would be carefully examined for semantic compatibility (see Appendix A.5,

Table 8, criteria 3–4). The more transformation stages in the process, the higher is the likelihood of semantic mismatches potentially leading to defects. If the relevant evidence is inadequate, the deficiency can be compensated for with more traditional evidence of correctness and rigor in testing and with an independent check of the output.[44]

Table 4 Limitations and Challenges in Tool-Automated Processes

ID#	Limitation or Challenge	Remark
1	Tool support and the validity of results from tool-automated processes is dependent on appropriate models and methods for requirements engineering, architecture design, coding or code generation, and the deployment and correct fit of the respective work products.	See Appendices A.1–A.6 Serious limitation: Shortage of skilled people
2	Verification of complex tools such as compilers and the integrated tool suite	Also see Table 8
3	Confidence in certitude of verification	To overcome limitations of any single verification method or tool, use as many (certified and uncertified) checking methods as available, in order to provide the greatest possible coverage across the gaps of individual tools. Diversity in the methods and tools can also help improve confidence.
4	Adapting traditional software processes to model-based development	
5	Understanding the effects of automation on the ability of humans to fully comprehend the state of a system or tool	
6	Determining appropriate mix of human and automation interaction to efficiently leverage respective strengths and compensate for individual weaknesses	
7	Automation can miss important aspects that have implicitly been performed by humans	
8	Ability to put enough practical detail in a model to be able to drive the development process realistically enough not to have to tweak results	

[43] The semantics of all of the languages used in the various transformation stages are not likely to match.

[44] Manual verification of the work product of such tools is difficult and introduces additional uncertainties.

6. Unknown Effects of Change

This discussion of uncertainties is predicated on the assumption that DI&C systems and their elements satisfy the conditions and criteria outlined in Appendices A.1–A.6. It is not implied that the assumption holds for current DI&C systems or their elements or development environments.

Although the NRC guidance includes change impact analysis, the adequacy of the analysis performed is unclear.[45] A major reason is that the degree of significance of a change is unclear. If a change is fine-grained or if it occurs in some indirect or supporting activity or item, it may be treated incorrectly as "insignificant." In such cases, the dependency of the safety goals on a particular item or activity may be obscure for the reasons identified in Table 5. Changes can expose existing unknown defects that did not have an effect before the change. Finding defects after change is very difficult without extensive re-verification. In essence, the regression analysis must be as convincing as the original analysis.

Table 5 Why Dependencies Are Obscure

ID#	Reason	Remark
1	Lack of a logical structure in the safety demonstration	See A.6 Safety Demonstration Framework"
2	Lack of awareness, explicit identification, representation, and documentation of all the dependencies, including: 1. Requirements 2. System architecture 3. Processes 4. Organizational architecture 5. Such other related factors	
3	Inadequate configuration management and inadequate granularity of configuration management	Assess adequacy of standards for change control and configuration control.
4	Lack of historical record of system development decisions—not known or not understood when making change	Developers make assumptions without explicit awareness. A hidden assumption may not affect the original system but may affect the change.
5	Lack of change in system corresponding to change in its environment	

Change may occur in various ways, including the following:

- corrective—"bug fix"
- perfective—do the same only better
- new functionality—change in requirements
- adaptive—responding to a change in the environment that invalidates earlier assumptions

[45] Programmatic requirements for change control and configuration control are not adequate to support change impact analysis.

- changes in personnel over the lifetime of a system (loss of knowledge)

For the analysis of degree of impact, the drivers or sources of change may be characterized as follows:

- known, planned, controlled:
- changes made during initial development
- anticipated changes in requirements
- anticipated changes in design
- unanticipated or uncontrolled (not expected at an earlier time)—more difficult to analyze

Table 6 outlines recommendations to improve the evaluation of change impact analysis. While these recommendations would help reduce uncertainties in general, numbers 1–2, 8, and 10–12 are more specific to the impact of change.

Table 6 Recommendations for Evaluating Change Impact Analysis

ID#	Recommendation[a]	Remarks
1	Items under configuration control and change control should include the safety demonstration and all items on which the safety demonstration is dependent (i.e., the arguments, evidence, and items on which these are dependent). The latter may include requirements, system architecture, processes, the tools, competencies, and data on which the processes depend, supporting tools, operating conditions, and maintenance.	
2	Ensure that the safety demonstration makes explicit what aspects, features, characteristics, items, or other factors the safety argument depends on so that one can tell if the change affects the safety argument.	
3	Impact of change should include analysis against the original system, not just the most recent version. This includes updates of all related documents and artifacts to ensure consistent and valid configurations.	
4	Test space is large—seek preventative approaches, instead of excessive reliance on testing.	See Appendices A.1–A.6
5	Ensure that architecture provably prevents or limits the propagation and effects of change.[b]	See "A.4 Baseline Architecture Specification Principles," esp. criteria 4–6[c]
6	Evaluate readability of documentation and code (e.g., by checking code against coding standards or procedures and reading documentation for comprehensibility and consistency with code).	Poor readability leads to mistakes and higher maintenance costs.
7	Check that rationale for design decisions (e.g., architectural) is documented for comprehension by an unfamiliar third party.	
8	Traceability documentation should be maintained to assess impact of changes (relate to and expand to include general dependencies).	
9	Check that information[d] is maintained in one place only (and referenced elsewhere—not copied and pasted).[e]	
10	Rate or calibrate the performance of an organization in following sound practices (e.g., in change and configuration control and management).	

ID#	Recommendation[a]	Remarks
	Adjust review depth and probes accordingly.	
11	Make sure that the people making changes are as qualified as the original developers and are adequately familiar with the system.	
12	Possible defensive measure: validation by parallel operation (new and old) for extended periods.	

[a] Even following all these recommendations cannot eliminate the possibility of defect introduced through change.

[b] Allows regression verification to be limited to that part of the system that is affected by the change.

[c] Information hiding allows formal analysis of the effect of change, thereby ensuring that change will not damage the system in unknown ways. There is extensive evidence on the success of information hiding. There is extensive evidence of failure through not using something like information hiding.

[d] Documentation, code, design.

[e] Duplication exposes the documentation to inconsistencies, which exposes the change process to mistakes, potentially leading to defects in the system.

7. Miscellaneous Research Recommendations

Knowledge gaps identified in the topics discussed above are candidates for research. Following are some additional recommendations, identified by the experts:

V&V: Research is recommended to overcome the limitations in V&V identified in Table 3. The NPP application domain can also benefit from tracking the state of the art and developments in other safety-critical application domains, such as the following:

- use of models and model-checking in the medical devices application domain
- state of practice in verification and certification of real-time operating systems[46] for safety-critical applications (e.g., flight control systems)

Safety demonstration: Evaluation of safety demonstrations (for the quality of arguments and evidence) could draw upon principles and techniques from other fields (e.g., philosophy, law, linguistics)[47], striving to transform this knowledge into a scientific foundation for reasoning about uncertainties, degrees of validity, and degrees of confidence. To ensure that the uncertainties do not lead to harm, experts recommend research for understanding the limitations in evidence and how to combine different types of evidence, such as shown in Table 2, including a theory of coverage. Mapping the areas of knowledge used to make the safety decision is important. It is recommended that research to overcome limitations in safety demonstration include learning more about the specific limitations or conditions experienced in licensing reviews, including the review of safety cases and assurance cases, where available, and the review of operational experience.[48]

Architecture: Better ways of defining an interface are recommended to integrate properties other than the function to be performed (e.g., timing constraints and criticality). Otherwise, it is difficult to analyze the impact of change when such properties are affected.

---

[46] A "homemade" real-time operating system should be checked for this level of verification and operational experience, at least.

[47] to integrate different types of evidence, such as shown in Table 2 and to complement knowledge from basic disciplines used in §3-4 of Table 2.

[48] The Haddon-Cave Report [24] attacks models of building an argument about why a system is safe. These documents have become too design time focused but they are not reviewed down the road for failures [CJ].

Tool automation and its correct use: Research is recommended to overcome the limitations and challenges identified in Table 4. There is also a need to develop more rigorous tool qualification standards.[49] In this context, the merits[50] of "self-certification" of tools by their suppliers could be examined.

8. Next Steps

While this RIL alerts the licensing offices about major sources of uncertainty in the safety assessment of DI&C systems, maturation of this information into published review guidance will be an iterative, evolutionary process. Following are some next steps recommended by the expert focus group:

8.1 Public Dissemination

For inviting public review, comment, discussion, and resolution, the experts suggested a workshop in conjunction with a mainstream software engineering conference. This would enable cross-domain participation and discussion, leading to cross-fertilization of ideas to the NPP domain from other safety-critical application domains.

