Evaluation of Guidance for Tools Used to Develop Safety-Related Digital Instrumentation and Control Software for Nuclear Power Plants

Task 1 Report: Survey of the State of Practice

Prepared by:
James R. Servatius
Paul Butchart

Reviewed by:
Stephen Alexander
William Arcieri

Energy and Space Division
Information Systems Laboratories, Inc.
11140 Rockville Pike, Suite 650
Rockville, MD 20852

July 2013

Prepared for:

U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
NRC Contract No: NRC-HQ-13-C-04-0004
ISL Project No. 2310

Intentionally Blank

Executive Summary

Current and future nuclear power plants will rely on digital (computer-based) instrumentation and control (I&C) systems and other types of software-driven systems which have been and are being developed using various types of software tools. Tool-automated and tool-assisted engineering activities may result in fewer defects than with manual engineering activities if tools are well designed, carefully developed, effectively tested, and appropriately used. However, unknown deficiencies in automated tools or tool-assisted engineering activities may affect the safety assurability of a digital safety-related system, dependent on such tools.

As defined in Institute of Electrical and Electronic Engineers (IEEE) Standard (Std.) 7-4.3.2-2010 [1], software tools are computer programs used in the design, development, testing, review, analysis, or maintenance of a program or its documentation. The Nuclear Regulatory Commission (NRC) considers that software tools are computer programs or computer-based engineering environments used in the specification of requirements, architecture, design, implementation, testing, and other forms of verification, review, analysis, validation, and documentation of a system or such components of the system as software, and maintenance of these artifacts or work products. Examples include compilers, assemblers, linkers, comparators, cross-reference generators, decompilers, editors, flow charters, monitors, test case generators, integrated development environments, timing analyzers, and code generators, among many others.

The NRC anticipates that more software development applications may include the use of software tools, and the existing guidance may not be adequate for consistent application. NRC Contract NRC-HQ-13-C-04-0004 is a research project intended to develop the technical basis for improved regulatory guidance that will follow industry standards and support NRC regulatory requirements. The improved regulatory guidance should be consistently implementable. The NRC seeks to document existing regulatory guidance, standards, practices, and experience concerning review and/or approval of the use of software tools in engineering systems in industries across the world. Most of the existing documents are not specifically designed for software tool reviews and the guidance is not always directly relevant to software tools. However, the documents were surveyed for their possible/indirect relevance to software tools.

This research project is phased in three tasks. In Task 1, Information Systems Laboratories, Inc. (ISL), the contractor, will survey standards, regulatory guidance, review/approval practices, and other current industry practices concerning the use of software tools on which the safety assurability of a digital safety-related system is dependent. The product of Task 1 is a report summarizing the results of the survey including a broad baseline of requirements covered by all the industry standards and a broad baseline of guidance and review/approval practices covered by all the regulatory guidance and industry review/approval practices. In Task 2, ISL will analyze the Task 1 findings and baselines, and modify the baselines for NRC use. The product of Task 2 will be a report on the analysis of the findings of Task 1. It will include an assessment of the vulnerabilities associated with the use of software tools in different stages of the engineering life cycle (e.g., challenges to assurability, verifiability, and analyzability) and corresponding criteria to qualify the tools for representative uses. In Task 3, ISL will propose

recommendations for influencing public consensus standards and a phased plan to incorporate review criteria in NRC's regulatory guidance. The product of Task 3 will be a corresponding report.

This report is the product of contract NRC-HQ-13-C-04-0004 Task 1. ISL surveyed and compared numerous documents from the space industry, the civil aviation industry, the NRC and commercial nuclear power industry, the IEEE, the International Electrotechnical Commission (IEC), the International Atomic Energy Agency (IAEA), the Digital Instrumentation and Control Working Group (DICWG) of the Multinational Design Evaluation Program (MDEP) of the Nuclear Energy Agency (NEA) of the Organization for Economic Cooperation and Development (OECD), the Electric Power Research Institute (EPRI), the National Institute of Standards and Technology (NIST), and the Atomic Energy of Canada, Ltd (AECL). The survey focused on standards, regulatory guidance, and review/approval practices related to the use of software tools, commercial-off-the-shelf (COTS) software tools, government-off-the-shelf (GOTS) software tools, and modified-off-the-shelf (MOTS) software tools. This report summarizes the content of each document with specific emphasis on discussions of software tools.

Evaluation and qualification of a tool is intimately dependent on the process activity for which it is used and the context of the lifecycle process in which that activity is performed. Therefore, for purposes of this report, the system and software life cycle processes of IEEE Std. 12207-2008 [2] have been used as a reference framework. This selection was based on a survey and evaluation of a wide range of lifecycle process standards, as a preliminary step in Task 2. This report does not endorse any particular life cycle model or any particular set of life cycle processes.

ISL has researched industry standards, guidance, and practices to determine the types of software tools used in the nuclear and other industries and the general philosophy for software tool use during software development. ISL has prepared a generic list of software tools and created a list of software tool categories based upon tools that are widely used within the software development industry. Based on similarities between certain software tools and software tool usage and availability in certain life cycle processes, all software tools have been grouped into nine software tool categories.

The survey found little information pertaining to software tools in the nuclear field internationally. While some guidance does exist in other industries, software tools are still not widely addressed or regulated except for the civil aviation industry, which contains several documents providing guidance and review/approval practice for software tool use, and IEEE standards that discuss the abundance of computer-aided software engineering (CASE) tools. Based on the extensive survey of major industries, ISL believes that the IEEE standards related to the selection, use, evaluation, and qualification of CASE tools along with the Federal Aviation Administration (FAA) standards, FAA orders, FAA advisory circulars, Certification Authorities Software Team (CAST) papers, and Radio Technical Commission for Aeronautics (RTCA) documents, may provide a basis upon which to build software tool (including COTS, GOTS, and MOTS software tools)

regulation, guidance, and review/approval practices for the commercial nuclear industry pending further analysis.

Based on the information in all of the documents surveyed, ISL has found that software tools are used in many of the software and system life cycle processes; however, guidance and review/approval practices are typically limited to software tools used in processes, activities, and tasks that have the potential of introducing defects or failing to detect defects in the final safety-related software. ISL considers that software tool regulatory guidance and review/approval practices should be expanded to cover more software life cycle processes when assurability is dependent on that tool. Based on the activities, tasks, and products associated with the system-context and software-specific processes of IEEE Std. 12207-2008 [2], and whether a tool used during a specific process impacts the final nuclear plant software, ISL believes that some of the system-context processes and most of the software-specific processes should be subject to regulatory guidance.

ISL has produced a broad baseline of requirements covering all the industry standards for the use of software tools (including COTS, GOTS, and MOTS software tools) and a broad baseline of guidance and review/approval practice covering all the regulatory guidance. The baseline requirements for software tools to support regulatory assurance engender the need for proper evaluation and qualification, adequate configuration management, thorough verification and validation (V&V), proper use as described by the tool's intended purpose, correct software tool classification level, thorough up-front planning, continuous quality assurance (QA), and adequate training on tool use. The baseline guidance and review/approval practices for software tools to support regulatory assurance include evaluation and qualification guidance, configuration management practices, V&V practices, guidance and limitations on tool use, tool classification level guidance, planning practices, QA guidance, and training practices.

Note that the term “qualification,” as used in the preceding paragraph, is used in aerospace industry and related government documents as a “term of art” with very specific meaning in that context; that is, a systematic process through which software tools are verified to be suitable for their intended use(s) and to the extent possible, to be free of deficiencies that could adversely affect critical DI&C system software or result in failure to detect defects in that software. However, in the commercial nuclear industry, in accordance with Part 21, “Reporting of Defects and Noncompliance,” of Title 10 of the *Code of Federal Regulations* (10 CFR Part 21), the corresponding process, called “qualification” in the aerospace industry, is called *commercial-grade dedication*, when applied to certain software tools of the type of concern, intended for use with safety-related software for nuclear power plants, when those tools were not designed or developed under a QA program meeting the requirements of 10 CFR Part 50, Appendix B, “Quality Assurance Criteria for Nuclear Power Plants and Fuel Reprocessing Plants.”

Under 10 CFR Part 21, software tools of the type of concern comprise COTS software tools (considered commercial-grade items (CGIs)) that are used or intended for use in the development or testing of safety-related software, which, under Part 21, is considered a *basic component*. For nuclear power plants, *basic components* are defined in 10 CFR Part 21 as safety-related components that have been designed and manufactured (developed for software)

under an Appendix B QA program or that have successfully undergone commercial-grade dedication (qualification for software in aerospace terms), which must itself have been performed under an Appendix B QA program. Thus, such COTS software tools of the type of concern must be dedicated (qualified) in accordance with 10 CFR Part 21. GOTS and MOTS software tools of the type of concern would also require dedication (qualification) to the extent that they were not designed and/or developed, or not modified under an Appendix B QA program.

Note also that in commercial nuclear parlance, the term “qualification” is also a term of art, i.e., has special meaning in this context, but refers to a design verification process, as defined in Criterion III, “Design Control,” of Appendix B to 10 CFR Part 50, which is usually associated with the verification that the design of safety-related hardware (including DI&C hardware) is suitable for its application and will perform its intended function(s) under all design-basis service conditions (seismic and environmental being the chief concerns). This means verification that a component of the design being verified would not be expected to be vulnerable to common-mode or common-cause failures (CCFs), i.e., due to a deficiency inherent to the design, under those postulated design-basis conditions. ISL recognizes this ambiguity and thus endeavors in this report to use the terms *dedication* and *qualification* consistent with their context.

Table of Contents

Tables

Intentionally Blank

Abbreviations and Acronyms

AC	FAA Advisory Circular
ACG	Automatic Code Generation
AECL	Atomic Energy of Canada, Ltd
AIR	Aircraft Certification Service
BTP	Branch Technical Position
CASE	Computer-Aided Software Engineering
CAST	Certification Authorities Software Team
CCF	Common-Cause Failures
CFR	Code of Federal Regulations
CGI	Commercial-Grade Item
CIO	Chief Information Officer
CM	Configuration Management
CNS/ATM	Communications, Navigation, Surveillance, and Air Traffic Management
COTS	Commercial-off-the-Shelf
DBA	Design-Basis Accident
DBE	Design-Basis Event
DBMS	Database Management System
DER	Designated Engineering Representative
DI&C	Digital Instrumentation and Control
DICWG	Digital I&C Working Group
DID	Data Item Descriptions
DP	Discussion Paper
DO	RTCA Document
DOT	U.S. Department of Transportation
EIA	Energy Information Administration
EIA	Electronic Industries Alliance
EMI	Electromagnetic Interference
EPRI	Electric Power Research Institute
FAA	Federal Aviation Administration
FAQ	Frequently Asked Question
FPGA	Field-Programmable Gate Array
GB	Guidebook
GDC	General Design Criteria
GL	NRC Generic Letter
GOTS	Government-Off-the-Shelf
GUI	Graphical User Interface
I&C	Instrumentation and Control
IAEA	International Atomic Energy Agency
IDE	Integrated Development Environment

IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFP	Instrument Flight Procedure
ISG	Interim Staff Guidance
ISL	Information Systems Laboratories, Inc.
ISO	International Organization for Standardization
IT	Information Technology
MDEP	Multinational Design Evaluation Program
MOTS	Modified-Off-the-Shelf
NAS	National Airspace System
NASA	National Aeronautics and Space Administration
NCIG	Nuclear Construction Issues Group
NDS	Non-Developmental Software
NEA	Nuclear Energy Agency
NEI	Nuclear Energy Institute
NIST	National Institute of Standards and Technology
NP	EPRI Notification Paper
NPP	Nuclear Power Plant
NPR	NASA Procedural Requirements
NRC	U.S. Nuclear Regulatory Commission
NSSS	Nuclear Steam Supply System
NUPIC	Nuclear Procurement Issues Council
NUMARC	Nuclear Management Resources Council
NUREG	U.S. Nuclear Regulatory Publication
OECD	Organization for Economic Cooperation and Development
OPA	Organizational Process Assets
PAL	Process Asset Library
PAM	Post-Accident Monitoring
PBN	Performance-Based Navigation
PLC	Programmable Logic Controller
PSAC	Plans for Software Aspects of Certification
QA	Quality Assurance
RFI	Radio-frequency Interference
RG	Regulatory Guide (NRC)
RSC	Reusable Software Component
RTAS	Reusable Tool Accomplishment Summary
RTCA	Radio Technical Commission for Aeronautics
RTCI	Reusable Tool Configuration Index
RTDS	Reusable Tool Data Sheet
RTQP	Reusable Tool Qualification Plan
SAS	Software Accomplishment Summary

SCI	Software Configuration Index
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SDLC	System Development Lifecycle
SDOE	Secure Development and Operational Environment
SDP	Software Development Plan
SER	Safety Evaluation Report
SME	Software Management Environment
SP	Special Publication
SPH	Standards and Procedures Handbook
SPLC	Software Project Life Cycle
SPLCM	Software Project Life Cycle Model
SPLCP	Software Project Life Cycle Process
SQA	Software Quality Assurance
SRP	Standard Review Plan
SRS	Software Requirements Specification
STC	Steering Technical Committee
STD, Std.	Standard
SVP	Software Verification Plan
SVTAS	Software Verification Tools Assessment Study
SWEBOK	Software Engineering Body of Knowledge
TQL	Tool Qualification Level
TR	Topical or Technical Report
TSO	Technical Standard Order
UMS	User-Modifiable Software
V&V	Verification and Validation

Intentionally Blank

Glossary

accident: An unplanned event or series of events that result in death, injury, illness, environmental damage, or damage to, or loss of, equipment or property. {RG 1.173 Rev 1 [4]}

activity: Tasks that provide a means of meeting the objectives. {RTCA/DO-178C [28]}

assurability: Ability to confirm the certainties of correctness of a claim, based on evidence and reasoning.

basic component: A safety-related structure, system, or component as defined below. {10 CFR 21.3}

(1)(i) When applied to nuclear power plants licensed under 10 CFR part 50 or part 52, basic component means a structure, system, or component, or part thereof that affects its safety function necessary to assure:

(A) The integrity of the reactor coolant pressure boundary;

(B) The capability to shut down the reactor and maintain it in a safe shutdown condition; or

(C) The capability to prevent or mitigate the consequences of accidents which could result in potential offsite exposures comparable to those referred to in § 50.34(a)(1), § 50.67(b)(2), or § 100.11, as applicable.

(ii) Basic components are items designed and manufactured under a quality assurance program complying with appendix B to part 50, or commercial-grade items which have successfully completed the dedication process.

(2) Basic component applicability to standard design certifications under 10 CFR Part 52, Subpart C - Not directly pertinent to software tools

(3) When applied to other facilities and other activities licensed under 10 CFR parts 30, 40, 50 (other than nuclear power plants), 60, 61, 63, 70, 71, or 72 of this chapter, basic component means a structure, system, or component, or part thereof, that affects their safety function, that is directly procured by the licensee of a facility or activity subject to the regulations in this part and in which a defect or failure to comply with any applicable regulation in this chapter, order, or license issued by the Commission could create a substantial safety hazard.

(4) In all cases, basic component includes safety-related design, analysis, inspection, testing, fabrication, replacement of parts, or consulting services that are associated with the component hardware, design certification, design approval, or information in support of an early site permit application under part 52 of this chapter, whether these services are performed by the component supplier or others.

certification: The legal recognition by the certification authority that a product, service, organization, or person complies with the requirements. Such certification comprises the activity of technically checking the product, service, organization, or person and the formal recognition of compliance with the applicable requirements by issue of a certificate, license, approval, or other documents as required by national laws and procedures. {RTCA/DO-178C [28]}

certification authority: The organization or person responsible within the state, country, or other relevant body, concerned with the certification or approval of a product in accordance with requirements. {RTCA/DO-178C [28]}

Class 1E: The safety classification of the electric equipment and systems that are essential to emergency reactor shutdown, containment isolation, reactor core cooling, and containment and reactor heat removal, or are otherwise essential in preventing significant release of radioactive material to the environment. {NUREG/CR-6421 [68], Section 1.3}

commercial-grade dedication: An acceptance process undertaken to provide reasonable assurance that a commercial-grade item to be used as a basic component will perform its intended safety function and, in this respect, will be deemed equivalent to an item designed and manufactured under a 10 CFR Part 50, Appendix B, quality assurance program. {NUREG/CR-6421 [68], Section 1.3} (See also 10 CFR 21.3 definition of dedication below (same as commercial-grade dedication in this context) Commercial-grade dedication, when applied to software or software tools, especially COTS, MOTS or GOTS software or software tools, is equivalent to the process of “qualification” as described in RTCA/DO-330.

commercial-grade item: (1) A structure, system, or component, or part thereof, that is used in the design of a nuclear power plant and which could affect the safety function of the plant, but was not designed and manufactured as a basic component. {NUREG/CR-6421 [68], Section 1.3} (2) When applied to nuclear power plants licensed pursuant to 10 CFR Part 50, commercial-grade item means a structure, system, or component, or part thereof that affects its safety function, that was not designed and manufactured as a basic component. Commercial-grade items do not include items where the design and manufacturing process require in-process inspections and verifications to ensure that defects or failures to comply are identified and corrected (i.e., one or more critical characteristics of the item cannot be verified). {10 CFR 21.3}

common-cause failure: Loss of function to multiple structures, systems, or components due to a shared root cause. This is important to an understanding of software tools as the same tool may be used in development of software for multiple systems thus making a fault in the tool a potential source of a common-cause failure. {IEEE Std. 603-2009 [71], Clause 3}

complexity: The degree to which a system or component has a design or implementation that is difficult to understand and verify. {IEEE Std. 100-2000 [136]}

configuration item: (1) One or more hardware of software components treated as a unit for configuration management purposes. (2) Software life cycle data treated as a unit for configuration management purposes. {RTCA/DO-178C [28]}

configuration management: (1) The process of (a) identifying and defining the configuration items of a system; (b) controlling the release and change of these items throughout the software life cycle; (c) recording and reporting the status of configuration items and problem reports; and (d) verifying the completeness and correctness of configuration items. (2) A discipline applying technical and administrative direction and surveillance to (a) identify and record the functional and physical characteristics of a configuration item; (b) control changes to those characteristics; and (c) record and report change control processing and implementation status.. {RTCA/DO-178C [28]}

critical characteristics: When applied to nuclear power plants licensed pursuant to 10 CFR Part 50, critical characteristics are those important design, material, and performance characteristics of a commercial-grade item that, once verified, will provide reasonable assurance that the item will perform its intended safety function. {10 CFR 21.3}

critical characteristics for design: The identifiable and/or measurable attributes of a replacement item which provides assurance that the replacement item will perform its design function. {EPRI 102260 [105], Section 2.2.1.2} Note that NRC Generic Letter 91-05 promulgated the NRC staff position that did not distinguish between critical characteristics for design and critical characteristics for acceptance. Rather it referred to the requirements of Criterion III, “Design Control,” and Criterion VII, “Control of Purchased Material, Equipment, and Services,” that licensees must assure the suitability of all material, equipment, and services for their intended safety-related applications. As discussed the cited generic letters, the NRC staff position was and is that not all design characteristics are critical, but if a characteristic is considered critical, i.e., its verification is essential to providing reasonable assurance that the item will perform its intended safety functions when required under all design-basis conditions, then it must be somehow verified, even if indirectly.

critical characteristics for acceptance: The identifiable and measurable attributes/variables of a commercial-grade item, which once selected to be verified, provide reasonable assurance that the item received is the item specified. {EPRI 102260 [105], Section 2.2.1.2} This position was not endorsed by the NRC in Generic Letter 89-02, and the NRC staff position on this EPRI definition was clarified in Generic Letter 91-05 as discussed above. The definition of critical characteristics in 10 CFR 21.3 as shown above is the one that must be used in connection with commercial-grade dedication of software tools used in the development of safety-related software for nuclear power plants.

critical software: Software that is part of, or could affect, the safety function of a basic component or a commercial-grade software item that undergoes commercial-grade dedication. {NUREG/CR-6421 [68], Section 1.3}

dedication (See commercial-graded dedication above): (1) When applied to nuclear power plants licensed pursuant to 10 CFR Part 50, dedication is an acceptance process undertaken to

provide reasonable assurance that a commercial-grade item to be used as a basic component will perform its intended safety function and, in this respect, is deemed equivalent to an item designed and manufactured under a 10 CFR Part 50, Appendix B, quality assurance program. This assurance is achieved by identifying the critical characteristics of the item and verifying their acceptability by inspections, tests, or analyses performed by the purchaser or third-party dedicating entity after delivery, supplemented as necessary by one or more of the following: commercial-grade surveys; product inspections or witness at holdpoints at the manufacturer's facility, and analysis of historical records for acceptable performance. In all cases, the dedication process must be conducted in accordance with the applicable provisions of 10 CFR Part 50, Appendix B. The process is considered complete when the item is designated for use as a basic component. {10 CFR 21.3}

defect: (1) An imperfection or deficiency in a project component where that component does not meet its requirements or specifications and needs to be either repaired or replaced. {ISO/IEC/IEEE 24765 [137]} (2) In the context of equipment, systems, or components (including software) important to safety in NRC-licensed facilities (nuclear power plants in particular), the term “defect” has a special meaning as set forth in 10 CFR 21.3:

(1) A deviation in a basic component delivered to a purchaser for use in a facility or an activity subject to the regulations in this part if, on the basis of an evaluation, the deviation could create a substantial safety hazard;

(2) The installation, use, or operation of a basic component containing a defect as defined in this section;

(3) A deviation in a portion of a facility subject to the early site permit, standard design certification, standard design approval, construction permit, combined license or manufacturing licensing requirements of part 50 or part 52 of this chapter, provided the deviation could, on the basis of an evaluation, create a substantial safety hazard and the portion of the facility containing the deviation has been offered to the purchaser for acceptance;

(4) A condition or circumstance involving a basic component that could contribute to the exceeding of a safety limit, as defined in the technical specifications of a license for operation issued under part 50 or part 52 of this chapter; or

(5) An error, omission or other circumstance in a design certification, or standard design approval that, on the basis of an evaluation, could create a substantial safety hazard. {10 CFR 21.3}

deviation: A departure from the technical requirements included in a procurement document, or specified in early site permit information, a standard design certification or standard design approval. {10 CFR 21.3}

error: (1) With respect to software, a mistake in requirements, design, or code. {RTCA/DO-178C [28]} (2) An incorrect state of hardware, software, or data resulting from a fault. {NUREG/CR-6101 [66]} (3) The difference between a computed, observed, or measured value

or condition and the true, specified, or theoretically correct value or condition. {IEEE Std. 100-2000 [136]}

failure: (1) The inability of a system or system component to perform a required function within specified limits A failure may be produced when a fault is encountered. {RTCA/DO-178C [28]} (2) The external manifestation of an error as seen by a (human or physical device) user, or by another program. {NUREG/CR-6101 [66]}

fault: (1) A manifestation of an error in software. A fault, if it occurs, may cause a failure. {RTCA/DO-178C [28]} (2) A deviation of the behavior of a computer system from the authoritative specification of its behavior. {NUREG/CR-6101 [66]}

field-programmable gate array: An integrated circuit designed to be configured by a customer or a designer after manufacturing. {Wikipedia}

important to safety: A structure, system, or component (a) whose failure could lead to a significant radiation hazard, (b) that prevents anticipated operational occurrences from leading to accident conditions, or (c) that is provided to mitigate consequences of failure of other structures, systems, or components. {NUREG/CR-6421 [68], Section 1.3} In NRC Regulations, the term “important to safety” is defined in 10 CFR 50.49(b) as comprising equipment that (b)(1) is safety-related, (b)(2) is not safety-related, but the failure of which could impact safety, and (b)(3) certain post-accident monitoring equipment as described in Regulatory Guide 1.97. {10 CFR 50.49}

mistake: A human action that produces an unintended result or an incorrect result. {IEEE Std. 100-2000 [136]}

partitioning: A technique for providing isolation between software components to contain and /or isolate faults. {RTCA/DO-178C [28]}

process: (1) A collection of activities performed in the software life cycle to produce a definable output or product. {RTCA/DO-178C [28]} (2) A set of interrelated activities, which transforms inputs into outputs. {IEEE Std. 100-2000 [136]}

qualification: As used in the aerospace industry, qualification is a systematic process through which software tools are verified to be suitable for their intended use(s) and to the extent possible, to be free of deficiencies that could adversely affect critical DI&C system software or result in failure to detect deficiencies in that software. However, in the commercial nuclear industry, in accordance with 10 CFR Part 21, the corresponding process is called *commercial-grade dedication*. Software tools of the type of concern, intended for use with safety-related software for nuclear power plants, but not designed or developed under a 10 CFR Part 50, Appendix B-compliant QA program (including COTS, MOTS, and GOTS software tools) must successfully complete the dedication process. However, in the nuclear regulatory context, the term “qualification” has its own special meaning, referring to the design verification processes of seismic and environmental qualification, which are applicable primarily to hardware.

reasonable: A level of confidence which is justifiable but not absolute. {EPRI 102260 [105], Section 2.2.1.2}

requirement: A provision in standards considered mandatory to meet the objectives of the standards, not to be construed as a regulatory requirement, i.e., a provision of NRC regulations.

reusable software component: Software, its supporting software life cycle data, and other supporting documentation being considered for reuse. The component designated for reuse may be any collection of software, such as libraries, operating systems, or specific system software functions. {FAA AC 20-148 [25], Appendix 1}

reverse engineering: The process of developing higher level software data from existing software data. Examples include developing source code from object code or executable object code, or developing high level requirements from low level requirements. {RTCA/DO-178C [28]}

safety-related: Pertaining to systems important to safety but that are not safety systems. {NUREG/CR-6421 [68], Section 1.3} This definition is inconsistent with the definition of the term "safety-related" in several places in NRC regulations, including 10 CFR 50.2, 10 CFR 50.49, 10 CFR 50.65, and 10 CFR 21.3 (as part of the definition of a basic component). These NRC regulations define things safety-related as a subset of things important to safety, specifically as "safety systems" are defined below (i.e., structures, systems, equipment and components that are relied upon to remain functional during and following design basis events to ensure (a) the integrity of the reactor coolant pressure boundary, (b) the capability to shut down the reactor and maintain it in a safe shutdown condition, or (c) the capability to prevent or mitigate the consequences of accidents that could results in potential offsite exposures) with the exception that the offsite exposure-related regulations have been updated to include 10 CFR 50.34(a)(1), 10 CFR 50.67(b)(2), and specifying §100.11 of 10 CFR 100.

safety systems: Those systems that are relied upon to remain functional during and following design-basis events to ensure (a) the integrity of the reactor coolant pressure boundary, (b) the capability to shut down the reactor and maintain it in a safe shutdown condition, or (c) the capability to prevent or mitigate the consequences of accidents that could results in potential offsite exposures comparable to the 10 CFR Part 100 guidelines. {NUREG/CR-6421 [68], Section 1.3}

secure development environment: The condition of having appropriate physical, logical, and programmatic controls during the system development phases (i.e., concepts, requirements, design, implementation, testing) to ensure that unwanted, unneeded, and undocumented functionality (e.g., superfluous code) is not introduced into digital safety systems. {RG 1.152 [34]}

secure operational environment: The condition of having appropriate physical, logical, and administrative controls within a facility to ensure that the reliable operation of digital safety systems are not degraded by undesirable behavior of connected systems and events initiated by inadvertent access to the system. {RG 1.152 [34]}

software: Computer programs and, possibly, associated documentation and data pertaining to the operation of a computer system. {RTCA/DO-178C [28]}

software level: The designation that is assigned to a software component as determined by the system safety assessment process. The software level establishes the rigor necessary to demonstrate compliance with RTCA/DO-178C [28]. {RTCA/DO-178C [28]}

software level reduction: A process used to reduce the rigor necessary to demonstrate compliance with RTCA/DO-178C [28]. A lower software level places less emphasis on verification of source code, low-level requirements and software architecture. A lower software level also places less emphasis on the degree of test coverage, control of verification procedures, independence of verification process activities, overlapping verification process activities, robustness testing, and verification activities with an indirect effect on error prevention or detection. {RTCA/DO-178C [28], Section 6.0}

software life cycle: (1) An ordered collection of processes determined by an organization to be sufficient and adequate to produce a software product. (2) The period of time that begins with the decision to produce or modify a software product and ends when the product is retired from service. {RTCA/DO-178C [28]}

software tools: A computer program used in the design, development, testing, review, analysis, or maintenance of a program or its documentation. Examples include compilers, assemblers, linkers, comparators, cross-reference generators, decompilers, editors, flow charters, monitors, test case generators, integrated development environments, and timing analyzers. {IEEE Std. 7-4.3.2-2010 [1], Clause 3.1.23}

state of the practice: The most advanced documented knowledge in an engineering field in actual use on a regular basis.

(i) A study of the “state-of-the-practice” in a particular field may entail a combination of review of literature such as public consensus standards, survey of leading practitioners, interviews with the best responders, and analysis and organization of the acquired data.

(ii) “State-of-the-practice” in an engineering field excludes a process requiring exceptional artistry and craftsmanship (e.g., knowledge not transferable in a written form).

(iii) Distinction between “state-of-the-practice” and “common practice”: The latter is characterized by breadth of usage in a particular engineering field and industry. The former is distinguished through repeated usage of the most advanced knowledge – also known as “best practice” in some communities.

system: A collection of hardware and software components organized to accomplish a specific function or set of functions. {RTCA/DO-178C [28]}

task: The basic unit of work from the standpoint of a control program. {RTCA/DO-178C [28]}

tool qualification: A process necessary to obtain certification credit for a software tool within the context of a specific airborne system. {RTCA/DO-178C [28]}

traceability: An association between items, such as between process outputs, between an output and its originating process, or between a requirement and its implementation. {RTCA/DO-178C [28]}

user-modifiable software: Software intended for modification without review by the certification authority, the airframe manufacturer, or the equipment vendor, if within the modificaiton constraints established during the original certification project. {RTCA/DO-178C [28]}

validation: The process of determining that the requirements are the correct requirements and that they are complete. The system life cycle processes may use software requirements and derived requirements in system validation. {RTCA/DO-178C [28]}

verification: The evaluation of the outputs of a process to ensure correctness and consistency with respect to the inputs and standards provided to that process. {RTCA/DO-178C [28]}

1 Introduction

As part of Task 1 of Nuclear Regulatory Commission (NRC) contract NRC-HQ-13-C-04-0004, Information Systems Laboratories, Inc. (ISL) surveyed standards, regulatory guidance, review/approval practices, and other current industry practices concerning the use of software tools on which the safety assurability of a digital, safety-related system is dependent. The particular software tools addressed in this report are those used in the development of software used or to be used in digital instrumentation and control (DI&C) applications including field-programmable gate arrays (FPGAs) where those tools could in any way adversely affect that software, and/or where tools used in the verification and validation (V&V) or testing of that software, could fail to detect defects in that software. As part of this task and in conformance with the statement of work, ISL has completed the following:

- Identified and surveyed existing standards related to the use of software tools in design, code generation, V&V, automated testing, and operation in engineering systems, developed in the United States and in other countries, for the nuclear power industry and other industries.

- Established a broad set of requirements covered by all these standards as a baseline set of requirements.

- Identified and surveyed existing regulatory guidance, positions, review, and approval practice related to software tool use in design, code generation, V&V, automated testing, and operation in engineering systems, established by regulatory organizations both inside and outside the United States, for the nuclear power industry and other industries.

- Established a broad baseline of guidance and practice covered by all this regulatory guidance, and review and approval practice. This baseline of guidance and approval practice may be used to develop suitable regulatory guidance, approval practices and acceptance criteria for software tools reviewed by the NRC and for NRC oversight of the review of software tools by licensees.

Section 1 of this report presents a series of tables that provide the name, title, version, and a brief abstract of all documents surveyed. Section 1.1 provides abstracts for standards and guidebooks from the National Aeronautics and Space Administration (NASA). Section 1.2 provides abstracts for standards, guidance and practices from the Federal Aviation Administration (FAA) and the Radio Technical Commission for Aeronautics (RTCA). RTCA, Inc., is a private, not-for-profit corporation that develops consensus-based recommendations regarding communications, navigation, surveillance, and air traffic management (CNS/ATM) system issues. See www.rtca.org for further details. Section 1.3 provides abstracts for existing guidance and practices from the NRC and the commercial nuclear power industry. Section 1.4 provides abstracts for standards from the Institute of Electrical and Electronics Engineers (IEEE). Section 1.5 provides abstracts for standards from the International Electrotechnical Commission (IEC). Section 1.6 provides abstracts for standards from the International Atomic

Energy Commission (IAEA). Section 1.7 provides abstracts for guidance from the Electric Power Research Institute (EPRI). Section 1.8 provides abstracts for publications and reports from the National Institute of Standards and Technology (NIST). Section 1.9 provides abstracts for standards from the Atomic Energy of Canada, Ltd. (AECL).