8.2 Follow-On Involvement of Expert Focus Groups

Recognizing that the scope of these problems is large and their resolution is a long-term effort, the experts recommended follow-on engagement of similar expert focus groups with more domain-specific information and specific topic areas.

The NRC will integrate this suggestion into several research activities as part of the NRC's 2010–2014 DI&C research plan. Some examples are outlined below.

Findings reported in this RIL that are relevant to the NRC's DI&C research plan [2] Section 3.1.2, "Safety Assessment of Tool Automated Processes," will be incorporated into the execution plan. The project will seek guidance from the world's leading researchers in the subject.

NRC/RES/DE Digital Instrumentation and Control Branch (DICB) will seek an expert focus group to guide research activities identified in the NRC's DI&C research plan [2] Section 3.1.5, "Analytical Assessment of DI&C Systems," as described below:

- Section 3.1.5, deliverable 3, requires analysis of each NRC-approved safety system platform (i.e., Common Q: Advant Fieldbus 100 and High Speed Links, Teleperm XS: PROFIBUS & Ethernet, Tricon: Tricon System Access Application and Peer-to-Peer) and identification of credible[51] fault and failure modes. The analysis includes models to "roll up" or "up-integrate" effects of malfunctions in networked elements. The scope includes an analysis for systems with tightly coupled integration of traditionally decoupled or loosely coupled functions, applications (e.g., reactor trip system, engineered safety features actuation system), signals, and infrastructural services, as exemplified in new licensing applications. Engagement of experts in this activity will help focus their knowledge to the issues of the NPP domain.
- Section 3.1.5, deliverable 2, requires characterization of different kinds of DI&C systems and their relationship to their environments, progressing in three stages:

[49] Learn from best practices and evolving standards in other application domains (e.g., commercial aviation).

[50] The issue is reduced degree of independence versus access to knowledge and experience.

[51] It is not easy to identify all credible fault and failure modes because "all" is not known and may not be knowable with the existing platforms.

1. Existing inventory in NPPs and the NRC-approved platforms mentioned above
2. Emerging systems in NPPs
3. Knowledge gained from safety-critical systems outside NPPs.[52]

These activities will also support the creation of a challenge problem model (another suggestion emerging from the expert clinic). This model will be representative of the system configurations, platforms, and applications seen or expected in the NPP domain.[53] In order to focus the experts' knowledge on problems being experienced or foreseen in the NRC, these activities will require the participation of experienced NRC licensing reviewers.

[52] Engagement of experts will help address classes of issues broader than the specific platforms mentioned above.

[53] It has been done successfully several times. It helped to develop the field and also got a lot of feedback [MB].

9. Glossary

The scope of this glossary[54] is limited to this RIL and the related RILs in this series [4 and 5].

Atomic element; Non-atomic element (of an architecture)

An element or unit (such as a hardware or software component) that is indivisible (i.e., for which no further subdivision is described in the architecture of a computer system, such as a DI&C system, or its software).

Related definition: A non-atomic element is one that consists of other elements (e.g., a software subsystem composed of a number of software components), as described in the architecture.

Complexity

(A) (software) The degree to which a system or component has functionality, design or implementation that is difficult to understand and verify. (definition (1)(A) in [15]).

(B) (software) Pertaining to any of a set of structure-based metrics that measure the attribute in Definition 1A in Ref. [15]. (definition (1)(B) in [15]).

Note 1: There is no universally accepted definition of the term "complexity."[55] The notes below give some other definitions of complexity to illustrate the diversity of perspectives.
Note 2: Conversely (changing negative expression to positive) Simplicity: The degree to which a system or component functionality, design or implementation can be understood and verified.
Note 3: The number of linearly independent paths (one plus the number of conditions) through the source code of a computer program is an indicator of control flow complexity, known as McCabe's cyclomatic complexity [16].
Note 4: In nontechnical language, we can define the effective complexity of an entity as the length of a highly compressed description of its regularities [17].
Note 5: An ill-defined term that means many things to many people [18].
Note 6: A system is classified as complex if its design is unsuitable for the application of exhaustive simulation and test, and therefore its behavior cannot be verified by exhaustive testing. Source: Defence Standard 00-54, Requirements for safety related electronic hardware in defence equipment, UK Ministry of Defence, 1999.

Constituent

"Serving to form, compose, or make up a unit or whole: COMPONENT <*constituent* parts>" (http://www.merriam-webster.com/dictionary/constituent?show=1&t=1283882195)

Diverse team

A team composed of individuals with complementary attributes needed to perform the assigned task (e.g., thought processes, communication styles, and competence, including education, training, and experience in different domains and disciplines).

Element

The smallest component identified in the architecture of a computer system or its composition into a larger component, subsystem, or a system in a system of systems, which affects one, or possibly more than one, elementary function. An element may be hardware, software, firmware, or a combination thereof.

---

[54] Because definitions come a variety of sources, there is an unintended side effect of inherent inconsistencies.

[55] Research is needed to clarify complexity within the context of system safety evaluation.

Error

The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition (definition 8A in [15]).

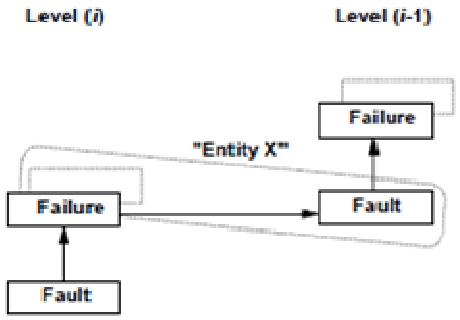
Failure

The termination of the ability of an item to perform a required function [19].

Note 1: After failure, the item has a fault [19].

Note 2: “Failure” is an event, as distinguished from “fault,” which is a state [19].

Note 3: This concept as defined does not apply to items consisting of software only [19].



From the point of view of IEC 61508 and ISO/IEC 2382-14

Figure 4 A fault must exist before a failure can occur

Figure 4 [20] illustrates the relationship between fault and failure across levels in the assembly hierarchy.

The following definitions from [15][56] represent the perspectives of different disciplines:

1A: The termination of the ability of an item to perform a required function.

1N: The termination of the ability of a functional unit to perform its required function.

1O: An event in which a system or system component does not perform a required function within specified limits. A failure may be produced when a fault is encountered.

9[57]**:** The termination of the ability of an item to perform its required function.

13[58]**:** The loss of the ability of a component, equipment, or system to perform a required function.

15[59]**:** The termination of the ability of an item to perform a required function.

Fault

An incorrect step, process, or data definition in a computer program (definition 7A in [15]). Note: This definition is used primarily by the fault tolerance discipline.

---

[56] The identifier, e.g. 1A, refers to the identifier in [15]

[57] Definition from standards concerning nuclear power generating stations

[58] Definition from standards concerning safety systems equipment in nuclear power generating stations

[59] Definition from standards concerning nuclear power generating systems

A defect in a hardware device or component; for example, a short circuit or broken wire (definition 9 in [15]).

Synonym: physical defect.

A defect or flaw in a hardware or software component (definition 13 in [15]).

Derived definition:

Latent fault: An existing fault that has not yet been recognized.

For software, this concept of fault corresponds to "defect."

Feasible

"Capable of being done with means at hand and circumstances as they are" (http://wordnetweb.princeton.edu/perl/webwn?s=feasible).

Other definitions also impose such constraints as "practicably," "reasonable amount of effort, cost, or other hardship" (http://www.fhwa.dot.gov/environment/sidewalks/appb.htm), or "cost-effectiveness" (http://uis.georgetown.edu/departments/eets/dw/GLOSSARY0816.html). Such constraints distinguish "feasibility" from "possibility."

Information hiding

The principle of segregation of design decisions in a computer program that is most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed. The protection involves providing a stable interface that protects the remainder of the program from the implementation (the details that are most likely to change).

Mistake

A human action that produces an unintended result (definition 1 in [15]: electronic computation).

Note: Common mistakes include incorrect programming, coding, and manual operation [15]. This definition suits mistakes concerning validation.

A human action that produces an incorrect result (definition 3 in [15]) (software).

Note: The fault tolerance discipline distinguishes between the human action (a mistake), its manifestation (a hardware or software fault), the result of the fault (a failure), and the amount by which the result is incorrect (the error) [15]. This definition suits mistakes concerning verification.

Mode confusion

A situation in which an engineered system can behave differently from its user's expectation, because of a misunderstanding or inadequate understanding of the system state.

Modularized

Having or made up of modules.