Section 1 does not address the relevance of the documents to software tools. The document abstracts simply provide the reader with the general content of the documents surveyed. The relevance of the document is discussed in Section 4.

Section 2 of this report discusses system and software life cycle processes. The discussion concentrates on three major documents, IEEE Standard (Std.) 1074-2006 [4], IEEE Std. 15288-2008 [5], and IEEE Std. 12207-2008 [2]. IEEE Std. 1074-2006 [4] is endorsed by NRC Regulatory Guide (RG) 1.173, Revision 1 [6]. This document does not provide software life cycle processes but instead is a standard for developing software life cycle processes, activities, and tasks. IEEE Std. 15288-2008 [5] is a standard that establishes a common process framework for describing the life cycle of man-made systems. It defines a set of processes and associated terminology for the full system life cycle, including conception, development, production, utilization, support, and retirement. IEEE Std. 12207-2008 [2] is a standard that establishes a common framework for software life cycle processes, with well-defined terminology, that can be referenced by the software industry.

Section 3 of this report discusses computerized tools used in the development of software. These are referred to as software tools or computer-aided software engineering (CASE) tools. Section 3 also establishes a set of software tool categories. Section 3 contains tables that provide a generic list of tools and indicates the life cycle steps or phases in which they can be used. Section 3 also discusses which tools are those described above collectively as being “of the type of concern” and hence, should be subject to regulatory guidance.

Section 4 of this report discusses the content of each of the 92 documents surveyed as part of this software tool research project. Numerous documents including nuclear regulatory guidance, international standards, and guidance from other industries have been surveyed to determine what guidelines and regulations exist for the use of software tools in critical applications in the nuclear industry and other industries. Documents were surveyed to determine their relevance to software tools, including those tools that are commercial-off-the-shelf (COTS) software, government-off-the-shelf (GOTS) software, or modified-off-the-shelf (MOTS) software. Documents relevant to the software tools project are discussed in Section 4.1. Documents found not to be relevant to the software tools project are discussed in Section 4.2.

Section 5 of this report discusses baseline requirements and baseline guidance and review/approval practices. It is based on the survey of industry standards related to the use of software tools in design, code generation, V&V, automated testing, and operation in engineering systems, developed in the United States and in other countries, for the nuclear power industry and other industries.

Section 6 of this report presents a brief summary and discusses conclusions from the software tools research project. Section 7 provides a list of references.

1.1 Space Industry Documents

Nine NASA documents were surveyed. Section 1.1.1 provides information on NASA standards. Section 1.1.2 provides information on NASA documents that contain guidance or review/approval practices.

1.1.1 NASA Standards

Table 1.1 provides the document number, title, date, and an abstract summarizing the content of NASA standards.

Table 1.1 – NASA Standards

NASA-STD-2201-93, "*Software Assurance Standard*," November 1992 NASA-STD-2201-93 [7] is a high-level standard for NASA software quality assurance (SQA) that is to be applied by the provider (creator) of NASA software. This standard provides a means for ensuring high-quality software, provides a means for ensuring the software is suitable for its intended purpose, and provides a high-level SQA plan that can and should be tailored to the specific software being provided. **This standard is superseded by NASA-STD-8739.8 [8].**
NASA-STD-8739.8, "*Software Assurance Standard*," July 2004 **NASA-STD-8739.8 [8] supersedes NASA-STD-2201-93 [7].** This standard specifies the software assurance requirements for software developed or acquired and maintained by NASA and for open source software, GOTS software, MOTS software, and COTS software when included in a NASA system.
NASA-STD-8719.13B, "*Software Safety Standard*," July 2004 NASA-STD-8719.13B [9] specifies the requirements for software safety. The standard describes the activities necessary to ensure that safety is designed into the software that is acquired or developed by NASA and that safety is maintained throughout the software and system life cycle. The standard specifies the software safety activities, data, and documentation necessary for the acquisition or development of software in a safety-critical system.

1.1.2 NASA Guidance and Review/Approval Practices

Table 1.2 provides the document number, title, date, and an abstract summarizing the content of NASA documents that contain guidance and review/approval practices.

Table 1.2 – NASA Guidance and Review/Approval Practices

NASA-GB-A201, "*Software Assurance Guidebook*," September 1989 NASA-GB-A201 [10] is a lower-tier guidebook for NASA SQA intended to be tutorial rather than directive for meeting NASA-STD-2201-93 [7] requirements. The guidebook is written as a narrative rather than line-by-line specifications and provides a general discussion of the software development process including reviews, testing, and documentation of the various stages of the process.
NASA-GB-A301, "*Software Quality Assurance Audits Guidebook*," November 1990 NASA-GB-A301 [11] is a second-level guidebook that provides more detail on SQA than NASA-GB-A201 [10]. This document describes SQA audits in a way that is compatible with practices at NASA centers. An audit is a fundamental QA technique. This guidebook defines audits, describes the audit process, and provides a sample checklist that can be tailored for use in an audit. The term "audit" refers to an SQA technique that is used to examine the conformance of a development process to procedures and the conformance of products to standards.
NASA-GB-8719.13-1, "*NASA Software Safety Guidebook*," March 2004 NASA-GB-8719.13-1 [12] is a guidebook that supplements NASA-STD-8719.13B [9]. The guidebook provides an overview of general software safety and good engineering practices that contribute to software system safety. It also provides the means to scope and tailor the software safety and software engineering activities to obtain the most cost effective, best quality, and safest products. It also provides analyses, methods, and guidance that can be applied during each phase of the software life cycle and includes development approaches, safety analyses, and testing methodologies that lead to improved safety in the software product.
NASA-GB-001-94, "*Software Measurement Guidebook*," August 1995 NASA-GB-001-94 [13] is a guidebook aimed at helping organizations to begin or improve a measurement program. It contains advice for establishing and using an effective software measurement program and for understanding some of the key lessons that other organizations have learned.
NASA-GB-001-95, "*Software Process Improvement Guidebook*," January 1996 NASA-GB-001-95 [14] is a guidebook that provides experience-based guidance in implementing a software process improvement program in any NASA software development or maintenance community. The guidebook details how to define, operate, and implement a working software process improvement program.

NASA-GB-001-96, "*Software Management Guidebook*," November 1996

NASA-GB-001-96 [15] is a guidebook that defines the core products and activities required of NASA software projects. The guidebook defines life-cycle models and activity-related methods and provides specific guidance to software project managers and team leaders in selecting appropriate life cycles and methods to develop a tailored plan for a software engineering project.

1.2 Civil Aviation Industry Documents

Fourteen FAA, Certification Authorities Software Team (CAST), and RTCA documents were surveyed. Section 1.2.1 provides information on the single FAA standard surveyed that was relevant. Section 1.2.2 provides information on FAA orders, CAST papers, and RTCA documents that contain guidance or review/approval practices.

1.2.1 FAA Standards

Table 1.3 provides the document number, title, date, and an abstract summarizing the content of the single relevant FAA standard surveyed.

Table 1.3 – FAA Standards

FAA-STD-026A, "*Software Development for the National Airspace System (NAS)*," June 2001

FAA-STD-026A [16] is a standard that establishes requirements for software development associated with National Airspace System (NAS) acquisition. It represents the FAA approved tailoring of IEEE/Electronic Industries Alliance (EIA) 12207.0 [17], IEEE/EIA 12207.1 [18], IEEE/EIA 12207.2 [19], and data item descriptions (DIDs). This standard can be used for all software development projects external or internal to the FAA. The purpose of this standard is to establish and communicate requirements to be applied during the development and support of computer software.

1.2.2 FAA, RTCA and CAST Guidance and Review/Approval Practices

Table 1.4 provides the document number, title, date, and an abstract summarizing the content of FAA, RTCA, and CAST documents that contain guidance and review/approval practices.

Table 1.4 – FAA, RTCA, and CAST Guidance and Review/Approval Practices

RTCA/DO-178B, "*Software Considerations in Airborne Systems and Equipment Certification*," December 1992 RTCA/DO-178B [20] discusses aspects of airworthiness certification that pertain to the production of software for airborne systems and equipment used on aircraft or engines. The purpose of the document is to provide guidelines for the production of software for airborne systems and equipment that perform their intended function with a level of confidence in safety that complies with airworthiness requirements. These guidelines are in the form of (1) objectives for software life cycle processes, (2) activities and design considerations for achieving those objectives, and (3) evidence that indicates that the objectives have been satisfied.DOT/FAA/AR-06/54, "*Software Verification Tools Assessment Study*," June 2007
FAA Order 8110.49, "*Software Approval Guidelines*," June 2003 FAA Order 8110.49 [21] supplements RTCA/DO-178B [20] for approving software used in airborne computers. The order clarifies software review and approval processes, inspection for software conformity, determines the level of FAA involvement, discusses approval of software changes made in the field to reduce aircraft maintenance time, and discusses the qualification of software tools. **This FAA order is superseded by FAA Order 8110.49 Change 1 [22].**
FAA Order 8110.49 Change 1, "*Software Approval Guidelines*," September 2011 **FAA Order 8110.49 Change 1 [22] replaces the 2003 version of FAA Order 8110.49 [21]** and guides aircraft certification service (AIR) field offices and designated engineering representatives (DER) on how to apply RTCA/DO-178B [20] for approving software used in airborne computers. This order establishes guidelines for approving software in compliance with RTCA/DO-178B [20].
DOT/FAA/AR-06/54 [23] is a report documenting the results of research activities in the software verification tools assessment study (SVTAS). SVTAS was a research effort to investigate criteria for effectively evaluating structural coverage analysis tools for use on projects intended to comply with RTCA/DO-178B [20]. The objective for developing these criteria was to improve the accuracy and consistency of the qualification process for these tools.

Position Paper CAST-13, "*Automatic Code Generation Tools Development Assurance*," June 2002

Position Paper CAST-13 [24] clarifies RTCA/DO-178B [20], Section 12.2.1.b, as it applies to automatic code generation (ACG) tools. RTCA/DO-178B [20] proposes that the software level of a development tool should be considered at least the same as the software level of the airborne system's software application the tool is being used to develop, unless the applicant can justify a reduction in software level of the tool to the certification authority.

This paper proposes a list of candidate objectives in RTCA/DO-178B [20] that could potentially be alleviated, when applicants are qualifying ACG tools, provided the applicants supply relevant rationale and justification for each objective's alleviation. The paper also provides a road map to reduce the ACG tool's software level relative to the level of the airborne software.

FAA AC 20-148, "*Reusable Software Components*," December 2004

Advisory circular (AC) FAA AC 20-148 [25] provides one acceptable means of compliance, but not the only means, for reusable software component (RSC) developers, integrators, and applicants to gain FAA "acceptance" of a software component that may be only a part of an airborne system's software applications and intended functions and gain credit for the reuse of a software component in follow-on systems and certification projects, including "full credit" or "partial credit" for compliance to the objectives of RTCA/DO-178B [20].

Position Paper CAST-22, Revision 1, "*Reuse of Software Tool Qualification Data Across Company Boundaries (Applying the Reusable Software Component Concept to Tools*," March 2005

Position Paper CAST-22, Revision 1 [26], documents the process for reusing tool qualification data across company boundaries (i.e., to approach tools as a kind of reusable component that may be reused with minimal certification authority involvement on the subsequent projects). This paper builds upon the guidance of AC 20-148 [25] with specific focus on tool-unique aspects.

RTCA/DO-248B, "*Final Report for Considerations of DO-178B*," October 2001

RTCA/DO-248B [27] provides clarification of the guidance material in RTCA/DO-178B [20] by resolving content errors, clarifying specific sections or topics, and resolving inconsistencies between RTCA/DO-178B [20] and any other relevant civil aviation standards.

RTCA/DO-178C, "*Software Considerations in Airborne Systems and Equipment Certification*," December 2011

RTCA/DO-178C [28] is an update of RTCA/DO-178B [20] to address advances in hardware and software technology that resulted in software development methodologies and issues that were not adequately addressed in RTCA/DO-178B [20] and RTCA/DO-248B [27]. This document provides the same kind of recommendations as RTCA/DO-178B [20], but revised based on experience. This document is not yet endorsed by the FAA.

RTCA/DO-248C, "*Supporting Information for DO-178C and DO-278A*," December 2011

RTCA/DO-248C [29] provides clarification of the guidance material in RTCA/DO-178C [28] and RTCA/DO-278A [30] through a series of frequently asked questions, discussion papers and rationale.

RTCA/DO-330, "*Software Tool Qualification Considerations*," December 2011

RTCA/DO-330 [31] provides tool qualification guidance by explaining the process and objectives for qualifying tools. Software tools are widely used in multiple domains, to assist in developing, verifying, and controlling other software. A tool is defined as a computer program or a functional part thereof, used to help develop, transform, test, analyze, produce, or modify another program, its data, or its documentation.

FAA Order 1370.109, "*Software Assurance Policy*," October 2009

FAA Order 1370.109 [32] establishes a security software assurance policy for the FAA to protect the confidentiality, integrity, and availability of FAA information systems. Software assurance is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner.

FAA AC 90-111, "*Guidance for the Validation of Software Tools Used in the Development of Instrument Flight Procedures by Third Party Service Providers*," June 2011

FAA AC 90-111 [33] provides guidance material for third party instrument flight procedure (IFP) developers, who use software-based tools to develop performance-based navigation (PBN) Title 14 of the Code of Federal Regulations (14 CFR) Part 97, "Standard Instrument Procedures," to have these software tools validated by the FAA.

1.3 NRC and Commercial Nuclear Power Industry Documents

Twenty NRC documents were surveyed. Section 1.3.1 provides information on NRC documents that contain guidance or review/approval practices.

1.3.1 NRC Guidance and Review/Approval Practices

Table 1.5 provides the document number, title, date, and an abstract summarizing the content of NRC guidance and review/approval practices.

Table 1.5 – NRC Guidance and Review/Approval Practices

NRC RG 1.152, Revision 3, "*Criteria for Use of Computers in Safety Systems of Nuclear Power Plants*," July 2011 NRC Regulatory Guide (RG) 1.152, Revision 3 [34], describes a method acceptable to NRC staff for complying with regulations for "promoting high functional reliability, design quality, and a secure development and operational environment (SDOE) for the use of digital computers in the safety systems of nuclear power plants." Hardware, software, firmware, and interfaces are addressed in this regulatory guide. The regulatory position is based on the waterfall life cycle model with a discussion of each phase of this model. This regulatory guide endorses IEEE Std. 7-4.3.2-2003 [35]. Conformance with the requirements of IEEE Std. 7-4.3.2-2003 [35] is a method that the NRC staff has deemed acceptable for satisfying the NRC's regulations with respect to high functional reliability and design requirements for computers used in safety systems of nuclear power plants. However, the NRC has not endorsed Annexes B through F of IEEE Std. 7-4.3.2-2003 [35] because the guidance is insufficient or inadequate.
NRC RG 1.153, Revision 1, "*Criteria for Safety Systems*," June 1996 NRC RG 1.153, Revision 1 [36], describes a method acceptable to NRC staff for satisfying the Commission's regulations with respect to the design, reliability, qualification, and testability of the power, instrumentation, and control portions of the safety systems of nuclear power plants. This regulatory guide endorses IEEE Std. 603-1991 [37], which establishes minimum functional and design requirements for the power, instrumentation, and control portions of safety systems for nuclear power plants.

NRC RG 1.168, Revision 1, "*Verification, Validation, Reviews, and Audits for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," February 2004

NRC RG 1.168, Revision 1 [38], describes a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to quality assurance programs as applied to software V&V and with respect to conducting audits, inspections, walkthroughs, and technical and management reviews.

This regulatory guide endorses IEEE Std. 1012-1998 [39] and IEEE Std. 1028-1997 [40] which provide guidance acceptable to the NRC staff for carrying out quality assurance programs as applied to software V&V and software reviews, inspections, walkthroughs, and audits. IEEE Std. 1012-1998 [39] describes the process of software V&V, including elements of a software V&V plan, and describes a minimum set of V&V activities for software at different integrity levels.