Note: Also see separation of concerns, information hiding, and orthogonality.

Module

A removal or changeable element of a system, with a well-defined interface such that certain properties of the system are preserved when it is replaced.

Note 1: This definition is generalized from IEEE Standard (Std) 100 [15], definition 8 (whose source is IEC 60050); this definition, "a plug-in unit," implies that "certain properties of the system are preserved when a module or its replacement is "plugged in" or integrated in the system."

Note 2: An example of the preserved system properties is the behavior of the remaining part of the system (i.e., it is implied that the functions of the module and the functions of the remaining part do not affect each other).
Note 3: It is implied that the element is identifiable and separable from and integratable with the remaining part of the system in some repeatable manner, following some well-defined procedure. For example, for software, IEEE Std 100 [15], definition 2A, mentions compiling and loading with the aid of an assembler, compiler, linkage editor, or executive routine. IEEE Std 100 [15], definition 2B, describes it as a "logically separable part of a program."
Note 4: IEEE Std 100 [15], definition 2B, states that the terms "module," "component," and "unit" are often used interchangeably or defined to be sub-elements of one another in different ways depending on the context. The relationship of these terms is not yet standardized. The RIL-1001 glossary distinguishes a module as a special kind of unit with well-defined interfaces preserving certain properties of the system.
Note 5: In the case of a hardware assembly or an assembly that may also include firmware or software, the interface also includes physical attributes and implies spatial boundaries to preserve the system architectural property of freedom from interference. For example, see IEEE Std 100 [15] definitions 9 and 10.

Noninterference

Absence of unwanted function or feature interaction and absence of cascading failures between two or more elements that could lead to the violation of a safety requirement (adapted from [21][60]).

Example 1: Element 1 is interference-free of element 2 if no failure of element 2 can cause element 1 to fail.
Example 2: Element 3 interferes with element 4 if there exists a failure of element 3 that causes element 4 to fail.

Orthogonality

A property of the variables on which a function depends, such that the variables are independent of each other (i.e., a change in one variable does not affect another).

Note 1: In computer science, orthogonality has been identified as an important property of the fundamental elements (building block functions and variables) of a programming language. When the language elements are orthogonal, the programs composed from these building blocks avoid unnecessary complexity and are easier to verify.
Note 2: When user-defined functions in a computer program are orthogonal compositions of orthogonal building blocks, these functions avoid unnecessary complexity and are easier to verify. Such programs lend themselves to use as modules in larger programs. Then orthogonality is a module property.

Note 3: When a software system is a composition of orthogonal modules, the software system avoids unnecessary complexity and is easier to verify. Then orthogonality is a system property.
Note 4: Orthogonality guarantees that modifying the technical effect produced by a module of a system neither creates nor propagates side effects to other modules of the system.
Note 5: The following are some example guidelines for practical application of orthogonality:
Minimize unnecessary coupling across modules (see http://www.eli.sdsu.edu/courses/spring01/cs635/notes/module/module.html#Heading2).
Maximize cohesion within a module (see http://www.cs.unc.edu/~stotts/145/cohesion.html).

[60] This uses the term "freedom from interference."

Separation of concerns

"The process of separating the development of a computer program into distinct features, aspects, and views that overlap in functionality as little as possible. A concern is any piece of interest or focus in a program. Typically, concerns are synonymous with features or behaviors." (adapted from en.wikipedia.org/wiki/Separation_of_concerns)

Synchronous architecture

An architecture that preserves synchronization between related items of information.

Note 1: Synchronization means maintaining the timing relationships (e.g., period and phase), usually with the aid of a clock.

Note 2: Asynchronous means not maintaining such timing relationships explicitly.

Note 3: Some experts claim[61] that synchronization makes it easier to verify satisfaction of timing constraints as found in a hard real-time system.

Note 4: Systems and development processes architected with this principle avoid unnecessary complexity. Systems are easier to verify. Also see "information hiding" and "orthogonality."

Systemic

Embedded within and spread throughout and affecting a group, system, or body. Also see "systemic cause" in [14].

Systematic failure

Failure, related in a deterministic way to a certain cause, that can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors [19].

Note 1: Corrective maintenance without modification will usually not eliminate the failure cause.

Note 2: A systematic failure can be induced by simulating the failure cause.

Note 3: In IEC 61508-4, CDV 3.6.6, examples of causes of systematic failures include human error in the following:

the safety requirements specification

the design, manufacture, installation, and operation of the hardware

the design, and implementation of the software

Also see "systemic cause" in [14].

[61] A counter-example from the NPP safety systems domain: At architectural level, the different computers which together constitute a protection system (acquisition units, logic units, voting units) usually work asynchronously to prevent a failure in a receiving (consumer) unit from propagating into an emitting (producer) unit. In this case, the asynchronous architecture is much simpler and easy to verify than a synchronous one.

10. Abbreviations and Acronyms

ACRS	Advisory Committee on Reactor Safeguards
DE	Division of Engineering (NRC/RES)
DI&C	digital instrumentation and control
DICB	Digital Instrumentation and Control Branch (NRC/RES/DE)
FMEA	failure modes and effects analysis
FPGA	field-programmable gate array
FTA	failure tree analysis
HW	hardware
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
IV&V	independent verification and validation
NPP	nuclear power plant
NRC	U.S. Nuclear Regulatory Commission
OpE	operational experience
PNNL	Pacific Northwest National Laboratories
RES	Office of Nuclear Regulatory Research (NRC)
RIL	research information letter
SRM	staff requirements memorandum
STD	standard
SW	software
V&V	verification and validation

Table 7 Abbreviations Identifying Sources

Abbreviation	Source	Abbreviation	Source
AW	Alan Wassyng	JM	John McDermid
CJ	Chris Johnson	MB	Manfred Broy
CW	Charles Wallace	MH	Michael Holloway
DC	Darren Cofer	PJ	Paul Jones
DD	Dan Dvorak	PM	Paul Miner
DJ	Dana Johnson	RK	Roger Kieckhafer
GH	Gerard Holzmann	SA	Steve Arndt
JC	James Christensen	SP	Stephen Prusha
JH	Jorgen Hansson	VV	Valerie Vyatkin
JK	John Knight	WL	William Locke

11. References

1. Computer Science and Telecommunications Board, *Software for Dependable Systems: Sufficient Evidence?*, 2007. URL: http://www.nap.edu/catalog.php?record_id=11923
2. U.S. Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, Division of Engineering, Digital Instrumentation and Controls Branch, "NRC Digital System Research Plan FY 2010–FY 2014," February 2010. URL: http://adamspublic.nrc.gov/fnopenclient/FnDocContent.aspx?Library=PU_ADAMS%5ePBNTAD01&Id=100570112&ObjType=2&Op=Open
3. U.S. Nuclear Regulatory Commission, Staff Requirements Memorandum M080605B. URL: http://adamswebsearch.nrc.gov/scripts/rwisapi.dll/@pip1.env?CQ_SESSION_KEY=DKVCWZBKJVVV&CQ_QUERY_HANDLE=125470&CQQNUM=1&CQ_DOCUMENT=YES&CQ_SAVE[ResultsReturnPage]=results_list.html&CQ_CUR_DOCUMENT=1
4. U.S. Nuclear Regulatory Commission, "Identification of Failure Modes in Digital Safety Systems and Analysis for Systemic Causes—Expert Clinic Findings, Part 2," Research Information Letter RIL-1002.
5. U.S. Nuclear Regulatory Commission, "Feasibility of Applying Failure Mode Analysis to Quantification of Risk Associated with Digital Safety Systems—Expert Clinic Findings, Part 3," Research Information Letter RIL-1003.
6. Transcript of 551st Advisory Committee on Reactor Safeguards meeting, April 10–12, 2008, page 23, lines 8–20, page 24, lines 5–10, page 110, lines 21–25, page 111, lines 1-24. URL: http://www.nrc.gov/reading-rm/doc-collections/acrs/tr/fullcommittee/2008/acrs041008-551.pdf
7. Committee on Application of Digital Instrumentation and Control Systems to Nuclear Power Plant Operations and Safety and National Research Council, *Digital Instrumentation & Control Systems in Nuclear Power Plants: Safety and Reliability Issues*, The National Academies Press, 1997. URL: http://www.nap.edu/catalog.php?record_id=5432
8. Chu, T.L., et al., "Review of Quantitative Software Reliability Methods," Brookhaven National Laboratory, Technical Report, BNL-94047-2010, September 2010.
9. Parnas, David, et al., "Assessment of Safety-Critical Software in Nuclear Power Plants," 1994.
10. "MISRA AC AGC: Guidelines for the Application of MISRA-C:2004 in the Context of Automatic Code Generation," November 2007. URL: http://www.misra-c2.com/Publications/tabid/57/Default.aspx#label-ac-agc
11. "MISRA AC GMG: Generic Modeling Design and Style Guidelines." URL: http://www.misra-c2.com/Publications/tabid/57/Default.aspx#label-ac-gmg
12. "MISRA AC SLSF: Modeling Design and Style Guidelines for the Application of MathWorks Simulink and Stateflow," May 2009. URL: http://www.misra-c2.com/Publications/tabid/57/Default.aspx#label-ac-slsf
13. "MISRA AC TL: Modeling Style Guidelines for the Application of TargetLink in the Context of Automatic Code Generation," November 2007. URL: http://www.misra-c2.com/Publications/tabid/57/Default.aspx#label-ac-tl

14. Eckert, C., “Identification and Elimination of Systemic Problems,” *Proceedings of the Society of Maintenance and Reliability Professionals Annual Symposium*, St. Louis, MO, October 20–22, 2009.
15. IEEE Standard 100-2000, “The Authoritative Dictionary of IEEE Standards,” 7th edition, 2000.
16. McCabe, T.J., “A Complexity Measure,” *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, December 1976.
17. Gell-Mann, M., and S. Lloyd, “Effective Complexity,” Santa Fe Institute, June 26, 2003.
18. Flake, G.W., “The Computational Beauty of Nature,” MIT Press.
19. International Electrotechnical Commission Vocabulary Chapter 191: Dependability and Quality of Service, first edition, 1990-12.
20. IEC 61508-4, “Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems—Part 4: Definitions and Abbreviations—Edition 2.0,” 2010.
21. ISO/DIS 26262, “Road Vehicles—Functional Safety,” 2009.
22. ISO/IEC 12207:2008(E) (IEEE Std 12207-2008), “Systems and Software Engineering—Software Lifecycle Processes.”
23. IEEE STD 1012-2004, “IEEE Standard for Software Verification and Validation,” June 8, 2005.
24. ISO/IEC 15504 (all parts), “Information Technology—Process Assessment.”
25. Haddon-Cave, C., QC, “The Nimrod Review.” URL: http://ethics.tamu.edu/guest/XV230/1025%5B1%5D.pdf
26. Parnas, D., “Inspection of Safety-Critical Software Using Program-Function Tables,” IFIP 94.
27. United Kingdom Ministry of Defence, Standard 00-56, “Safety Management Requirements for Defence Systems,” parts 1–4. URL: http://www.dstan.mod.uk/standards/defstans/00/056/01000400.pdf
28. Advisory Committee on Reactor Safeguards Digital Instrumentation and Controls Systems Subcommittee, Official Transcript of Proceedings, page 132, line 15, through page 147, line 8, August 20, 2009.
29. Broy, M., “Compositional Refinement of Interactive Systems. Journal of the ACM, Volume 44, No. 6 (Nov 1997), 850-891”

Appendix A

A.1 Baseline Process

The discussion of uncertainties is based on satisfaction of the following process conditions (labeled as the baseline) in the development,[1] third-party verification, assessment, and audit[2] activities of a digital instrumentation and control (DI&C) system for safety functions. Failure to satisfy these conditions gives rise to uncertainties[3] beyond those discussed in this report.

1. The organization has defined an executable process to a level of detail such that it specifies the competence, tools, information, and other resources required to execute that work element correctly[4] and to integrate the results of such work elements correctly.
   1.1. In recognition that each process variable in each work element may contribute to some defect, the organization has selected and maintained methods, tools, and competence levels to minimize the effects of these known contributing factors.
   1.2. The cognitive load (or intellectual complexity) imposed by a specified work element, including an integration activity, is assured to be well within the capabilities of personnel available to perform that activity. [5]

   Note: Because the definability of such a deterministic process also depends on the architecture and complexity of the product (DI&C system), also see the conditions identified in Appendix A.4.6

2. The executable processes follow the NRC regulatory and guidance framework (see Figure 5[7]), which includes references to standards such as Institute of Electrical and Electronics Engineers (IEEE) Standards 1012 [23],[8] 1028, 1044, 603, and 7-4.3.2 and International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) Standards 12207 [22] and 60880.

3. The process is assessed and certified independently [VV]; resources are available to assess the process.

---

[1] Examples of activities included requirements analysis, architecture, design, implementation, and testing.

[2] The NASA Software Architecture Review Board approach is that review of a project should tell you about the quality attribute of a system….demonstrate how well…achieving that requirement (Ex: safety). A project should…keep appropriate records to provide evidence to proper attention and show ability of designers to achieve certain qualities [DD].

[3] Even in a well-defined process, the uncertainty space of the process variables is very large. For example, the quality produced by different people of the same level of nominal qualification can vary over an order of magnitude.

[4] The DI&C system and the processes for its development, verification, assessment, and audit are designed with the awareness that a human activity may incur a mistake.

[5] This may require certification of personnel through a standardized process.

[6] Considering the relative simplicity of the required safety functions, if the identified conditions are satisfied, the requisite architectures are technically feasible and the overall complexity is decomposable into manageable levels.

[7] With reference to Electric Power Research Institute TR-102348 and such other industry documents, NRC endorsement may be limited and other NRC guidance may govern—it is not implied that the NRC accepts industry standards verbatim.

[8] Appendix F.1 to IEEE STD 1012-2004, "IEEE Standard for Software Verification and Validation," dated June 8, 2005 [23] shows the relationship of verification and validation (V&V) to other activities affecting the safety properties of the product. Examples of relevant references include the lifecycle process reference model [22], including the relationship to system lifecycle processes per Annex C to ISO/EIC Standard 12207-2008, "Systems and Software Engineering—Software Lifecycle Processes" [22], and the process assessment model [24].

4. The process in execution is audited independently.
   4.1. Resources are available to audit the process.
   4.2. Process audits are performed in sufficient[9] degree of completeness.[10]
5. For each contributing factor[11], performance criteria are established such that each criterion can be evaluated comparatively[12] on a defined scale to measure relative performance.
   5.1. Norms are established for achieving correct execution.
   5.2. Observable values of performance are established in proficiency- deficiency rank order.
   5.3. A threshold is established for the unacceptable[13] performance level[14] (§5.2).