NRC RG 1.168, Revision 2, "*Verification, Validation, Reviews, and Audits for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," DRAFT, February 2013

NRC RG 1.168, Revision 2 [41], is a proposed update to RG 1.168, Revision 1 [38]. Proposed Revision 2 of this regulatory guide updates its endorsement of IEEE Std. 1012-1998 [39] to IEEE Std. 1012-2004 [42] as a method that is acceptable to the NRC staff for meeting the requirements of 10 CFR Part 50 as they apply to verification and validation of safety system software. Proposed Revision 2 of this regulatory guide also updates its endorsement of IEEE Std. 1028-1997 [40] to IEEE Std. 1028-2008 [43] as an approach acceptable to the NRC staff for carrying out software reviews, inspections, walkthroughs, and audits.

NRC RG 1.169, "*Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," September 1997

NRC RG 1.169 [44] describes a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to the maintenance of appropriate controls and records of software development activities.

This regulatory guide endorses IEEE Std. 828-1990 [45] and IEEE Std. 1042-1987 [46], which provide guidance acceptable to the NRC staff for the activity of planning for configuration management of safety system software.

NRC RG 1.169, Revision 1, "*Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," DRAFT, February 2013

NRC RG 1.169, Revision 1 [47], is a proposed update to the original RG 1.169 [44]. Proposed Revision 1 of this regulatory guide updates its endorsement of IEEE Std. 828-1990 [45] to IEEE Std. 828-2005 [48] as a method that is acceptable to the NRC staff for meeting the requirements of 10 CFR Part 50 as they apply to the maintenance and control of appropriate records of software development activities. This proposed regulatory guide no longer endorses IEEE Std. 1042-1987 [46] because IEEE Std. 1042-1987 [46] has been withdrawn and information on software configuration management is fully covered in IEEE Std. 828-2005 [48].

NRC RG 1.170, "*Software Test Documentation for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," September 1997

NRC RG 1.170 [49] describes a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to the documentation of software testing activities.

This regulatory guide endorses IEEE Std. 829-1983 [50], which provides guidance acceptable to the NRC staff for the activity of test documentation of safety system software.

NRC RG 1.170, Revision 1, "*Test Documentation for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," DRAFT, February 2013

NRC RG 1.170, Revision 1 [51], is a proposed update to the original RG 1.170 [49]. Proposed Revision 1 of this regulatory guide updates its endorsement of IEEE Std. 829-1983 [50] to IEEE Std. 829-2008 [52] as a method that is acceptable to the NRC staff for meeting the requirements of 10 CFR Part 50 as they apply to the documentation of software testing activities.

NRC RG 1.171, "*Software Unit Testing for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," September 1997

NRC RG 1.171 [53] describes a method acceptable to the NRC staff for satisfying the Commission's regulations with respect to software unit testing activities.

This regulatory guide endorses IEEE Std. 1008-1987 [54], which provides guidance acceptable to the NRC staff for planning, preparing for, conducting, and evaluating unit testing of safety system software.

NRC RG 1.171, Revision 1, "*Software Unit Testing for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," DRAFT, February 2013

NRC RG 1.171, Revision 1 [55], is a proposed update to the original RG 1.171 [53]. Proposed Revision 1 of this regulatory guide continues to endorse IEEE Std. 1008-1987 [54] as a method that is acceptable to the NRC staff for meeting the requirements of 10 CFR Part 50 as they apply to planning, preparing, conducting, and evaluating unit testing of safety system software.

NRC RG 1.172, "*Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," September 1997

NRC RG 1.172 [56] describes a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to the development of software requirements specifications.

This regulatory guide endorses IEEE Standard 830-1993 [57], which provides guidance acceptable to the NRC staff for describing current practices for writing software requirements specifications that will exhibit characteristics important for developing safety system software consistent with the NRC staff's goals of ensuring high-integrity software in reactor safety systems.

NRC RG 1.172, Revision 1, "*Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," DRAFT, February 2013

NRC RG 1.172, Revision 1 [58], is a proposed update to the original RG 1.172 [56]. Proposed Revision 1 of this regulatory guide updates the endorsement of IEEE Std. 830-1993 [57] to IEEE Std. 830-1998 [59] even though IEEE Std. 830-1998 [59] has been superseded by IEEE Std. 29148-2011 [60]. IEEE Std. 830-1998 [59] provides guidance acceptable to the NRC staff for describing current practices for writing software requirements specifications that will exhibit characteristics important for developing safety system software consistent with the NRC staff's goals of ensuring high-integrity software in reactor safety systems.

NRC RG 1.173, "*Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," September 1997

NRC RG 1.173 [61] describes a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to software development processes.

This regulatory guide endorses IEEE Std. 1074-1995 [62], which provides guidance acceptable to the NRC staff for describing, in terms of inputs, development, verification or control processes, and outputs, a set of processes and constituent activities that are commonly accepted as composing a controlled and well-coordinated software development process.

NRC RG 1.173, Revision 1, "*Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants*," DRAFT, February 2013 NRC RG 1.173, Revision 1 [4], is a proposed update to the original RG 1.173 [61]. Proposed Revision 1 of this regulatory guide updates the endorsement of IEEE Std. 1074-1995 [62] to IEEE Std. 1074-2006 [4]. IEEE Std. 1074-2006 [4] provides guidance acceptable to the NRC staff for describing, in terms of inputs, development, verification or control processes, and outputs, a set of processes and constituent activities that are commonly accepted as composing a controlled and well-coordinated software project life cycle process.
DI&C-ISG-06, Revision 1, January 2011 DI&C-ISG-06, Revision 1 [63], provides the licensing process used by the staff for review of license amendment requests associated with digital I&C system modifications in operating plants originally licensed under 10 CFR Part 50. This document presents a detailed approach for the review process to be used by the staff for digital I&C systems, including software. Of particular interest is the approach for commercial dedication of components if critical characteristics can be verified. The approach makes extensive use of IEEE standards.
NUREG-0800 Branch Technical Position (BTP) 7-14, Revision 7, "*Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems*," March 2007 NRC BTP 7-14, Revision 7 [64], provides guidelines for evaluating software life cycle processes for digital computer I&C systems. This document is closely linked to DI&C-ISG-06 [63].
NUREG-0800 Appendix 7.1-D, "*Guidance for Evaluation of the Application of IEEE Std. 7-4.3.2*," March 2007 Appendix 7.1-D [65] discusses the implementation of IEEE Standard 7-4.3-2-2003 [35]. Note that IEEE Standard 7-4.3-2-2003 [35] has been superseded by IEEE Standard 7-4.3-2-2010 [1].
NUREG/CR-6101, "*Software Reliability and Safety in Nuclear Reactor Protection Systems*," June 1993 NUREG/CR-6101 [66] discusses the role of software in conjunction with computer hardware in safety-related systems at nuclear power plants. This document also discusses methods to ensure engineering reliability and safety in computer systems throughout the software life cycle.

NUREG/CR-6263, "*High Integrity Software for Nuclear Power Plants: Candidate Guidelines, Technical Basis, and Research Needs*," June 1995

The work documented in NUREG/CR-6263 [67] was performed in support of the NRC to examine the technical basis for candidate guidelines that could be considered in reviewing and evaluating high-integrity computer software used in the safety systems of nuclear power plants. The framework for the work consisted of the following software development and assurance activities: requirements specification; design; coding; verification and validation, including static analysis and dynamic testing; safety analysis; operation and maintenance; configuration management; quality assurance; and planning and management. Each activity (framework element) was subdivided into technical areas (framework sub-elements). The report describes the development of approximately 200 candidate guidelines that span the entire range of software life-cycle activities; the assessment of the technical basis for those candidate guidelines; and the identification, categorization and prioritization of research needs for improving the technical basis. The report has two volumes: Volume 1, Executive Summary, includes an overview of the framework and of each framework element, the complete set of candidate guidelines, the results of the assessment of the technical basis for each candidate guideline, and a discussion of research needs that support the regulatory function; Volume 2 is the main report.

NUREG/CR-6421, "*A Proposed Acceptance Process for Commercial Off-the-Shelf (COTS) Software in Reactor Applications*," March 1996

NUREG/CR-6421 [68] proposes a process for acceptance of COTS software products for use in reactor systems important to safety. An initial set of four criteria establishes COTS software product identification and its safety category. Based on safety category, three sets of additional criteria, graded in rigor, are applied to approve (or disapprove) the product. These criteria fall roughly into three areas: product assurance, verification of safety function and safety impact, and examination of usage experience of the COTS product in circumstances similar to the proposed application. A report addressing the testing of existing software is included as an appendix.

1.4 Institute of Electrical and Electronics Engineers Documents

Thirty-five IEEE documents were surveyed. Section 1.4.1 provides information on IEEE standards.

1.4.1 IEEE Standards

Table 1.6 provides the document number, title, date, and an abstract summarizing the content of the IEEE standards.

Table 1.6 – IEEE Standards

IEEE Std. 7-4.3.2-2003, "*IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*," September 2003 IEEE Std. 7-4.3.2-2003 [35], endorsed by RG 1.152, Revision 3 [34], evolved from IEEE Std. 7-4.3.2-1993 [69] and represents a continued effort to support the specification, design, and implementation of computers in safety systems of nuclear power generating stations. This standard specifies additional computer-specific requirements (incorporating hardware, software, firmware, and interfaces) to supplement the criteria and requirements of IEEE Std. 603-1998 [70]. This standard should be used in conjunction with IEEE Std. 603-1998 [70] to assure the completeness of the safety system design when a computer is to be used as a component of a safety system.
IEEE Std. 7-4.3.2-2010, "*IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*," June 2010 IEEE Std. 7-4.3.2-2010 [1] evolved from IEEE Std. 7-4.3.2-2003 [35] and represents a continued effort to support the specifications, design, and implementation of computers in safety systems of nuclear power generating stations. This standard specifies additional computer-specific requirements (incorporating hardware, software, firmware, and interfaces) to supplement the criteria and requirements of IEEE Std. 603-2009 [71]. This standard refers to other standards as appropriate, e.g., IEEE Std. 7-4.3.2-2010 [1] references IEEE Std. 12207-2008 [2] for software development and provides an approach for dedication of software tools.
IEEE Std. 279-1971, "*IEEE Standard*: *Criteria for Protection Systems for Nuclear Power Generating Stations*," June 1971 IEEE Std. 279-1971 [72] establishes the minimum requirements for the safety-related functional performance and reliability of protection systems for stationary, land-based nuclear reactors producing steam for electric power generation. **This standard has been withdrawn due to obsolescence.**
IEEE Std. 323-2003, "*IEEE Standard for Qualifying Class 1E Equipment for Nuclear Power Generating Stations*," January 2004 IEEE Std. 323-2003 [73] describes the basic requirements for qualifying Class 1E equipment and interfaces that are to be used in nuclear power generating stations. The principles, methods, and procedures described are intended to be used for qualifying equipment, maintaining and extending qualification, and updating qualification, as required, if the equipment is modified. The qualification requirements in this standard, when met, demonstrate and document the ability of equipment to perform safety function(s) under applicable service conditions including design-basis events, reducing the risk of common-cause equipment failure. This standard does not provide environmental stress levels and performance requirements.

IEEE Std. 603-1991,"*IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations*," June 1991

Endorsed by RG 1.153, Revision 1 [36], IEEE Std. 603-1991 [37] establishes minimum functional design criteria for the power, instrumentation, and control portions of nuclear power generating station safety systems. The criteria are to be applied to those systems required to protect the public health and safety by functioning to mitigate the consequences of design-basis events. **According to IEEE, this standard has been superseded by IEEE Std. 603-1998 [70] and IEEE 603-2009 [71].**

IEEE Std. 603-1998,"*IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations*," July 1998

IEEE Std. 603-1998 [70] establishes minimum functional and design criteria for the power, instrumentation, and control portions of nuclear power generating station safety systems. The criteria are to be applied to those systems required to protect the public health and safety by functioning to mitigate the consequences of design-basis events. **According to IEEE, this standard has been superseded by IEEE 603-2009 [71].**

IEEE Std. 603-2009,"*IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations*," November 2009

IEEE Std. 603-2009 [71] establishes minimum functional and design criteria for the power, instrumentation, and control portions of nuclear power generating station safety systems. The criteria are to be applied to those systems required to protect the public health and safety by functioning to mitigate the consequences of design-basis events.

IEEE Std. 828-1990, "*IEEE Standard for Software Configuration Management Plans*," September 1990

IEEE Std. 828-1990 [45] is endorsed by RG 1.169 [44], but not by proposed RG 1.169, Revision 1 [47]. This document describes the IEEE standard for a software configuration management plan (SCMP), establishes the minimum required contents of an SCMP, and defines the specific activities to be addressed and their requirements for any portion of a software product's life cycle. **This standard is superseded by IEEE Std. 828-1998 [74].**

IEEE Std. 828-1998, "*IEEE Standard for Software Configuration Management Plans*," June 1998

IEEE Std. 828-1998 [74] establishes the minimum required contents of a SCMP and defines the specific activities to be addressed and their requirements for any portion of a software product's life cycle. This revision was part of the Life Cycle Harmonization Working Group and updates the standard to conform to the requirements of IEEE/EIA Std. 12207.1-1997 [18]. **This standard is superseded by IEEE Std. 828-2005 [48].**

IEEE Std. 828-2005, "*IEEE Standard for Software Configuration Management Plans*," August 2005

IEEE Std. 828-2005 [48] is endorsed by proposed RG 1.169, Revision 1 [47]. The minimum required contents of a SCMP are established via this standard. This standard applies to the entire life cycle of critical software (e.g., where failure would impact safety or cause large financial or social losses). It also applies to noncritical software and to software already developed. The application of this standard is not restricted to any form, class, or type of software.

This revision is a minor update to IEEE Std. 828-1998 [74] and was done to ensure consistency among the software configuration management (SCM) guidance provided by this standard, IEEE/EIA 12207.1-1997 [18] and the IEEE Software Engineering Body of Knowledge (SWEBOK) project. **This standard is superseded by IEEE Std. 828-2012 [75].**

IEEE Std. 828-2012, "*IEEE Standard for Software Configuration Management Plans*," March 2012

IEEE Std. 828-2012 [75] is the active IEEE 828 standard but not yet endorsed by the NRC. This standard establishes the minimum requirements for processes for configuration management (CM) in systems and software engineering. The application of this standard applies to any form, class, or type of software or system. This revision of the standard expands the previous version to explain CM, including identifying and acquiring configuration items, controlling changes, reporting the status of configuration items, as well as software builds and release engineering. Its predecessor defined only the contents of a software configuration management plan. This standard addresses what CM activities are to be done, when they are to happen in the life cycle, and what planning and resources are required. It also describes the content areas for a CM plan. The standard supports IEEE 12207-2008 [2] and IEEE 15288-2008 [5] and adheres to the terminology in International Organization for Standardization (ISO)/IEC/IEEE Std. 24765 [76] and the information item requirements of IEEE Std. 15939 [77].

IEEE Std. 829-1983, "*IEEE Standard for Software Test Documentation*," August 1983

IEEE Std. 829-1983 [50] is endorsed by RG 1.170 [49], but not by proposed RG 1.170, Revision 1 [51]. This standard describes a set of basic test documents that are associated with the dynamic aspects of software testing (that is, the execution of procedures and code). The purpose, outline, and content of each basic document are defined. Although the documents described in the standard focus on dynamic testing, several of them may be applicable to other testing activities. Documentation on electronic media as well as paper is covered. The standard does not call for specific testing methodologies, approaches, techniques, facilities, or tools, and does not specify the documentation of their use. It also does not imply or impose specific methodologies for documentation control, configuration management, or quality assurance. **This standard is superseded by IEEE Std. 829-1998 [78].**

IEEE Std. 829-1998, "*IEEE Standard for Software Test Documentation*," September 1998

IEEE Std. 829-1998 [78] describes a set of basic software test documents. This standard specifies the form and content of individual test documents. It does not specify the required set of test documents. **This standard is superseded by IEEE Std. 829-2008 [52].**

IEEE Std. 829-2008, "*IEEE Standard for Software and System Test Documentation*," July 2008

IEEE Std. 829-2008 [52] is the active IEEE 829 standard and is endorsed by proposed RG 1.170, Revision 1 [51]. Test processes determine whether the development products of a given activity conform to the requirements of that activity and whether the system and/or software satisfies its intended use and user needs. Testing process tasks are specified for different integrity levels. These process tasks determine the appropriate breadth and depth of test documentation. The documentation elements for each type of test documentation can then be selected. The scope of testing encompasses software-based systems, computer software, hardware, and their interfaces. This standard applies to software-based systems being developed, maintained, or reused (legacy, COTS, and/or non-developmental items). The term "software" also includes firmware, microcode, and documentation. Test processes can include inspection, analysis, demonstration, verification, and validation of software and software-based system products.