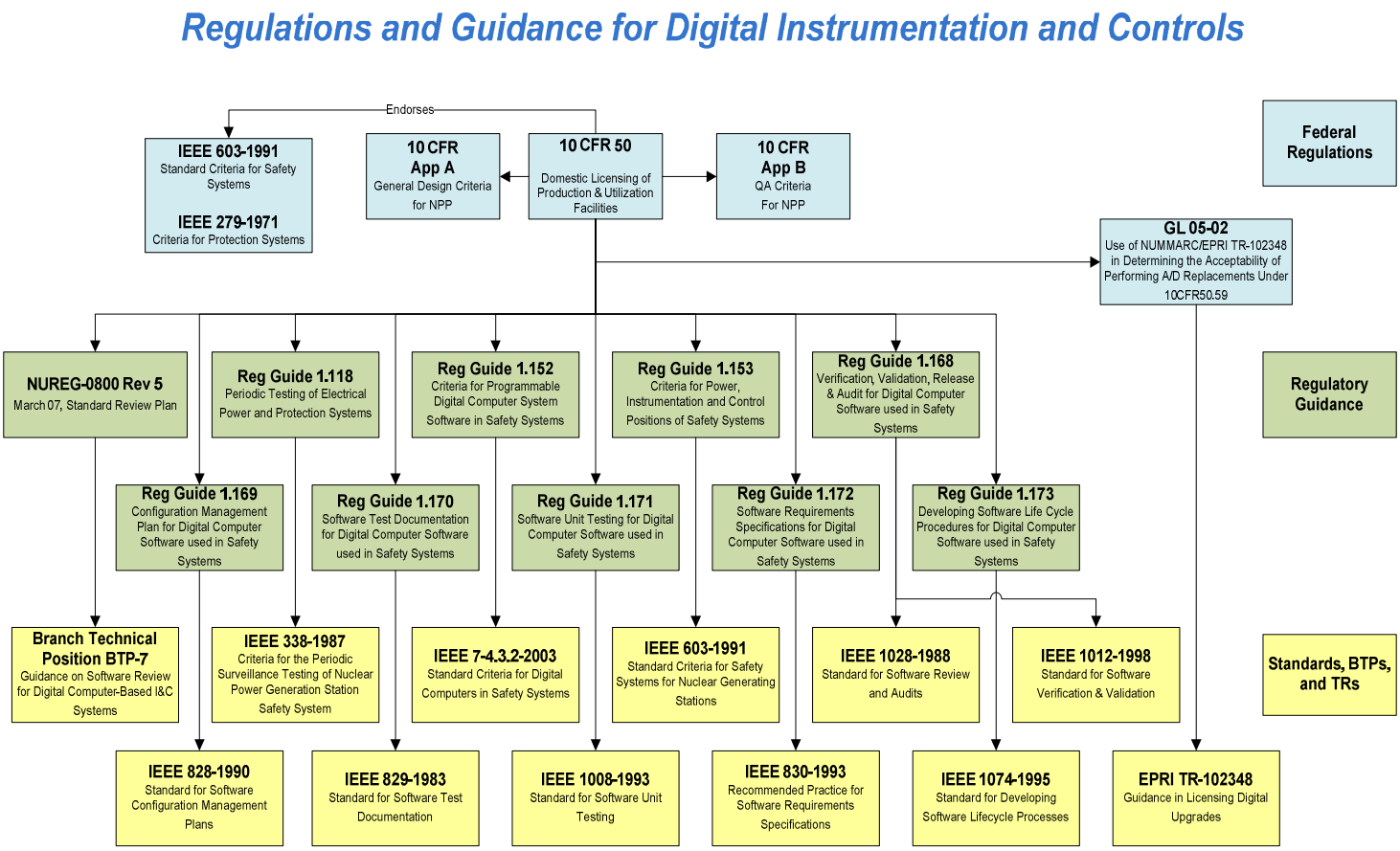


Figure 5: NRC's regulatory and guidance framework - overview

[9] e.g., based on statistical processes for selecting audit samples, assessing acceptability of audit samples, and evaluating audit results solely based on the audit process

[10] When the performance level is low or a good track record does not exist, the audit may take enormous effort, interfering with the audited activity.

[11] e.g., competence to perform an activity

[12] i.e., with ability to rank order

[13] An item dependent on this factor cannot be credited for contribution to assurance.

[14] It could be mapped into zero in a numeric quantification scheme of evaluation.

A.2 Baseline Verification Process

For the baseline postulated in the section, "Baseline for Discussion of Uncertainties," the corresponding V&V processes are identified in IEEE Standard 1012 [23] for integrity level 4, including relevant optional[15] activities and reports. It includes activities for management of V&V processes, acquisition, development, operation, maintenance, retirement and disposal, reporting, and documentation. IEEE Standard 1012 specifies verification and correction activities for each phase[16] in the development cycle. IEEE Standard 1012 specifies process activities for anomalies: recording, reporting, evaluation, impact analysis, root cause analysis, resolution, corrective actions, and analysis of the effect of changes to resolve an anomaly. Corrective action may require process improvement and regression implementation of process improvements on the system development effort and products.

A.3 Baseline Criteria for Specifying Requirements

The discussion of uncertainties in this report is based on the satisfaction of the following conditions (treated as the baseline) in the development, verification, and validation of requirements[17] that a DI&C system and its software must satisfy to support nuclear power plant (NPP) safety functions.[18] Failure to satisfy these conditions gives rise to uncertainties beyond those discussed in this report.

One of the weakest links in the overall process is establishing valid requirements to drive the design of the DI&C system. It is often thought to be the most common root cause of system failure [GH, JM, DC, PM, JK]. Failures traceable to shortcomings in requirements cannot be caught through such verification activities as simulation and testing. Early formalization (in the requirements phase) can help to discover and remove impreciseness, incompleteness, and inconsistency in requirements. However, formal methods do not help to discover or elicit missing requirements or intent or to understand intent [PM, GH].

The baseline conditions are the following:

1. The team engaged in these activities is an assemblage of high competence in multiple disciplines, capable of creatively eliciting and synthesizing information from diverse sources, including implicit, experiential knowledge about the environment [GH].
2. A diverse team reviews the requirements and their validation [GH]; it includes experts from other domains [PM]. The review team has expertise in requirements engineering, esp. in discovering the following types of mistakes or shortcomings:
   2.1. Misunderstanding the environment [DC, GH]
   2.2. Incompleteness [VV]
   2.3. Inconsistency [VV]
   2.4. Ambiguity in the natural-language textual description [DC]

---

[15] Optional activities and reports are tailored to the application (e.g., see Table 3 and Annex G to [23].

[16] The phases are concept, requirements, design, implementation, integration, change, and modification.

[17] Because most issues with inadequate software requirements can be traced to inadequate validation in the system engineering lifecycle, the treatment here is integrated. The scope also includes quality of service requirements, derived requirements and design constraints.

[18] The scope includes all derived and supporting requirements (i.e., any requirement on which the correct, timely performance of the safety function depends).

2.5. Formalization (from intent or natural-language text) [DC]

2.6. Input constraints misunderstood, improperly captured [DC]

2.7. Input constraints improperly formalized [DC]

2.8. Inadequate input validation

2.9. Inadequate or improper abstraction[19] to capture classes of issues [MH]

2.10. Adverse effects of the complexity of the functional requirement set[20] [WL, JM]

2.11. Also, see under item 5.

3. Assumptions about the environment are documented explicitly, including how and when these assumptions will be validated or the consequences if the assumption turns out to be false [JM].

4. Assumptions about the downstream design are documented explicitly, including how and when these assumptions will be validated [JM] (e.g., through explicit derived requirements or constraints on the architecture, design, and implementation) and the associated methods and tools.
   Example: Requirements imposed by the application software on system platform services, including hardware and software resources to support the workload, timing constraints to be satisfied, compatibility, and so forth, especially across maintenance updates [VV].

5. Requirements include measures to mitigate the consequences of assumptions that fail to hold [GH].

6. Results of the system safety engineering process (e.g., through system hazard analysis [25][21] are explicit, especially the resulting system safety requirements and their allocation to software requirements. This process is iterated at every phase[22] in the lifecycle.[23] Example areas of inquiry[24] include the following:

   6.1. Propagation of the effect of a fault (inability to contain it)

   6.2. Disallowing unwanted or unintended functions [MH]

   6.3. Effects of sudden hardware failure,[25] especially semiconductors [JC]

   6.4. Effects of faulty inputs

   6.5. In general, identifying sources of uncertainty, their effects, and their mitigation [MH]

   6.6. Provision of resiliency and robustness, assuming the presence of faults[26] [SP]

   6.7. Assuring that no unnecessary requirements exist.

---

[19] generalization

[20] The bigger your hazard analysis, the less useful it is [CJ].

[21] Consider the issues raised in [25].

[22] Concept→requirements model→system architecture→design→implementation.

[23] Safety requirements are discovered at every stage.

[24] as applicable in a particular phase of the development lifecycle

[25] We [should look] at systems engineering from a holistic viewpoint. Current practice divides systems engineers into one group and then we have software engineers here and we have hardware engineers there and very often the failures occur due to the gaps in between; usually, there is no one spanning all three (system; hardware; software) throughout the project. There is a need for crosscutting analytical approaches [JH].

[26] Identifying safety requirements requires first-class engineers. However, there is no way to know what all of the hazards are [JK].

7. Requirements include failure or fault detection and predefined action (for fail-safe systems) or containment[27] measures[28] [GH],[29] where appropriate[30], including the ability to locate and isolate the source of the fault (e.g., a hardware or software component) [PM].
8. Requirements are documented in a manner (e.g., language, structure) that is unambiguously comprehensible [AW, JK,26,7] to the community of their users (for example, reviewers, architects, designers, and implementers) (i.e., the people and the tools they use) [VV]. Example: Modeling
9. Relationships among requirements are unambiguously comprehensible to the community of users [VV].
10. Constraints associated with functional requirements (such as timing constraints [JH, AW], sometimes known as para-functional requirements, should be unambiguously comprehensible to the community of users [VV].
11. Satisfaction of requirements and constraints should be verifiable.
12. Interrelated requirements should be bi-directionally traceable.
13. Quality-of-service requirements[31] affecting safety should also be unambiguous and verifiable, referring to such properties as the following:
    13.1. Verifiability
    13.2. Analyzability
    13.3. Composability.
    13.4. Maintainability (design for change),
    13.5. Reusability[32]
    13.6. Reconfigurability

To the extent that any of these conditions is not satisfied, independent experts with commensurately higher level of competence[33] arerecommendedto evaluate the effect on system-failure likelihood. The variation in expert judgment will also increase.

---

[27] There is a recent example with a space system involving a quad-redundant system with four general computers. They were connected by a multiplexor/de-multiplexor module, and one of these was faulty due to a failed diode. It failed in such a way that it was not observed by the four computers similarly. This had not been anticipated when it was built [PM].

[28] Adding backups (or fault protection) can increase complexity and introduce new hidden dependencies [GH].

[29] Although layered protection has benefits, there can be dilemmas from keeping software protected with several layers. Layers increase complexity [PM].

[30] e.g. for a fail-operational system

[31] other than functional and para-functional – also known as service requirements or "-ilities" or non-functional requirements

[32] Reuse is very dangerous because, in a certain sense, you assume that the requirements are the same [including the environment]. But if they are not, we have seen bad results. So it is a nice idea, but you have look at a lot of factors [for correct reuse or fitness of purpose] [MB].

[33] We need to consider issues about the competency of the regulators. Many are not competent enough to make judgments on aspects of detailed software engineering. Processes are critical, and they depend on the quality of the people implementing those processes. You need to have a sufficient number of trained expert regulators who can assess the implementation and not just the processes in engineering [CJ].

A.4 Baseline Architecture Specification Principles

The discussion of uncertainties in this report is based on the satisfaction of the following conditions (treated as the baseline) in the development, verification, and evaluation activities of a DI&C system architecture and software architecture[34] to support safety functions. Failure to satisfy these conditions gives rise to uncertainties beyond those discussed in this report.

1. The architecture supports system verifiability,[35] especially avoiding unnecessary [JM] complexity. The following conditions reduce complexity and improve analyzability—thereby improving verifiability.
2. The allocation of requirements to some function and that function to some element of the system is bidirectionally traceable. This condition is satisfied through all levels in the architectural hierarchy[36] and through all stages of derived requirements and design specifications [JH].
3. The behavior specification avoids mode confusion, especially when functionality is nested [GH, JH].
4. The behavior of every element in the architecture is unambiguously specified. This condition is satisfied through all levels in the architectural hierarchy.
5. The behavior of a nonatomic element is a composition of the behaviors of its constituent elements, with well-defined, unambiguous rules of composition [PJ] [29].
   5.1. Interfaces of elements are unambiguously specified, including behavior.
   5.2. Interactions across elements occur only through their specified interfaces[37].
6. Only required interactions are allowed. The architecture precludes unwanted interactions and unwanted, unknown hidden coupling (see Appendix C) or dependencies,[38] interference, or side effects across its elements: specified information exchanges or communications occur in safe ways.
7. The system is modularized using principles of information hiding and separation of concerns, considering orthogonality of functions and data (i.e., avoiding unnecessary interdependence).
8. Each element (e.g., a software unit) is internally well-architected (satisfying conditions 1–6 above) and relatively simple. For example, for a software unit implementing some NPP safety functions, the following conditions are true:
   8.1. It is composed from atomic functions using well-defined, unambiguous rules of composition (see condition 4).

---

[34] System failures traceable to architecture rank high in numbers found in various safety-critical, mission-critical, high-quality DI&C systems. For example, unwanted interactions, hidden couplings, and side effects have led to unexpected failures; traditional testing or simulation-based verification did not detect such flaws.

[35] Example: [in] a synchronous design … lots of things…are much easier to analyze than in a asynchronous [design], and classes of behaviors can't arise; [an] intrinsically synchronous design avoids a whole bunch of uncertainties….[see in Glossary "Synchronous architecture"].

[36] Some abstractions can mask problems, and that is a risk [PM].

[37] i.e., adhering to principles of encapsulation

[38] One of the accident investigation reports that I like reading is the initial launch of the Ariane 5 rocket. …There were many things that lined up for that incident to take place even though almost all of the decisions made were reasonable [PM].

8.2. The atomic functions use a combination of logical, relational, and simple arithmetic operators.

8.3. The number of inputs and outputs is relatively small.

8.4. The types of inputs and outputs are relatively simple, e.g.:

- Boolean variables
- Physical quantities (pressure, temperature, flow)
- Rates of change of physical quantities.

9. The architecture is explicitly specified in a manner (e.g., language; structure) that is unambiguously comprehensible [JH] to the community of its users (such as reviewers, architects, designers, and implementers) (i.e., the people and the tools they use).

9.1. This condition is satisfied through all levels in the architectural hierarchy, including the interfaces of the elements[39] and their interrelationships, interactions, and interconnections [JH].

9.2. This condition is also satisfied for the mappings from one level to another in the architecture hierarchy [JH]. Conformance to the architecture is assured at all levels, including the code.

To the extent that any condition is not satisfied, independent experts with commensurately higher level of competence are recommended to evaluate the effect on system-failure likelihood. The variation in expert judgment will also increase.

To the extent that the DI&C system architecture satisfies these conditions, an FTA becomes commensurately more credible. The fault sequences could be traced from the externally visible failure event through the paths of composition to the failures of functions in the comprising elements, down to the smallest software unit identified in the architecture. However, the correctness of the FTA is predicated on the assumption that there is no possibility of random data corruption or nondeterministic effects that can cause bizarre and unpredictable software behavior under generally unpredictable conditions. Given that every single execution sequence of a complex software system has a vanishingly small probability of occurrence, there is very little one can say about the reliability of any such system, no matter how long it has actually executed without a failure. Every execution is, for all practical purposes, one that has never been seen before (that is, the precise interleaving and event orderings that occur in that execution). Structural safeguards have to save the day in these cases [GH].

A.5 Baseline Criteria for Evaluating Tool Automation

The discussion of uncertainties in this report is based on the satisfaction of the following criteria, constraints, and guidelines (treated as the baseline) in the evaluation and qualification of tool-automated processes for development and verification of software[40] and other forms of logic implemented in DI&C systems. Failure to satisfy these criteria gives rise to uncertainties beyond those discussed in this report.

As in earlier discussions, uncertainty associated with tool-automated processes is reduced by eliminating the corresponding causes of defects, that is, through means to improve the quality. To the extent that tools are based on software, the uncertainty issues and conditions discussed

[39] Interface specifications are available for all components of the architecture.

[40] Code generators and verifiers are being used for single programs, software components, or other implementation of the logic. Automated integration of components is in its infancy. Thus, this baseline is focused on generating the code for a component and verifying this code.

for software in the DI&C system also apply to the software in the tools. IEEE 7-4.3.2-2003 "IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generation Stations", clause 5.3.2, is based on this position. Therefore, the baseline criteria given in Appendices A.1–A.4 to this RIL also apply to tools and tool-automated processes.

Table 8 provides additional criteria, constraints, and guidelines specific to tool suites and environments used for development and verification of software and other forms of logic implemented in DI&C systems.

Experts' position: There is concern [about the use of] uncertified tools with loosely defined specification formalisms. It is strongly recommended that uncertified tools be considered unacceptable in the production of code (e.g., in a chain of transformations) for safety-critical applications.

Table 8: Evaluation of Tools to produce Safety-Critical Logic

ID	Factor	Criterion/Constraint/Guideline
1	Independence of verification and development	Verification cases for the product are not dependent on the information that tools and other resources use for automated code generation.
2	Transformation process	The process is mechanized (reduced to a routine) correctly. The process activity is deterministic; that is, its input and output are unambiguously defined and the transformation of the input to the output is algorithmically specified.
3	Input	The input language (e.g., for the design model[a]) has a published specification that is unambiguously comprehensible to the community of its users—humans and other tools—engaged in development, verification, or other evaluation activities. Example: Restricted version of a modeling language [11, 12, 13].
4	Output	The output language (e.g., the programming language) has a published specification that is unambiguously comprehensible to the community of its users. Example: Restricted version [24] of a type-safe language.
5	Composition rules	There are unambiguous, published rules of composition in the source language and the target language.
6	Elements mappable	There is an unambiguous mapping (transformation) from each source element to a corresponding target element or composition of elements in the target language, such that the mapping is backward-traceable.
7	Compositions mappable	There is an unambiguous mapping (transformation[c]) from a composition of source language elements to a composition of target language elements.
8	Transformation rules	The transformation rules are distinctly identifiable, unambiguous, and verifiable.
9	Architecture	The architecture of the tool provides clear distinction and independence of the following key elements: • Input • Output • Transformation rules and associated data

ID	Factor	Criterion/Constraint/Guideline
		• Transforming mechanism • User interface • Environment in which the input artifact is produced • Environment in which the output artifact is used
10	Complexity	Unnecessary complexity of the software is avoided by using sound architectural principles such as separation of concerns, encapsulation, unambiguously specified interfaces, and unambiguously specified interactions, as described above and in A.4 Baseline Architecture Specification Principles."
11	Published limitations	The users of the tool are aware of its limitations and conditions of use.
12	User competence	The users of the tool are competent[b] in its correct use for the assigned process activity, considering known limitations.
13	Developer competence	People developing and maintaining tools and tool suites possess and maintain the requisite competence, commensurate with the complexity of the assigned tasks.
14	Community of users	The individual persons and other tools engaged in development, verification, or other evaluation activities or dependent on the tool are identified explicitly, are qualified to use the tool correctly, and are included in the configuration-managed set for which the tool is qualified.
15	Configuration management	The tool and all items and factors on which the correctness of the tool is predicated are configuration-managed, verified, and validated as an integrated set (e.g., the restricted versions of the input and output languages, the community of users; libraries).