IEEE Std. 830-1993, "*IEEE Recommended Practice for Software Requirements Specifications*," December 1993

IEEE Std. 830-1993 [57] is endorsed by RG 1.172 [56], but not by proposed RG 1.172, Revision 1 [58]. This standard describes the content and qualities of a good software requirements specification (SRS) and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. **This standard is superseded by IEEE Std. 830-1998 [59].**

IEEE Std. 830-1998, "*IEEE Recommended Practice for Software Requirements Specifications*," June 1998

IEEE Std. 830-1998 [59] is endorsed by proposed RG 1.172, Revision 1 [58]. The content and qualities of a good SRS are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 [18] are also provided. **This standard is superseded by IEEE Std. 29148-2011 [60].**

IEEE Std. 1008-1987, "*IEEE Standard for Software Unit Testing*," December 1986

IEEE Std. 1008-1987 [54] is endorsed by RG 1.171 [53] and proposed RG 1.171, Revision 1 [55]. This standard's primary objective is to specify a standard approach to software unit testing that can be used as a basis for sound software engineering practice. A second objective is to describe the software engineering concepts and testing assumptions on which this standard approach is based. A third objective is to provide guidance and resource information to assist with the implementation and usage of the standard unit testing approach.

IEEE Std. 1012-1998, "*IEEE Standard for Software Verification and Validation*," July 1998

IEEE Std. 1012-1998 [39] is endorsed by RG 1.168, Revision 1 [38], but not by proposed RG 1.168, Revision 2 [41]. This standard describes software V&V processes, which determine whether development products of a given activity conform to the requirements of that activity, and whether the software satisfies its intended use and user needs. This determination may include analysis, evaluation, review, inspection, assessment, and testing of software products and processes. V&V processes assess the software in the context of the system, including the operational environment, hardware, interfacing software, operators, and users. **This standard is superseded by IEEE Std. 1012-2004 [42].**

IEEE Std. 1012-2004, "*IEEE Standard for Software Verification and Validation*," April 2005

IEEE Std. 1012-2004 [42] is endorsed by proposed RG 1.168, Revision 2 [41]. This standard updates IEEE Std. 1012-1998 [39] adding guidance for COTS software, clarifies the definition of software to include firmware, and updates other concepts. **This standard is superseded by IEEE Std. 1012-2012 [79].**

IEEE Std. 1012-2012, "*IEEE Standard for System and Software Verification and Validation*," May 2012

IEEE Std. 1012-2012 [79] is the active IEEE 1012 standard; however, it is not yet endorsed by the NRC. V&V processes are used to determine whether the development products of a given activity conform to the requirements of that activity and whether the product satisfies its intended use and user needs. V&V life cycle process requirements are specified for different integrity levels. The scope of V&V processes encompasses systems, software, and hardware, and it includes their interfaces. This standard applies to systems, software, and hardware being developed, maintained, or reused (legacy, COTS, and/or non-developmental items). The term software also includes firmware and microcode, and each of the terms system, software, and hardware includes documentation. V&V processes include the analysis, evaluation, review, inspection, assessment, and testing of products.

IEEE Std. 1028-1997, "*IEEE Standard for Software Reviews*," December 1997

IEEE Std. 1028-1997 [40] is endorsed by RG 1.168, Revision 1 [38], but not by proposed RG 1.168, Revision 2 [41]. This standard defines five types of software reviews, together with procedures required for the execution of each review type. This standard is concerned only with the reviews; it does not define procedures for determining the necessity of a review, nor does it specify the disposition of the results of the review. **This standard is superseded by IEEE Std. 1028-2008 [43].**

IEEE Std. 1028-2008, "*IEEE Standard for Software Reviews*," August 2008

IEEE Std. 1028-2008 [43] is the active IEEE 1028 standard and is endorsed by proposed RG 1.168, Revision 2 [41]. Five types of software reviews and audits, together with procedures required for the execution of each type, are defined in this standard. This standard is concerned only with the reviews and audits; procedures for determining the necessity of a review or audit are not defined, and the disposition of the results of the review or audit is not specified. Types included are management reviews, technical reviews, inspections, walkthroughs, and audits.

IEEE Std. 1042-1987, "*IEEE Guide to Software Configuration Management*," September 1987

IEEE Std. 1042-1987 [46] is endorsed by RG 1.169 [44], but not by proposed RG 1.169, Revision 1 [47]. This guide describes the application of configuration management disciplines to the management of software engineering projects. **This standard has been withdrawn because it contains obsolete or erroneous information. Software configuration management is discussed in IEEE Std. 828-2005 [48].**

IEEE Std. 1074-1995, "*IEEE Standard for Developing Software Life Cycle Processes*," September 1995

IEEE Std. 1074-1995 [62] is endorsed by RG 1.173 [61], but not by proposed RG 1.173, Revision 1 [6]. This standard provides a process for creating a software life cycle process. Although this standard is directed primarily at the process architect, it is useful to any organization that is responsible for managing and performing software projects. **This standard is superseded by IEEE Std. 1074-1997 [80].**

IEEE Std. 1074-1997, "*IEEE Standard for Developing Software Life Cycle Processes*," December 1997

IEEE Std. 1074-1997 [80] provides a process for creating a software life cycle process. Although this standard is directed primarily at the process architect, it is useful to any organization that is responsible for managing and performing software projects. This standard allows for continuing harmonization with IEEE/EIA 12207.0-1996 [17] and EIA/IEEE J-STD-016-1995 [81] and their successors. **This standard is superseded by IEEE Std. 1074-2006 [4].**

IEEE Std. 1074-2006, "*IEEE Standard for Developing a Software Life Cycle Process*," July 2006

IEEE Std. 1074-2006 [4] is endorsed by proposed RG 1.173, Revision 1 [6]. This standard provides a process for creating a software project life cycle process (SPLCP). It is primarily directed at the process architect for a given software project. This standard allows for continuing harmonization with IEEE/EIA 12207.0 [17] and its successors. The standard may be used to develop the primary and supporting life cycle processes specified in IEEE/EIA 12207.0 [17].

IEEE Std. 1209-1992, "*IEEE Recommended Practice for the Evaluation and Selection of CASE Tools*," December 1992

IEEE Std. 1209-1992 [82] addresses the evaluation and selection of CASE tools supporting software engineering processes, including project management processes, pre-development processes, development processes, post-development processes, and integral processes. The evaluation and selection processes recommended are based upon the perspective of a CASE tool user. Therefore, the evaluation and selection criteria address only program characteristics visible to the user, such as program inputs and outputs, program function, user interfaces, and documented external program interfaces. **This standard has been withdrawn because it contains obsolete or erroneous information. Adoption and selection of CASE tools are discussed in IEEE Std. 14102-2010 [83], IEEE Std. 14471-2010 [84], and IEEE Std. 1175.4-2008 [85].**

IEEE Std. 1462-1998, "*Information Technology - Guideline for the Evaluation and Selection of CASE Tools*," March 1998

IEEE Std. 1462-1998 [86] is an adoption of international standard ISO/IEC 14102:1995 [87]. The international standard deals with the evaluation and selection of CASE tools, covering a partial or full portion of the software engineering life cycle. The adoption of the international standard by IEEE includes an implementation note, which explains terminology differences, identifies related IEEE standards, and provides interpretation of the international standard.

IEEE Std. 1540-2001, "*IEEE Standard for Software Life Cycle Processes - Risk Management*," March 2001

IEEE Std. 1540-2001 [88] defines a process for the management of risk in the life cycle of software. It can be added to the existing set of software life cycle processes defined by the IEEE/EIA 12207 [17, 18, 19] series of standards, or it can be used independently. **This standard is superseded by IEEE Std. 16085-2006 [89].**

IEEE Std. 12207-2008, "*Systems and Software Engineering - Software Life Cycle Processes*," February 2008

IEEE Std. 12207-2008 [2] is an adoption of ISO/IEC 12207:1995 [90] and its two amendments. This international standard establishes a common framework for software life cycle processes with well-defined terminology that can be referenced by the software industry. It applies to the acquisition of systems and software products and services, to the supply, development, operation, maintenance, and disposal of software products and the software portion of a system, whether performed internally or externally to an organization. Those aspects of system definition needed to provide the context for software products and services are included.

IEEE Std. 14102-2010, "*Information Technology – Guideline for the Evaluation and Selection of CASE Tools*," June 2010

IEEE Std. 14102-2010 [83] gives guidelines for the evaluation and selection of CASE tools, covering a partial or full portion of the software engineering life cycle. It establishes processes and activities to be applied for the evaluation of CASE tools and selecting the most appropriate CASE tools from several candidates. These processes are generic, and organizations must tailor them to meet organizational needs. The CASE tool evaluation and selection processes should be viewed in the larger context of the organization's technology adoption process.

IEEE Std. 15288-2004, "*Adoption of ISO/IEC 15288:2002 Systems Engineering - System Life Cycle Processes*," December 2004

IEEE Std. 15288-2004 [91] is an adoption of ISO/IEC 15288:2002 [92]. This standard establishes a common framework for describing the life cycle of systems created by humans. It defines a set of processes and associated terminology. These processes can be applied at any level in the hierarchy of a system's structure. Selected sets of these processes can be applied throughout the life cycle for managing and performing the stages of a system's life cycle. **This standard is superseded by IEEE Std. 15288-2008 [5].**

IEEE Std. 15288-2008, "*Systems and Software Engineering - System Life Cycle Processes*," February 2008

IEEE Std. 15288-2008 [5] supersedes IEEE Std. 15288-2004 [91]. This revised standard is a step in a harmonizing strategy to achieve a fully integrated suite of system and software life cycle processes and guidance for their application. This standard establishes a common process framework for describing the life cycle of man-made systems. It defines a set of processes and associated terminology for the full life cycle, including conception, development, production, utilization, support, and retirement. This standard also supports the definition, control, assessment, and improvement of these processes. These processes can be applied concurrently, iteratively, and recursively to a system and its elements throughout the life cycle of a system.

IEEE Std. 16085-2006, “Standard for Software Engineering - Software Life Cycle Processes - Risk Management,” December 2006 IEEE Std. 16085-2006 [89] defines a process for the management of risk in the life cycle. It can be added to the existing set of software life cycle processes defined by the ISO/IEC 12207 [90] or ISO/IEC 15288 [92] series of standards, or it can be used independently. **This standard supersedes IEEE Std. 1540-2001 [88].**
IEEE Std. 29148-2011, “IEEE Systems and Software Engineering - Life Cycle Processes -- Requirements Engineering,” December 2011 IEEE Std. 29148-2011 [60] contains provisions for the processes and products related to the engineering of requirements for systems and software products and services throughout the life cycle. It defines the construct of a good requirement, provides attributes and characteristics of requirements, and discusses the iterative and recursive application of requirements processes throughout the life cycle. This standard also provides additional guidance in the application of requirements engineering and management processes for requirements-related activities in IEEE Std. 12207-2008 [2] and IEEE Std. 15288-2008 [5]. **This standard replaces IEEE Std. 830-1998 [59], IEEE Std. 1233-1998 [93], and IEEE Std. 1362-1998 [94].**

1.5 International Electrotechnical Commission Documents

Four IEC documents were surveyed. Section 1.5.1 provides information on IEC standards.

1.5.1 IEC Standards

Table 1.7 provides the document number, title, date, and an abstract summarizing the content of the IEC standards.

Table 1.7 – IEC Standards

IEC 60880, Ed. 2.0, “*Nuclear Power Plants – Instrumentation and Control Systems Important to Safety – Software Aspects for Computer-Based Systems Performing Category A Functions*,” May 2006 IEC 60880, Ed. 2.0 [95] is related to IEC 61508 [96] and the Common Position of Nuclear Domain Regulators. IEC 60880, Ed. 2.0 [95] specifies requirements for achieving highly reliable software, covers all software life cycle activities, and, together with IEC 61508 [96], can serve as the basis for certification of safety critical software in Europe. The common position paper “Licensing of Safety Critical Software for Nuclear Reactors” gives the requirements from the perspective of European nuclear regulatory organizations.

IEC 62138 Ed. 1.0B, "*Nuclear Power Plants – Instrumentation and Control Important for Safety – Software Aspects for Computer-Based Systems Performing Category B or C Functions*," January 2004

IEC 62138 Ed. 1.0B [97] provides requirements for the software of computer-based instrumentation and control (I&C) systems performing functions of Safety Category B or C as defined by IEC 61226 [98]. This document complements IEC 60880 Ed. 1.0 [99] and IEC 60880-2 Ed. 1.0 [100], which provide requirements for the software of computer-based I&C systems performing functions of Safety Category A. This document is part of the IEC Technical Committee 45, Subcommittee A standard series that begins with IEC 61513 [101].

IEC 62340 Ed. 1.0B, "*Nuclear Power Plants – Instrumentation and Control Important for Safety – Requirements for Coping with Common Cause Failures (CCF)*," December 2007

IEC 62340 Ed. 1.0B [102] gives requirements related to the avoidance of CCF of I&C systems that perform Category A functions. The document additionally requires the implementation of independent I&C systems to overcome CCF, while the likelihood of CCF is reduced by strictly applying the overall safety principles of IEC Technical Committee 45, Subcommittee A (notably IEC 61226 [98], IEC 61513 [101], IEC 60880-2 [100] and IEC 60709 [103]). The document also gives an overview of the complete scope of requirements relevant to CCF.

IEC 61226 Ed. 3.0B, "*Nuclear Power Plants – Instrumentation and Control Important to Safety – Classification of Instrumentation and Control Functions*," July 2009

IEC 61226 Ed. 3.0B [98] responds to an International Atomic Energy Agency (IAEA) requirement (requirement 5.1 of IAEA NS-R-1 [134]) to classify nuclear power plant instrumentation and control systems according to their importance to safety. With distributed computer-based I&C systems now being used for nuclear power plant (NPP) instrumentation and control systems, the functions important to safety are distributed over several systems or subsystems. Therefore, it is the intent of this standard to (1) classify the I&C functions important to safety into categories, depending on their contribution to the prevention and mitigation of postulated initiating events, and to develop requirements that are consistent with the importance to safety of each of the categories, and (2) assign specification and design requirements to I&C systems and equipment concerned with performing the classified functions.

1.6 International Atomic Energy Agency Documents

One IAEA document was surveyed. Section 1.6.1 provides information on the IAEA standard.

1.6.1 IAEA Standards

Table 1.8 provides the document number, title, date, and an abstract summarizing the content of the IAEA standard.

Table 1.8 – IAEA Standards

IAEA NS-G-1.1, "*Software for Computer Based Systems Important to Safety in Nuclear Power Plants – Safety Guide*," November 2000 IAEA NS-G-1.1 [138] provides guidance on the collection of evidence and preparation of documentation to be used in the safety demonstration for the software for computer-based systems important to safety in nuclear power plants, for all phases of the system life cycle. Guidance on the other aspects of computer-based systems, such as those concerned with the design of the computer-based system itself and its hardware, is limited to the issues raised by the development, verification and validation of software.

1.7 Electric Power Research Institute Documents

Six EPRI documents were surveyed. Section 1.7.1 provides information on EPRI documents that contain guidance or review/approval practices.

1.7.1 EPRI Guidance and Review/Approval Practices

Table 1.9 provides the document number, title, date, and an abstract summarizing the content of EPRI guidance.

Table 1.9 – EPRI Guidance

EPRI NP 5652, "*Guideline for the Utilization of Commercial Grade Items in Nuclear Safety Related Applications (NCIG-07)*," June 1988 EPRI NP 5652 [104] contains a generic procedure for acceptance of commercial-grade items (CGIs), as well as four alternative acceptance methods to provide flexibility. These four alternative acceptance methods are (1) special tests and inspections, (2) commercial-grade surveys of vendors supplying CGIs, (3) source verification, and (4) the recording of acceptable supplier/item performance. The report describes the implementation of each method and reviews examples of situations where it could be beneficial to use more than one acceptance method.