[a] A limitation observed by experts is the ability to build models with accurate enough abstractions.

[b] Certified by an independent certification authority.

[c] In a chain of transformations, it is recommended that semantic consistency be assured across all tools involved.

A.6 Safety Demonstration Framework

The applicant for certification or license should demonstrate the satisfaction of the safety goal through a logical (argument-based) organization and integration of the evidence (see Figure 6) from verification, validation, and audit activities [JH, JM, PJ, PM]. This is sometimes called a "safety demonstration" and, in some communities, it is known as an "assurance case" and for the whole system a "safety case" [27]. It is not enough to show only a clause-by-clause compliance with the applicable requirements, guidelines, and standards [28].[41] Each item of evidence from verification, validation, or audit by itself does not provide adequate assurance. The evidence should include diverse redundancy (e.g., testing to complement formal verification [JK, GH] integrated with the reasoning to demonstrate how it reduces the uncertainty of meeting the safety goal [JM]. The demonstration should also include an invitation for challenges (as depicted in Figure 6), their resolution, and, to the extent that challenges,[42] rebuttals, and

[41] Satisfaction of programmatic requirements is not enough [28].

[42] UK defense standard 0056 [26] added in the last edition a requirement to search for counter evidence, things which show it isn't safe [JM].

inconsistencies are not fully resolved, commensurate qualification of the degree or strength of the validity of the claim or assertion.

Criteria and conditions given in Appendices A.1–A.4 to this RIL will improve the systematization of the safety demonstration framework and the expert judgment process. Even then, the evaluation will require significant expertise, involving project-specific customization and judgment. The systematization of the expert judgment process is still a subject of research [JM].

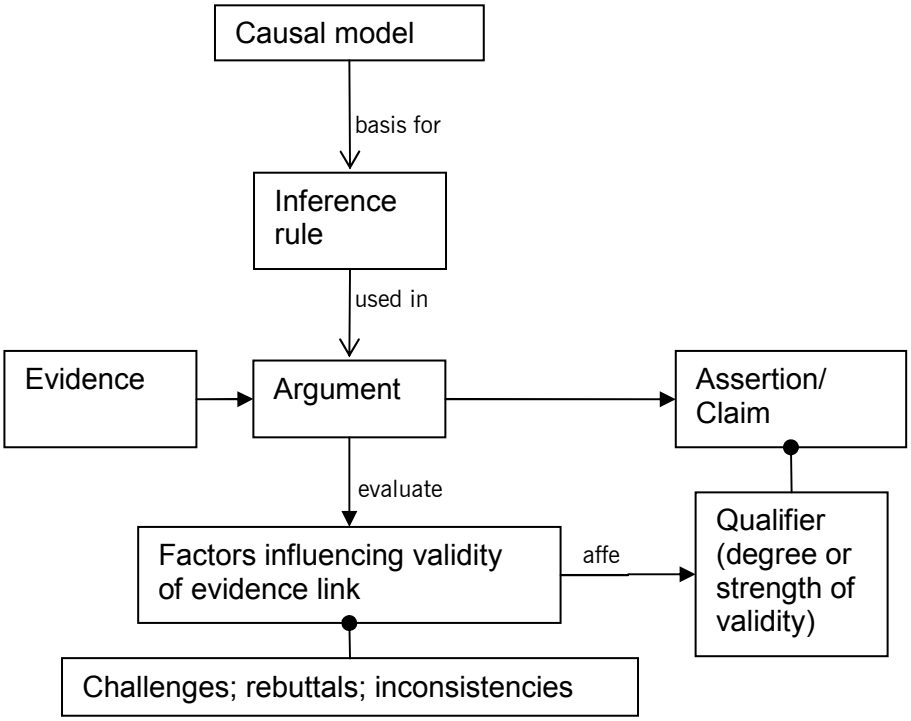


Figure 6 Evidence-argument-claim structure

A safety plan based on such a safety demonstration framework can be instrumental in generating significant measurable evidence early in the project lifecycle, avoiding surprises later.

Appendix B Expert Elicitation Process

The expert elicitation process was designed and executed with the purpose of acquiring knowledge outside the nuclear power plant (NPP) domain relevant to understanding software-related uncertainties in the assurance of digital instrumentation and control (DI&C) systems used for NPP safety functions.

The process was designed for the elicitation to do the following:

- Be supported with evidence, appropriately qualified for its degree of validity.
- Synthesize complementary individual knowledge, applying qualitative reasoning.
- Use collective judgment to integrate and apply this knowledge to the NPP domain.
- Identify reasons for differences across experts (e.g., different underlying assumptions).

The elicitation process is outlined in the form of a flow diagram in Figure 7, followed by an explanation of each step in the flow.