EPRI TR-102260, "*Supplemental Guidance for the Application of EPRI Report NP-5652 on the Utilization of Commercial Grade Items*," March 1994

In 1988, EPRI published NP-5652 [104] in response to a growing industry concern over an effective methodology to ensure the proper dedication of CGIs used in safety-related applications as later discussed in 1995 in 10 CFR Part 21. Since the issuance of NP-5652 [104], several industry and NRC activities have occurred which have indicated the need for further clarification and guidance regarding the implementation of the process for dedicating CGIs for safety-related applications. The purpose of EPRI TR-102260 [105] is to provide supplemental guidance regarding the implementation of the process for dedicating CGIs for nuclear safety-related applications.

Prior to the development of viable options for the acceptance of CGIs, utilities tended to procure items intended for safety-related applications from 10 CFR Part 50, Appendix B suppliers. These items were often expensive (reflecting the cost a supplier incurs to maintain the nuclear quality assurance (QA) program) and required a large lead-time for supply. The various commercial-grade dedication options presented in EPRI Report NP-5652 [104] and this supplemental guidance document will assist utilities in reducing the costs associated with the use of CGIs in safety-related applications. They also provided a systematic process for acceptance of CGIs for use in safety-related applications that are no longer available from Appendix B suppliers or from former Appendix B suppliers who discontinued their Appendix B QA programs.

EPRI TR-106439, "*Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications*," October 1996

EPRI TR-106439 [106] is a non-proprietary topical report developed by an EPRI utility working group to recommend guidance for the dedication of commercial-grade digital equipment for use in nuclear power plant safety systems (commonly referred to as commercial-grade dedication). The guidance provided in TR-106439 [106] addresses: (1) application of the preexisting non-digital commercial-grade dedication guidance to digital systems, (2) identification of applicable codes and standards, (3) a process to perform the evaluation of commercial-grade digital equipment, and (4) acceptance criteria for using commercial-grade equipment in a safety system. This guideline will help utilities evaluate, design, and implement digital upgrades involving commercial software-based equipment. It will also prove useful in obtaining regulatory approvals for such equipment, when required.

EPRI TR-107339, "*Evaluating Commercial Digital Equipment for High Integrity Applications*," December 1997 EPRI TR-107339 [107] is a supplement to TR-106439 [106]. TR-106439 [106], the predecessor to this document, provides "what-to-do" guidance for nuclear safety-related applications. This document provides "how-to" guidance with procedures and examples. This topical report provides a "tool box" of techniques, procedures, and examples that can be applied to specific applications according to their safety significance and complexity.
EPRI Technical Report 1025243, *"Plant Engineering: Guideline for the Acceptance of Commercial-Grade Design and Analysis Computer Programs Used in Nuclear Safety-Related Applications,"* June 2012 EPRI Technical Report 1025243 [108], which does NOT meet the requirements of 10 CFR Part 50, Appendix B, 10 CFR Part 21, ANSI N45.2-1977, and/or the intent of ISO-9001 (1994), provides methodology that can be used to perform safety classification of non-process computer programs, such as design and analysis tools, that are not resident or embedded (installed as part of) plant systems, structures, and components. The report also provides guidance for using commercial-grade dedication methodology to accept commercially procured computer programs that perform a safety-related function.
EPRI TR-107330, *"Generic Requirements Specification for Qualifying a Commercially Available PLC for Safety-Related Applications in Nuclear Power Plants,"* December 1996 Commercially available programmable logic controllers (PLC) with appropriate qualification programs are well suited for upgrading I&C systems in nuclear power plants. EPRI TR-107330 [109], which was submitted to the NRC for review, describes the generic functional and qualification requirements for a PLC. Following the guidance of this specification will help the utility to successfully implement PLC-based applications.

1.8 National Institute of Standards and Technology Documents

Two NIST documents were surveyed. Section 1.8.1 provides information on NIST documents that contain guidance or review/approval practices.

1.8.1 NIST Guidance and Review/Approval Practices

Table 1.10 provides the document number, title, date, and an abstract summarizing the content of NIST guidance.

Table 1.10 – NIST Guidance

NIST SP 500-234, *"Reference Information for the Software Verification and Validation Process,"* March 1996 NIST SP 500-234 [110] addresses computing systems that may be employed in the health care environment in efforts to increase reliability of care and reduce costs. Software V&V is an aid in determining that the software requirements are implemented correctly and completely and are traceable to system requirements. It helps to ensure that those system functions controlled by software are secure, reliable, and maintainable. Software V&V is conducted throughout the planning, development and maintenance of software systems, including knowledge-based systems, and may assist in assuring appropriate reuse of software.
NIST IR 4909, *"Software Quality Assurance: Documentation and Reviews,"* September 1992 NIST IR 4909 [111] is intended to provide NRC reviewers with guidelines for evaluating software used in nuclear power plant safety systems. These evaluations are based on software documentation. Utilities submit vendor products to NRC for approval at the end of product development, and sometimes during development. NRC may conduct audits during the development process. NRC reviewers evaluate these products based on the documentation, reviews, and reports supplied by the vendor. They check for compliance with product plans, safety requirements, and with recommended standards and guidelines for software development and assurance. NRC may be asked to review other types of software products, which may be only one component of a system, or NRC may examine an entire system, consisting of software and other components. The recommendations in this report apply only to the software documentation aspects of the safety evaluation.

1.9 Atomic Energy of Canada, Ltd. Documents

One AECL document was surveyed. Section 1.9.1 provides information on the AECL standard.

1.9.1 AECL Standards

Table 1.11 provides the document number, title, date, and an abstract summarizing the content of the AECL standard.

Table 1.11 – AECL Standards

CE-1001-STD, Revision 2, *"Standard for Software Engineering of Safety Critical Software,"* December 1999 CE-1001-STD, Revision 2 [139] is a standard jointly prepared by Ontario Power Generation, Nuclear and Atomic Energy of Canada Limited to specify software engineering requirements for safety-critical software used in real-time protective, control, and monitoring systems. Software engineering includes requirements definition, design, code implementation, review, systematic verification, testing, hazards analysis, reliability qualification, planning, configuration management and training processes. This standard defines a minimum set of software engineering processes to be followed in creating and revising the software, a minimum set of outputs to be produced by the processes, and requirements for the content of the outputs.

Intentionally Blank

2 Life Cycle Process Pertaining to Software Tools

The modern approach to software engineering including application of software tools revolves around the implementation of suitable software life cycle processes. As such, any regulatory guidance must consider software life cycle processes. Many different types of life cycle models have been proposed over the years including waterfall, spiral, and others. Institute of Electrical and Electronics Engineers (IEEE) Standard (Std.) 1074-2006 [4] provides examples of three software project life cycle models. One is a "requirements-defining software project life cycle model" which would apply to a project whose only purpose is to deliver software requirements and not a finished software project. Another is a "system retirement software project life cycle model" whose purpose is simply to retire the software system. The last is a "development and delivery software project life cycle model" where the project's mission is to plan, develop, and deliver a software system. The activities in each of these life cycle models vary and are tailored to the specific project. It is not the intention of this report to advocate any particular life cycle model or set of activities or processes. Rather, the report addresses each phase of life cycle processes that are typically used in software engineering. This allows determination of whether software tool regulation or guidance is warranted or not warranted in certain software life cycle phases. Regulation and guidance are only warranted when the tools have an impact on the final plant safety software either by introduction of defects or by a failure to detect defects.

Three principal IEEE standards apply to system and software life cycles. The first, endorsed by proposed Regulatory Guide (RG) 1.173, Revision 1 [6], is IEEE Std. 1074-2006 [4]. This document is a standard for developing a software project life cycle process (SPLCP) and can be used where software is the total system or where software is part of a larger system. IEEE Std. 1074-2006 [4] is a revision of IEEE Std. 1074-1997 [80]. The revision allows for continuing harmonization with IEEE Std. 12207-2008 [2] and its successors. Completing the activities and tasks of IEEE Std. 1074-2006 [4] completes a single process of IEEE Std. 12207-2008 [2], i.e., Clause 6.2.1. Therefore, IEEE Std. 1074-2006 [4] has the narrowest coverage of the three life cycle standards.

The second standard that applies to system and software life cycles is IEEE Std. 15288-2008 [5], which was written concurrently with IEEE Std. 12207-2008 [2]. This document establishes a common process framework for describing the life cycle of man-made systems. IEEE Std. 15288-2008 [5] does not limit the application to software life cycles and instead opens the application to a broader system life cycle which may be configured with one or more of the following: hardware, software, data, humans, processes (e.g., processes for providing service to users), procedures (e.g., operator instructions), facilities, materials, and naturally occurring entities. Of the three standards, IEEE Std. 15288-2008 [5] has the broadest range of coverage.

The third standard that applies to system and software life cycles is IEEE Std. 12207-2008 [2]. This document establishes a framework for software life cycle processes. IEEE Std. 12207-2008 [2] applies to the acquisition of systems and software products and services; to the supply, development, operation, maintenance, and disposal of software products; and the software portion of a system; whether performed internally or externally to an organization. IEEE Std. 12207-2008 [2] integrates ISO/IEC 12207:1995 [90] with its two amendments and was

coordinated with the parallel revision of International Organization of Standardization (ISO)/International Electrotechnical Commission (IEC) 15288:2002 [92] to align structure, terms, and corresponding organizational and project processes. This standard does not have the broad coverage of IEEE Std. 15288-2008 [5] but has more extensive coverage than the narrow focus of IEEE Std. 1074-2006 [4].

Each of these three standards is discussed in the following sections.

2.1 IEEE Std. 1074-2006 - Standard for Developing Software Life Cycle Processes

IEEE Std. 1074-2006 [4] is a standard that defines the process by which an SPLCP is developed. This methodology begins with a selection of a software project life cycle model (SPLCM) based on the organization's mission, vision, goals, and resources. It continues through the definition of the software project life cycle (SPLC) using the selected model, application of the activities in Annex A of the standard that are mapped within the selected SPLCM, and selection of the portion of the software life cycle that is relevant to the project. The methodology concludes with the augmentation of the software life cycle with organizational process assets (OPAs) to create the SPLCP.

The activities listed in this standard are not executable processes. They are components of processes and are not intended to stand alone. The activities are generic and do not imply a sequential order. This standard provides activities to be addressed in a software life cycle and allows the user flexibility in the manner in which activities are mapped onto the selected model and software project life cycle.

This standard is written to provide direction and guidance to process architects and other project personnel concerned with the implementation or performance of project processes. Clause 4 of this standard describes the way in which implementation of this standard is to be approached. First, the process architect assures that requirements for an SPLCP are established. Relevant stakeholders are identified, and their needs and expectations are transformed into a feasible set of requirements that are acceptable to the stakeholders. Then the process architect shall select the SPLCM to which the activities will be mapped. Then the process architect maps the required activities from Annex A onto the SPLCM. The SPLCM or the project itself could benefit from or require activities that are not included in Annex A. Mapping is performed by placing the activities in an executable sequence, developing and justifying a list of activities not used, and finally verifying the map. This produces the SPLC. The next step is to establish the SPLCP by applying available OPAs to the SPLC activities. The final step is to validate the SPLCP against the set of requirements established in the first step.

Annex A of this standard contains five major sections divided into 17 activity groups. The five sections and 17 activity groups contain 69 activities that cover the entire software life cycle from concept exploration through eventual retirement of the software system. Each of the 69 activities contains a required set of inputs, the source activity from which the input is obtained, a description of the activity, a defined set of outputs, and the destination activity that uses that output. Table 2.1 provides a summary of the activities discussed in IEEE Std. 1074-2006 [4].

Table 2.1 – IEEE Standard 1074-2006 Life Cycle Activities

Section	Activity Group	Activity
A.1 - Project Management	A.1.1 - Project Initiation Activity Group	A.1.1.1 - Develop SPLCP
		A.1.1.2 - Perform Estimations
		A.1.1.3 - Allocate Project Resources
		A.1.1.4 - Define Metrics
		A.1.1.5 - Determine Security Objectives
	A.1.2 - Project Planning Activity Group	A.1.2.1 - Plan Evaluations
		A.1.2.2 - Plan Configuration Management
		A.1.2.3 - Plan System Transition
		A.1.2.4 - Plan Installation
		A.1.2.5 - Plan Documentation
		A.1.2.6 - Plan Training
		A.1.2.7 - Plan Project Management
		A.1.2.8 - Plan Integration
		A.1.2.9 - Plan Release Management
	A.1.3 - Project Monitoring and Control Activity Group	A.1.3.1 - Manage Risks
		A.1.3.2 - Manage the Project
		A.1.3.3 - Identify SPLCP Improvement Needs
		A.1.3.4 - Retain Records
		A.1.3.5 - Collect and Analyze Data
		A.1.3.6 - Close Project
A.2 - Pre-Development	A.2.1 - Concept Exploration Activity Group	A.2.1.1 - Identify Ideas or Needs
		A.2.1.2 - Formulate Potential Approaches
		A.2.1.3 - Conduct Feasibility Studies
		A.2.1.4 - Refine and Finalize the Idea or Need
	A.2.2 - System Allocation Activity Group	A.2.2.1 - Analyze System Functions
		A.2.2.2 - Develop System Architecture
		A.2.2.3 - Allocate System Requirements
	A.2.3 - Software Importation Activity Group	A.2.3.1 - Identify Imported Software Requirements
		A.2.3.2 - Evaluate Software Import Sources
		A.2.3.3 - Define Software Import Method
		A.2.3.4 - Import Software
A.3 - Development	A.3.1 - Software Requirements Activity Group	A.3.1.1 - Define and Develop Software Requirements
		A.3.1.2 - Define Interface Requirements
		A.3.1.3 - Prioritize and Integrate Software Requirements
	A.3.2 - Design Activity Group	A.3.2.1 - Perform Architectural Design
		A.3.2.2 - Design Database
		A.3.2.3 - Design Interfaces
		A.3.2.4 - Perform Detailed Design
	A.3.3 - Implementation Activity Group	A.3.3.1 - Create Executable Code
		A.3.3.2 - Create Operating Documentation
		A.3.3.3 - Perform Integration
		A.3.3.4 - Manage Software Release
A.4 - Post-Development	A.4.1 - Installation Activity Group	A.4.1.1 - Distribute Software
		A.4.1.2 - Install Software
		A.4.1.3 - Accept Software in Operational Environment
	A.4.2 - Operation and Support Activity Group	A.4.2.1 - Operate the System
		A.4.2.2 - Provide Technical Assistance and Consulting
		A.4.2.3 - Maintain Support Request Log
	A.4.3 - Maintenance Activity Group	A.4.3.1 - Identify Software Improvement Needs
		A.4.3.2 - Implement Problem Reporting Method

Section	Activity Group	Activity
		A.4.3.3 - Reapply SPLCP
	A.4.4 - Retirement Activity Group	A.4.4.1 - Notify User
		A.4.4.2 - Conduct Parallel Operations
		A.4.4.3 - Retire System
A.5 - Support	A.5.1 - Evaluation Activity Group	A.5.1.1 - Conduct Reviews
		A.5.1.2 - Create Traceability Matrix
		A.5.1.3 - Conduct Audits
		A.5.1.4 - Develop Test Procedures
		A.5.1.5 - Create Test Data
		A.5.1.6 - Execute Tests
		A.5.1.7 - Report Evaluation Results
		A.5.1.8 - Confirm Security Accreditation
	A.5.2 - Software Configuration Management Activity Group	A.5.2.1 - Develop Configuration Identification
		A.5.2.2 - Perform Configuration Control
		A.5.2.3 - Perform Status Accounting
	A.5.3 - Documentation Development Activity Group	A.5.3.1 - Implement Documentation
		A.5.3.2 - Produce and Distribute Documentation
	A.5.4 - Training Activity Group	A.5.4.1 - Develop Training Materials
		A.5.4.2 - Validate the Training Program
		A.5.4.3 - Implement the Training Program

2.2 IEEE Std. 15288-2008 - System Life Cycle Processes

IEEE Std. 15288-2008 [5] establishes a common process framework for describing the life cycle of man-made systems. It defines a set of processes and associated terminology for the full life cycle, including conception, development, production, utilization, support, and retirement. This standard also supports the definition, control, assessment, and improvement of these processes. These processes can be applied concurrently, iteratively, and recursively to a system and its elements throughout the life cycle of a system.

This latest revision aligns with the revision to ISO/IEC 12207 [90] within the context of system life cycle processes and applies guidelines for process definition to support consistency; to improve usability; and to align structure, terms, and corresponding organizational and project processes. The system life cycle processes, presented in IEEE Std. 12207-2008 [2], differ from those presented in Clause 6 of IEEE Std. 15288-2008 [5] by establishing a higher-level system perspective. For example, with respect to the activity of preparing for the acquisition, IEEE 15288-2008 [5] in Section 6.1.1.3, “Acquisition Process, Activities and Tasks,” requires two tasks to be completed. One is to “Establish a strategy for how the acquisition will be conducted.” The second is to “Prepare a request for the supply of a product or service that includes the definition of requirements.” IEEE 12207-2008 [2], Section 6.1.1.3, expands these two tasks into 13 tasks. Similar generalization of the requirements presented in IEEE Std. 12207-2008 [2] can be found in the remainder of IEEE Std. 15288-2008 [5]. Another significant difference between the two documents is that the software-specific guidance presented in IEEE Std. 12207-2008 [2], Clause 7, is loosely incorporated into the system life cycle steps in IEEE Std. 15288-2008 [5]. A note in IEEE Std. 12207-2008 [2] states that a later revision of IEEE Std. 15288 will resolve this difference.

2.3 IEEE Std. 12207-2008 - Software Life Cycle Processes

IEEE Std. 12207-2008 [2] presents a set of system life cycle processes that provides a framework for describing the life cycle of software engineering. The main purpose of this standard is to provide a defined set of processes to facilitate communication among acquirers, suppliers, and other stakeholders in the life cycle of the system. This standard has a strong relationship with IEEE Std. 15288-2008 [5] and may be used in conjunction with it. The processes of IEEE Std. 12207-2008 [2] directly correspond to processes of IEEE Std. 15288-2008 [5] but with some specialization for software products and services. In the case where the system has important non-software elements, an organization may wish to apply IEEE Std. 15288-2008 [5] to provide the appropriate life cycle processes. For each software element of the system, the organization would apply the software implementation process of IEEE Std. 12207-2008 [2] to create the software element. In the case where the non-software portions of the system are minimal, an organization may apply IEEE Std. 12207-2008 [2] without reference to IEEE Std. 15288-2008 [5] because IEEE Std. 12207-2008 [2] contains the additional system-level process to provide the minimum appropriate system context for the software.

The software life cycle steps presented in IEEE Std. 12207-2008 [2] are subdivided into system-context and software-specific process categories, which are presented in Tables 2.2 and 2.3, respectively. Table 2.2 lists the software-related, system-context processes from IEEE Std. 12207-2008 [2]. The first of the system-context life cycle processes is Agreement Processes, which define the activities necessary to establish agreements or contracts between two organizations. The second system-context life cycle process is Organizational Project-Enabling Processes, which manage an organization's capability to acquire and supply products or services through the initiation, support, and control of projects. The third system-context life cycle process is Project Processes, which addresses project planning, assessment, and control. Project processes are subdivided into Project Management Processes and Project Support Processes. Project Management Processes are used to establish and evolve project plans, assess actual achievement and progress against the plans and to control execution of the project from conception through fulfillment. Project Support Processes provide a specific focused set of tasks for performing a specialized management objective. The fourth system-context life cycle process is Technical Processes, which are used to define the requirements of a system, transform the requirements into an effective product, use the product, provide the required services, sustain the provision of those services, and dispose of the product when it is retired from service.

Table 2.3 lists the software-specific processes from IEEE Std. 12207-2008 [2]. In this table, the first two processes are the Software Implementation Processes and Software Support Processes. Software Implementation Processes are used to produce a specified system element (software item) implemented in software. Those processes transform specified behavior, interfaces, and implementation constraints into implementation actions resulting in a system element that satisfies the requirements derived from the system requirements. The Software Support Processes provide a specific focused set of activities for performing a specialized software process. These supporting processes assist the Software Implementation Processes as an integral part with a distinct purpose, contributing to the success and quality of

the software project. A third software-specific process from IEEE Std. 12207-2008 [2] is the Software Reuse Processes that support an organization's ability to reuse software items across project boundaries.

The processes that are judged by ISL to have a role in developing regulatory guidance for software tools are shown in bold in Tables 2.2 and 2.3. This judgment is described below and is based on the activities, tasks, and products associated with the processes and whether a tool used during a specific process impacts the final nuclear plant software. It is recognized that not all of the various processes that are listed in Tables 2.2 and 2.3 are addressed in the standards, guidance, and review and approval practices surveyed as part of this report. This is not surprising given the breadth of industries and even countries that are represented in these standards. **ISL concludes that software tools related to the following processes and activities could impact the final nuclear plant software and should be subject to regulatory guidance.**

- Processes that assure that products and services meet organizational quality objectives thereby minimizing the potential introduction of errors into the final safety-related plant software (Quality Management Process and Software Quality Assurance Process)

- Processes that establish and maintain the integrity of software development products and data thereby preventing unauthorized or unintended code from being introduced into the final safety-related plant software (Configuration Management Process and Software Configuration Management Process)

- Processes that prepare, collect, analyze, report and control application data relating to the products developed to objectively demonstrate the quality of the products thereby minimizing the potential introduction of errors into the final safety-related plant software (Measurement Process)

- Processes that assess progress against objectives; determine compliance with requirements; and identify, analyze, manage, and resolve discovered problems to ensure a high-integrity software product free of known faults (Software Review, Audit, and Problem Resolution Processes)

- Processes used to capture system and/or software requirement specifications and to transform requirement specifications into final system code and data thereby ensuring that requirement specifications are accurately represented in the final safety-related plant software (System Technical Processes and Software Implementation Processes)

- Processes supporting the installation, operation, and maintenance of software within the system thereby ensuring that the safety-related plant software is not used in a manner for which it was not designed (Software Technical Processes)

- Processes supporting the performance of verification, validation, and testing which confirm the safety-related software reflects the specified requirements (Software

Qualification Testing Process, Software Verification Process, and Software Validation Process)

- Processes supporting the development and maintenance of domain models, architecture, and assets and that plan, manage, and control the life of reusable assets thereby ensuring valid application and use of reusable software components (Software Reuse Processes)

The remaining processes in Tables 2.2 and 2.3 cannot introduce errors or fail to detect errors in the final safety-related plant software and should be outside the scope of regulatory guidance. Specifically, the Acquisition Process and the Supply Process consist of primarily contractual and/or management-related activities and tasks. Software tools used in these processes are considered off-line tools and cannot introduce errors or fail to detect errors in the final safety-related plant software. Therefore, software tools used in these processes should not be subject to regulatory guidance.

The majority of the Organizational Project-Enabling Processes including Life Cycle Model Management Process, Infrastructure Management Process, Project Portfolio Management Process, and Human Resource Management Processes consist of primarily management or personnel-related activities and tasks. Software tools used in these processes could impact project productivity and effectiveness but cannot introduce errors or fail to detect errors in the final safety-related plant software. Therefore, software tools used in these processes should not be subject to regulatory guidance.

The two Project Management Processes, Project Planning Process and Project Assessment and Control Process, consist primarily of scheduler and management-related activities and tasks. Software tools used in these processes could impact project productivity and effectiveness but cannot introduce errors or fail to detect errors in the final safety-related plant software. Therefore, software tools used in these processes should not be subject to regulatory guidance.

Two Project Support Processes, Decision Management Process and Information Management Process, consist primarily of management-related activities and tasks. Software tools used in these processes cannot introduce errors or fail to detect errors in the final plant safety-related software. Therefore, software tools used in these processes should not be subject to regulatory guidance.

The final system-context Software Technical Process, Software Disposal Process, also cannot impact the development of safety-related plant software and software tools used in this process should not be subject to regulatory guidance.

Software tools used in one software-specific process in Table 2.3, Software Documentation Management Process, should not be subject to regulatory guidance. Software tools used to automate the creation, maintenance, and use of software documentation do not have a direct effect on the final software outputs. Therefore, they do not need regulatory guidance unless they are also used during the code construction process.

All other system-context processes in Table 2.2 and all software-specific processes in Table 2.3 consist of activities and tasks where software tools could impact the final safety-related plant software, either through specification, development, implementation, configuration management, quality assurance, verification and validation, qualification, or reuse. Therefore, software tools used in these processes should be subject to regulatory guidance.

Table 2.2 – IEEE Std. 12207-2008 System-Context Processes

Process	Sub-process	Purpose
Agreement Processes	Acquisition Process	Obtain the product and/or service that satisfies the need expressed by the acquirer. The process begins with the identification of customer needs and ends with the acceptance of the product and/or service needed by the acquirer.
	Supply Process	Provide a product or service to the acquirer that meets the agreed requirements.
Organizational Project-Enabling Processes	Life Cycle Model Management Process	Define, maintain, and assure availability of policies, life cycle processes, life cycle models, and procedures for use by the organization. Provides life cycle policies, processes, and procedures that are consistent with the organization's objectives, that are defined, adapted, improved and maintained to support individual project needs within the context of the organization, and that are capable of being applied using effective, proven methods and tools.
	Infrastructure Management Process	Provide the enabling infrastructure and services to projects to support organization and project objectives throughout the life cycle. Define, provide, and maintain the facilities, tools, and communications and information technology assets needed for the organization's business.
	Project Portfolio Management Process	Initiate and sustain necessary, sufficient, and suitable projects to meet strategic objectives of the organization. Commit the investment of adequate organization funding and resources. Sanctions the authorities needed to establish selected projects. Performs continued qualification of projects to confirm they justify continued investment.
	Human Resource Management Processes	Provide the organization with necessary human resources and to maintain their competencies, consistent with business needs. Assure the providing of a supply of skilled and experienced personnel qualified to perform life cycle processes to achieve organization, project, and customer objectives.
	Quality Management Process	**Assure that products, services, and implementation of life cycle processes meet organizational quality objectives and achieve customer satisfaction.**
Project Management Processes	Project Planning Process	Produce and communicate effective and workable project plans. Determine the scope of the project management and technical activities; identify process outputs, project tasks, and deliverables; establish schedules and required resources to accomplish project tasks.
	Project Assessment and Control Process	Determine the status of the project and ensure that the project performs according to plans and schedules, and within projected budgets, and that it satisfies technical objectives. Redirect project activities, as appropriate, to correct identified deviations from other processes.
Project Support Processes	Decision Management Process	Select the most beneficial course of project action where alternatives exist. Respond to requests for a decision to reach specified, desirable or optimized outcomes. Analyze alternative actions and select a course of action.
	Risk Management Process	Identify, analyze, treat, and monitor the risks continuously throughout the life cycle of a system or software product or service.
	Configuration Management Process	**Establish and maintain the integrity of all identified outputs of a project or process and make them available to concerned parties.**
	Information Management Process	Provide relevant, timely, complete, valid and, if required, confidential information to designated parties during and, as appropriate, after the system life cycle. Generate, collect, transform, retain, retrieve, disseminate and dispose of information.

Process	Sub-process	Purpose
	Measurement Process	**Collect, analyze, and report data relating to the products developed and processes implemented within the organizational unit, to support effective management of the processes, and to objectively demonstrate the quality of the products.**
System Technical Processes	**Stakeholder Requirements Definition Process**	**Define the requirements for a system that can provide the services needed by users and other stakeholders in a defined environment. Identify stakeholders throughout the life cycle and their needs and desires. Analyze and transform needs and desires into a common set of stakeholder requirements.**
	System Requirements Analysis Process	**Transform the defined stakeholder requirements into a set of desired system technical requirements that will guide the design of the system.**
	System Architectural Design Process	**Identify which system requirements should be allocated to which elements of the system.**
	Implementation Process	**Realize a specified system element.**
	System Integration Process	**Integrate the system elements (including software items, hardware items, manual operations, and other systems, as necessary) to produce a complete system that will satisfy the system design and the customers' expectations expressed in the system requirements.**
	System Qualification Testing Process	**Ensure that the implementation of each system requirement is tested for compliance and that the system is ready for delivery.**
Software Technical Processes	**Software Installation Process**	**Install the software product that meets the agreed requirements in the target environment.**
	Software Acceptance Support Process	**Assist the acquirer to achieve confidence that the product meets requirements.**
	Software Operation Process	**Operate the software product in its intended environment and to provide support to the customers of the software product.**
	Software Maintenance Process	**Provide cost-effective support to a delivered software product.**
	Software Disposal Process	Purpose is to end the existence of a system's software entity.

Table 2.3 – IEEE Std. 12207-2008 Software-Specific Processes

Process	Sub-process	Purpose
Software Implementation Processes	**Software Implementation Process**	**Produce a specified system element implemented as a software product or service. Transform specified behavior, interfaces and implementation constraints into actions that creates a software item.**
	Software Requirements Analysis Process	**Establish the requirements of the software elements of the system.**
	Software Architectural Design Process	**Provide a design for the software that implements the requirements and can be verified against the requirements.**
	Software Detailed Design Process	**Provide a design for the software that implements and can be verified against the requirements and the software architecture and is sufficiently detailed to permit coding and testing.**
	Software Construction Process	**Produce executable software units that properly reflect the software design.**
	Software Integration Process	**Combine the software units and software components, producing integrated software items, consistent with the software design, that demonstrate that the functional and non-functional software requirements are satisfied on an equivalent or complete operational platform.**
	Software Qualification Testing Process	**Confirm that the integrated software product meets its defined requirements.**
Software Support Processes	Software Documentation Management Process	Develop and maintain the recorded software information produced by a process.
	Software Configuration Management Process	**Establish and maintain the integrity of the software items of a process or project and make them available to concerned parties.**
	Software Quality Assurance Process	**Provide assurance that work products and processes comply with predefined provisions and plans.**
	Software Verification Process	**Confirm that each software work product and/or service of a process or project properly reflects the specified requirements.**
	Software Validation Process	**Confirm that the requirements for a specific intended use of the software work product are fulfilled.**
	Software Review Process	**Maintain a common understanding with the stakeholders of the progress against the objectives of the agreement and what should be done to help ensure development of a product that satisfies the stakeholders. Software reviews are at both project management and technical levels and are held throughout the life of the project.**
	Software Audit Process	**Independently determine compliance of selected products and processes with the requirements, plans and agreement, as appropriate.**
	Software Problem Resolution Process	**Ensure that all discovered problems are identified, analyzed, managed and controlled to resolution.**
Software Reuse Processes	**Domain Engineering Process**	**Develop and maintain domain models, domain architectures and assets for the domain.**
	Reuse Asset Management Process	**Manage the life of reusable assets from conception to retirement.**
	Reuse Program Management Process	**Plan, establish, manage, control, and monitor an organization's reuse program and to systematically exploit reuse opportunities.**

2.4 Comparison of IEEE 12207 and IEEE 1074 Software Life Cycle Processes

IEEE Std. 1074-2006 [4] states that IEEE/ Electronic Industries Alliance (EIA) 12207.0 [17] places requirements upon the characteristics of a designated set of life cycle processes, but does not specify the detailed implementation of those processes. IEEE Std. 1074-2006 [4] also states that it complements the application of IEEE/EIA 12207.0 [17] and is written for the project architect in developing project-specific processes complying with the requirements of IEEE/EIA 12207.0 [17]. IEEE Std. 12207-2008 [2] provides a comprehensive list of project-specific processes, activities, and tasks within those activities. IEEE Std. 12207-2008 [2] also includes an annex that describes how the processes can be tailored to suit the specific project.

Proposed RG 1.173, Revision 1 [6], which endorses IEEE Std. 1074-2006 [4], states that IEEE Std. 1074-2006 [4] is an organizing standard that ensures that activities deemed important to software quality are performed and are related properly to one another; however, it does not provide detailed information on the implementation of specific life-cycle activities. IEEE Std. 1074-2006 [4] states that IEEE/EIA 12207.0 [17] also does not specify the detailed implementation of its processes. Since IEEE Std. 12207-2008 [2] contains a defined set of software life cycle processes and IEEE Std. 1074-2006 [4] is only a standard for developing software life cycle processes, the processes in IEEE Std. 12207-2008 [2] will be used on a go-forward basis for evaluating whether software tools used during execution of a process requires regulation or guidance.

Both IEEE Std. 1074-2006 [4] and IEEE Std. 12207-2008 [2] discuss activities involved in a software project. Following either standard will result in all required activities in a software life cycle being performed. However, each standard approaches the software life cycle from a slightly different perspective; therefore, the processes and activities associated with each standard cannot be directly correlated to one another.