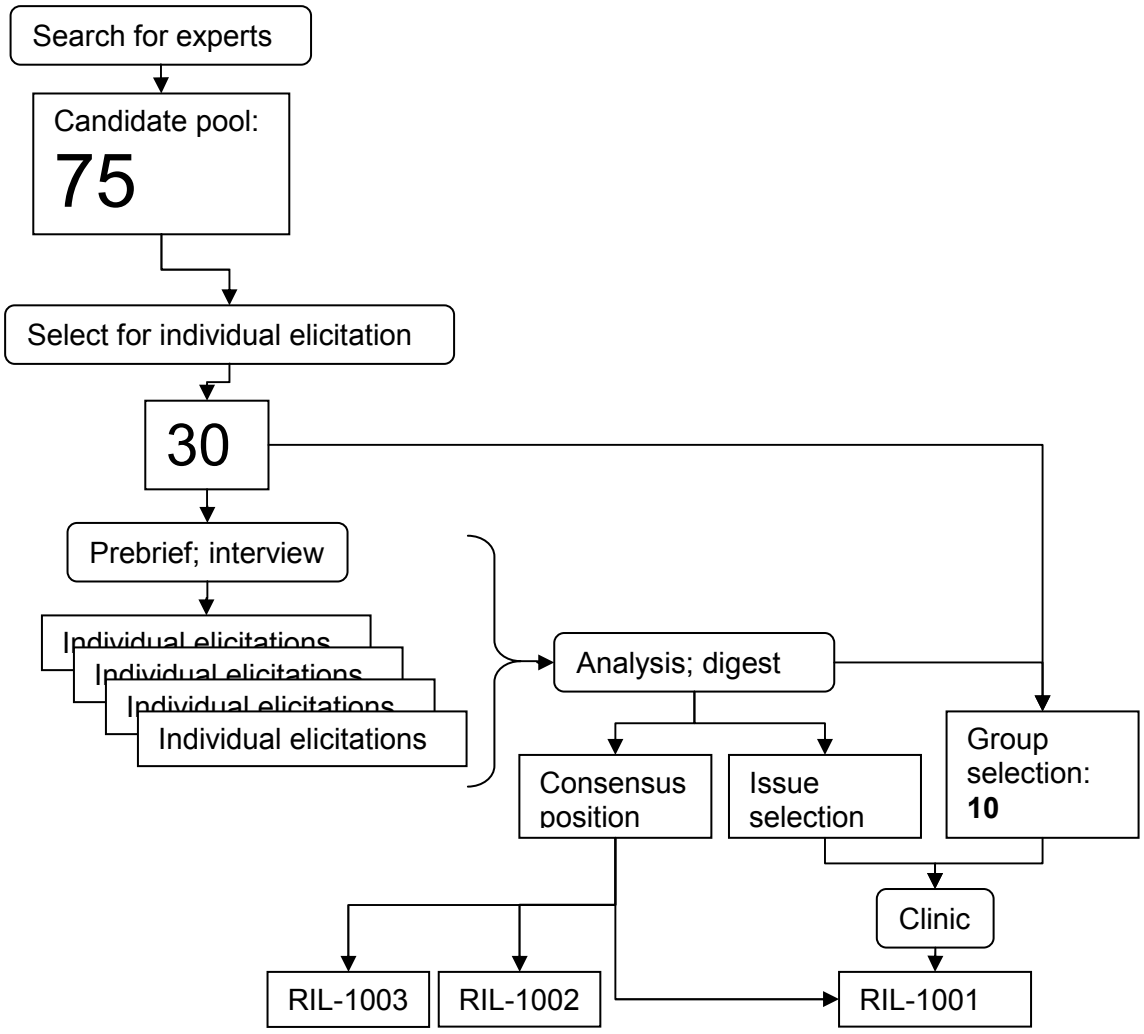


Figure 7 Design of overall elicitation process

Building the Candidate Pool of Subject-Matter Experts

Based on the purpose and objectives defined for the project, the U.S. Nuclear Regulatory Commission (NRC) identified the complement of expertise needed and desired. NRC internal experts identified an initial set of subject-matter experts outside the NPP domain, based on their own knowledge of researchers in the assurance of software-reliant safety-critical systems who could contribute to the desired complement of expertise. When each member of the candidate pool was contacted to explore a match of interest, Pacific Northwest National Laboratories (PNNL) asked the expert to suggest other experts who should be contacted for a match of interest. Their suggestions were added to the candidate pool. Similarly, each new contact was asked for referrals. PNNL team members and NRC experts also reviewed the candidate pool periodically and suggested other names. This process of successive referrals built up a candidate pool of 75. PNNL searched for publicly available information about and publications by each candidate.

Selecting for Individual Elicitation

Based on information publicly available, collected from the referring individuals, and obtained from the candidates, PNNL sequenced contacts with candidates to explore matches or conflicts of interest and to determine availability. PNNL screened candidates and selected approximately 30 to schedule for individual briefing and elicitation.

The selection focused on experience outside the commercial NPP industry, such as defense, space flight, commercial aviation, medical devices, automobiles, telecommunications, and railways, spread across seven countries (United Kingdom, Sweden, Germany, Canada, United States, Australia, and New Zealand).

Prebriefing

Before individual elicitation, each of the 30 candidates was briefed on the background: the NPP application domain and the issues about which his or her expertise was to be solicited. The purpose of the briefing was to assist the expert in retrieving and focusing his or her knowledge about the problem domain. At this stage, some candidates decided to withdraw from further engagement or suggested other candidates.

Individual Elicitation

After the orientation, the PNNL team interviewed each expert, aided by an inventory of questions. Interviews adapted to the information flow dynamics specific to each individual. With the consent of the expert, the interview was audio-recorded to aid future recollection. Later, PNNL created a text summary of the interview, mapping the information to the relevant topic or question in the inventory to the extent readily feasible. In selected cases, PNNL invited the expert's review and further input. In some cases, for clarification or to probe further, PNNL conducted a second interview or follow-up e-mail messages.

Analysis and Digest

The NRC analyzed these interview summaries progressively and iteratively to identify areas of broad consensus, points of difference, reasons for the differences, and issues worthy of further discussion, clarification, or resolution. At the same time, the NRC began identifying experts who could be strong contributors in moving forward on these issues in an expert focus group setting.

The collection of individual elicitations had not been able to cover all aspects of the NRC's initial scope. The elicited information was also sparse in some topics and rich in others.

The NRC's digest, in the form of a reference position document was sent to the interviewees for their review feedback. This process helped solidify areas of broad consensus and helped identify issues requiring face-to-face discussion by a suitably selected focus group. The design of the group interaction—the expert clinic—was finalized accordingly, including the topics and questions for discussion and the selection of 10 participants.

Consensus Position

The consensus findings were digested in the form of a reference position that included references to the individual elicitations and reflected back on the sources cited. As feedback or additional information was received, the reference position was modified.

Selection of Issues for Focus Group Discussion

The amount of information collected was too large to review and discuss in a group setting for further refinement. Therefore, the NRC analyzed the information for most relevance and best value to the program offices, considering issues encountered in recent reviews. Based on this analysis, the NRC selected certain topics for discussion by a focus group.

Focus Group

Based on their contribution to the topics selected, diversity of perspectives, and availability, the NRC selected 10 experts outside of the NRC and NPP industry for a 2-day meeting in a focus group setting.

Clinic

The focus group was notified, in advance, of the topics selected for group discussion.
For each of the selected topics, the focus group was charged with the following objectives:

- Identify limitations in the current state-of-practice (i.e., sources of uncertainty), due to which software assurance is heavily dependent on expert judgment.
- Identify the evidence that is needed to more effectively assure software for safety, based on best practices in other application domains.
- Identify knowledge gaps to be filled (i.e., areas in need of research and development) to enable more consistent reviews and to reduce judgment-based variation.

In addition to the 10 experts in the focus group, the clinic included other participants as follows:

- Two experts from U.S. Naval Reactors and one from U.S. Food and Drug Administration participated by providing comments at the introduction and conclusion of the clinic and at invited points during each segment of the clinic.
- Five experts from the NRC provided the context of the problem domain:
  - NPP application domain
  - Regulatory framework and boundaries
  - Difficult technical issues experienced in licensing reviews
- PNNL staff facilitated the execution of the clinic.

All discussions were audio-recorded and transcribed at the end of the day for review (.

All participants had access to laptops, networked and facilitated for real-time communication among all participants. The facilitator invited all participants for their inputs and interactions through the networked laptops. For example, an NRC expert and a focus group member could conduct a clarification interchange concurrent with oral discussions in the focus group.

For each of the topics discussed, the focus group provided the following:

- a summary of its conclusions
- additional (post-it) notes written for immediate display near the end of the topic segment
- inputs through laptops—chat logs

Formation of RIL-1001

PNNL reviewed various forms of input, including the pre-clinic individual elicitations and the consensus position , selected excerpts relevant to each discussed topic, and organized the snippets and excerpts accordingly.

The NRC analyzed the excerpts collected for each topic and reorganized it into RIL-1001 (this document), focusing the main body on information elicited during the clinic and placing other relevant information from the consensus position in Appendix A.

Information for RIL-1002 and RIL-1003

While RIL-1001 is focused on uncertainties in assurance of software in digital safety systems and other manifestations of complex logic, the remaining information from the consensus position and individual elicitations pertaining to identifying and analyzing DI&C failure modes will be addressed in RIL-1002 [1]. The remaining information from the consensus position and individual elicitations pertaining to the feasibility of applying failure mode analysis to quantification of risk associated with DI&C will be addressed in RIL-1003 [2].

References for Appendix B

1. U.S. Nuclear Regulatory Commission, “Identification of Failure Modes in Digital Safety Systems and Analysis for Systemic Causes—Expert Clinic Findings, Part 2,” Research Information Letter RIL-1002.
2. U.S. Nuclear Regulatory Commission, “Feasibility of Applying Failure Mode Analysis to Quantification of Risk Associated with Digital Safety Systems—Expert Clinic Findings, Part 3,” Research Information Letter RIL-1003.

Appendix C – Hidden couplings can lead to severe consequences

Failures with severe consequences can result from unexpected combinations of seemingly benign "minor" faults. As systems get larger and more complex, typically, failure combinations are not (and generally cannot be) studied in detail, because there are far too many possible combinations, as explained below. Thus, even though each separate failure cause can be well understood, the combinations are not. Some combination could cause a failure with a severe consequence [1].

Consider an example of a system with 10^2 seemingly "minor" latent[1] faults, there can be up to 10^4 possible combinations of two faults and 10^6 combinations of three, and so on - each with a vanishingly small probability of occurrence. Two such latent faults could fall in the same execution path; or one fault, when activated in one execution, could change the system state such that a second fault, when activated in another execution, causes a different failure. Thus, two latent faults combine in an unforeseeable way and cause a behavior that could be catastrophic. The Three-Mile Island accident is an example of a catastrophe caused by a combination of several seemingly "minor faults" - each harmless by itself. However, because each latent fault has a small probability of being triggered in an execution, the combination of two or more such small faults has an even smaller probability. However, even if some case has a probability 1/N with a very large value for N, it could still be a concern if there are M such potential cases, with M also being very large and approaching N. Several studies have shown that serious software failures can be caused by very low probability events (cases for which the software was not tested thoroughly). Typically, in rarely executed code, such as exception[2] handling code, a good example of a combination of seemingly "minor" defects is a relatively benign fault triggering a rarely executed exception handler, which itself contains a fault. [GH]

Reference for Appendix C

[1] Charles Perrow, "Normal Accidents: Living with High Risk Technologies", New York: Basic Books, 1984.

[1] A latent defect is one that is dormant in the code, but has not revealed itself yet (for instance because that piece of code has not yet been executed under precisely the circumstances that are required to reveal the fault and cause a problem).

[2] Commonly called "error"