Intentionally Blank

3 List of Software Tools and Software Tools Categories

According to the Multinational Design Evaluation Program (MDEP) Digital Instrumentation and Control Working Group (DICWG) Generic Common Position DICWG No 2 [3], the use of appropriate software tools can increase the integrity of the instrumentation and control (I&C) development process and the software reliability by reducing the risk of introducing faults during the process. The use of tools can also have economic benefits by reducing the time and effort required to produce systems, components, and software. Tools can be used to automatically check for adherence to standards, generate records and documentation, and support configuration management. Tools are primarily used to capture system and software requirements; transform requirements into final system code and data; perform verification, validation, and testing; prepare and control application data; and manage and control processes. Tools can also reduce the effort required for testing and maintaining logs. Tools are most powerful when they are clearly defined and designed to work co-operatively with each other.

Computerized tools used in the development of software are referred to as software tools or computer-aided software engineering (CASE) tools. These tools can be used at almost every phase of the software life cycle and assist in almost every task. The tools may be individual programs or they may be combined into a single program that performs multiple related tasks such as an integrated development environment (IDE).

Tables 3.1 through 3.9 provide a generic list of CASE tools. Since the science of software engineering changes so rapidly, it is not meant to be an all-inclusive listing but rather a guideline to help classify software tools that may be used in development of safety-significant software. These tables were generated by Information Systems Laboratories, Inc. (ISL) based on the CASE tool characteristics and attributes discussed in IEEE Std. 14102-2010 [83] which is an update of IEEE Std. 1462-1998 [86]. Additional information from IEEE Std. 12207-2008 [2] and IEEE Std. 1074-1995 [62] was used by ISL to determine where the tools would be used in the life cycle process. ISL has generated nine tool categories based on ten life-cycle process-related CASE tool characteristics identified in IEEE Std. 14102-2010 [83]. Tools associated with verification and validation processes have been combined into a single category since these processes are closely related. Other IEEE Std. 12207-2008 [2] life cycle processes not addressed by these nine tool categories are absent either because CASE tools do not typically provide support for those processes, or because the process and/or the CASE support for it are not stable at this time. Each of the nine tables are organized according to life cycle process, sub-process, or activity and subdivided into specific tool types.

Each of the nine tool categories are discussed in Sections 3.1 through 3.9. In these sections, ISL states a position as to whether regulatory guidance is appropriate for that category of tool based on the potential impact of the tool on the final safety-related software. The impact of the tool on the safety-related software is purely ISL judgment based on descriptions of the tools in IEEE Std. 14102-2010 [83] and the IEEE Std. 12207-2008 life cycle process, sub-process, or activity in which the tool is used.

3.1 Management Support Tools

Management support tools are software tools that can be found in almost any type of business. They are not specific to software development. These tools are generally classified as enterprise resource planning tools and do not directly affect the final I&C software. **Because these tools function at the project level and do not directly affect the final software, ISL concludes that no regulatory guidance is necessary for these tools.**

Table 3.1 – Software Tools – Management Support Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Project Management	Project Planning	Cost and Schedule Estimating	Tools that support the ability to estimate cost, schedule, and other project parameters based upon organizational inputs.
	Project Assessment and Control	Project Tracking	Tools that support user entry of project activity data, including automated data gathering.
	Project Assessment and Control	Project Status Analysis and Reporting	Tools that support analysis of project activities based on the data tracked and provide status reports and projections in user definable formats.
	Project Planning	Managing Processes	Tools that support the management of processes.

3.2 Implementation Tools

Implementation tools are used in the design process to support elicitation, analysis, specification, verification, validation, and management of software requirements. **Because the output of some of these tools can be used directly in the generation of source code and databases, ISL concludes that implementation tools should have regulatory guidance.** The extent of the review should be based on the effect that the tool can have on the final software.

Table 3.2 – Software Tools – Implementation Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
System Technical	Stakeholder Requirements Definition	Requirement Elicitation Support	Tools that support the conduct of elicitation interviews, workshops and other requirements information gathering activities.
Software Implementation	Software Requirements Analysis	Requirement Analysis Support	Tools that support the review and analysis of requirements, problem solving activities, and prioritizing of requirements.
	Software Requirements Analysis	Requirement Specification Support	Tools that support the entry and editing of requirements specification data and checking the consistency and completeness of the requirements data against allowable specification constructs and rules.
	Software Requirements Analysis	Requirement Validation and Verification Support	Tools that support the evaluation of the completeness, correctness, and usability of the requirements.
	Software Requirements Analysis	Requirement Management Support	Tools that support the recording, storage, manipulation, import and export, and the administration of requirements and associated information.

3.3 Modeling Tools

Modeling tools are used in the design process to simplify development of the software architecture, database design, user interface design, and other tasks. **Because the output of some of these tools can be used directly in the generation of source code and databases, ISL concludes that modeling tools should have regulatory guidance.** The extent of guidance should be based on the effect that the tool can have on the final software.

Table 3.3 – Software Tools – Modeling Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Implementation	Software Architectural Design	Diagram Development	Tools that support the entry and editing of diagrams of types of interest to the user, and to translate between diagram types, and between diagrams and text.
	Software Architectural Design	Diagram Analysis	Tools that support the analysis of graphical figures input to the CASE tool and extracting and storing requirements and/or design information.
	Software Requirements Analysis	Requirement Specification Support	Tools that support the entry and editing of requirements specification data and checking the consistency and completeness of the requirements data against allowable specification constructs and rules.
	Software Detailed Design	Design Specification Support	Tools that support the entry and editing of design specification data and checking the consistency and completeness of design data against allowable specification constructs and rules.
	Software Detailed Design	Specification Construct Modeler	Tools that support the entry and editing of information describing the types of constructs that a specification can contain, including their relationships and depiction.
	Software Construction	Simulation	Tools that support the ability to simulate aspects of a system's potential operation based upon requirements and/or design data available to the CASE tool.
	Software Construction	Prototyping	Tools that support the ability to generate a prototype model of all or portions of a system based upon user-supplied requirements and/or design information
	Software Construction	Human Interface Modeler	Tools that support the ability to model the content aspects of human-computer interactions and the mechanical aspects of those interactions.

3.4 Construction Tools

Construction tools are used in the development process to translate source code into machine executable code, create databases, and user interfaces. They include compilers, linkers, and integrated development environments. **Because the output of these tools has a direct effect on the final software output, ISL concludes that construction tools should have regulatory guidance.**

Table 3.4 – Software Tools – Construction Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Implementation	Software Construction	Code Generator	Tools that support the ability to generate code in one or more specific languages based upon design data available to the CASE tool.
	Software Construction	Database Schema Generator	Tools that support the ability to generate database schema based upon user-supplied information.
	Software Construction	Screen Generator	Tools that support the ability to generate display screens based upon user-supplied information
	Software Construction	Report Generator	Tools that support the ability to automate the development of reports to be produced by the system under development (as opposed to the CASE tool).
	Software Integration	Compiler	Tools that support the ability to compile code written in one or more specific programming languages.
	Software Integration	Syntax Directed Editor	Tools that support the entry of source code in one or more specific languages with syntax support provided by the editor.
	Software Integration	Debugger	Tools that support the identification and isolation of errors in a program

3.5 Maintenance Tools

Maintenance tools support troubleshooting, fault isolation, reverse engineering, and other similar tasks. These tools can have the ability to alter the software in the Instrumentation and Control systems and they may fail to identify faults during use. **Because of these possible effects, ISL concludes that maintenance tools should have regulatory guidance.**

Table 3.5 – Software Tools – Maintenance Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Support	Software Problem Resolution	Problem Understanding	Tools that support the ability to determine that a problem: 1) results from a user misunderstanding, 2) has already been resolved, 3) is going to be resolved in the context of another maintenance action, or 4) is a new problem to be resolved.
	Software Problem Resolution	Localization	Tools that support the ability to identify the portion of the software requiring modification, given the identification of a problem.
	Software Problem Resolution	Impact Analysis	Tools that support the ability to, for each change foreseen, identify potential consequences of making the change.
Software Technical	Software Maintenance	Data Reverse Engineering	Tools that support the ability to extract information from source code which defines or describes the data elements and structures of the software.
	Software Maintenance	Process/ Procedure Reverse Engineering	Tools that support the ability to extract process design data from source code.
	Software Maintenance	Source Code Restructuring	Tools that support the ability to input existing source code in one or more specific languages, modify its format and/or structure according to defined directives (e.g., reduce size of code, reduce execution time, implement code format standard) and output a source code file in the same language.
	Software Maintenance	Source Code Translation	Tools that support the ability to input existing source code written in one or more specific languages, translate it into a different language, and output the resulting code.

3.6 Documentation Tools

Tools used to support the documentation process do not have a direct effect on the final software outputs. **Therefore, ISL concludes that documentation tools do not need regulatory guidance unless they are also used during the code construction process.**

Table 3.6 – Software Tools – Documentation Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Support	Software Documentation Management	Text Editing	Tools that support the ability to edit text
	Software Documentation Management	Graphical Editing	Tools that support the ability to enter and edit data in graphical format.
	Software Documentation Management	Forms-Base Editing	Tools that support user definition of forms and subsequent forms-based editing.
	Software Documentation Management	Publishing	Tools that support desktop publishing
	Software Documentation Management	Hypertext Support	Tools that support hypertext formats and functions.
Software Reuse	Reuse Asset Management	Variant Handling	Tools that support the ability to reuse the same generation of the product with limited variation.
	Reuse Asset Management	Automatic Data Extraction and Document Generator	Tools that support the ability to accept, store and retrieve specifications of the content, format and layout of textual and graphical data to be extracted and produced, and its ability to then extract and produce the data in compliance with a specification.

3.7 Configuration Management Tools

Configuration management tools can have a direct effect on the final software output by allowing unauthorized or untracked modifications to be made to the software. These tools control access to the source code, track changes, and may be used to generate the build package for compilation. **Because of these characteristics, ISL concludes that configuration management tools need to have regulatory guidance.**

Table 3.7 – Software Tools – Configuration Management Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Support	Software Configuration Management	Access Control	Tools that support the ability to control access to data elements
	Software Configuration Management	Tracking of Modifications	Tools that support the ability to maintain a record of all modifications made to the system under development or maintenance.
	Software Configuration Management	Definition and Management of Multiple Versions	Tools that support the ability to maintain records and perform management functions on multiple versions of a system that may share common components.
	Software Configuration Management	Configuration Status Accounting	Tools that support the ability to provide the user with reports defining the history, contents, and status of the various configuration items being managed.
	Software Configuration Management	Release Generator	Tools that support user definition of steps required to create a version (build) of the software for release, and to automatically execute those steps.
	Software Configuration Management	Archival Capability	Tools that support the ability to automatically place data elements in secondary storage for subsequent retrieval.

3.8 Quality Assurance Tools

Quality assurance tools do not have a direct effect on the final software output of the development process; however, these tools can have an indirect effect on the final software output. **Because the quality assurance process is an integral part of an Appendix B development, ISL concludes that quality assurance tools need to have regulatory guidance.**

Table 3.8 – Software Tools – Quality Assurance Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Support	Software Quality Assurance	Quality Data Management	Tools that support user entry of quality data, analyze quality data, and generate information to support quality management.
Project Support	Risk Management	Risk Management	Tools that support risk identification, risk estimation, risk impact assessment, risk monitoring, and controlling.

3.9 Verification and Validation Tools

Verification and validation tools are used to simplify and organize the testing required for these tasks. They do not have a direct effect on the final software output. However, they have the capability of failing to locate defects in the software or they may be used to justify eliminating other verification and validation tasks or to justify eliminating programming functions that can catch faults such as fault handling subroutines. **Because of this ability, ISL concludes that verification and validation tools should have regulatory guidance.**

Table 3.9 – Software Tools – Verification and Validation Category

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Support	Software Verification	Specification Traceability Analysis	Tools that support the ability to perform traceability analyses.
	Software Verification	Specification Analyses	Tools that support the ability to perform analyses based upon the requirements and design data available to the tool.
	Software Verification	Source Code Analysis	Tools that support the ability to input source code in one or more specific languages and perform analyses.
	Software Verification	Proof of Correctness Techniques	Tools that support the ability to formally prove assertions about features or operations of the software to be validated.
	Software Verification	Failure Analysis	Tools that support the ability to analyze failures and trace them back to defects.
	Software Verification	Defect Analysis	Tools that support the ability to analyze defects and trace them forward to failures.
	Software Verification	Test Case and Expected Result Entry	Tools that support user entry of test cases and entry of expected test case results.
	Software Verification	Test Case and Expected Result Generator	Tools that support the ability to automatically generate test cases, based upon existing requirements and/or design specification data available to the tool, and to automatically generate expected test case results.
	Software Verification	Test Traceability	Tools that support the traceability of test activities and data.
	Software Validation	Source Code Instrumentation	Tools that support the ability to automatically instrument code to be tested in order that test events can be identified and recorded.
	Software Validation	Input Capture and Replay	Tools that support the ability to capture operator inputs (e.g., keyboard, mouse) and the extent to which such data can be edited and replayed in subsequent test cases.

Life Cycle Process	Life Cycle Sub-Process	Software Tool	Description
Software Support	Software Validation	Test Driving	Tools that support the ability to execute and/or replay test cases.
	Software Validation	Run-time Analysis	Tools that support the ability to analyze the performance of a program as it executes.
	Software Validation	Reliability Analysis	Tools that support the ability to analyze measures of software reliability.
	Software Validation	Test Coverage Analysis	Tools that support the ability to analyze and report on test coverage, including system coverage analysis and function coverage analysis.
	Software Validation	Test Procedure Management	Tools that support the ability to manage test activities and a test program.
	Software Validation	Regression Testing	Tools that support regression testing.
	Software Validation	Automatic Result Checking	Tools that support the ability to automatically compare expected test case results and actual test case results.
	Software Validation	Test Statistical Analysis	Tools that support the ability to statistically analyze and report on test results.
	Software Validation	Operations Environment Simulation	Tools that support the simulation of a real operations environment, such as a large number of users, as well as various scenarios of use and various configurations.
	Software Validation	Integration Testing	Tools that support software integration activities.

Intentionally Blank

4 Standards, Guidelines, Positions, Review and Approval Practice Discussion

Numerous documents including nuclear regulatory guidance, international standards, and guidance from other industries were surveyed to determine what guidelines and regulations exist for the use of software tools. Documents from the National Aeronautics and Space Administration (NASA), Federal Aviation Administration (FAA), Radio Technical Commission for Aeronautics (RTCA), Institute of Electrical and Electronics Engineers (IEEE), International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), International Atomic Energy Agency (IAEA), Electric Power Research Institute (EPRI), National Institute of Standards and Technology (NIST), and Atomic Energy of Canada Limited (AECL) were surveyed. Documents were first scanned to determine whether they discuss software tool usage, commercial-off-the-shelf (COTS) software tool usage, government-off-the-shelf (GOTS) software tool usage, or modified-off-the-shelf (MOTS) software tool usage. If the initial scan showed that the document was relevant to software tools, then the document was surveyed in detail and a complete summary of the document is provided in Section 4.1. If the initial scan showed that the document provided no information on the handling of software tools, including COTS, GOTS or MOTS software tools, then only a short summary of the document is provided in Section 4.2. Separating the relevant and non-relevant documents into two different sections allows the reader to concentrate on only relevant material if the reader so chooses.

4.1 Documents Relevant to Software Tools

4.1.1 Space Industry Documents Discussion

4.1.1.1 NASA-STD-2201-93 Discussion

NASA-STD-2201-93 [7] is a high-level standard for NASA software quality assurance (SQA) that is to be applied by the provider, i.e., creator, of NASA software. The high-level standard applies for NASA providing software to another NASA division or for another company providing NASA software. This standard provides a means for ensuring high quality software, provides a means for ensuring the software is suitable for its intended purpose, and provides a high-level SQA plan that can and should be tailored to the specific software being provided. The document states that this standard should be applied to "non-deliverable" software that includes software tools. The standard is superseded by NASA-STD-8739.8 [8].

This NASA standard specifies the software assurance program for the provider of software. The standard also delineates the assurance activities for the provider and the assurance data that are to be furnished by the provider to the acquirer.

Section 1.3.3 states that the NASA software assurance standard and other NASA standards should be applied to all software developed for and by NASA, including non-deliverable software (commercially available or user-developed) used for development, fabrication manufacturing process control, testing, or acceptance of deliverable software or hardware (test and acceptance software; software design, test, and analysis tools; compilers, etc.).

Section 3.2 discusses the software assurance plan and states that the software assurance process to be used must be documented in a plan. The software assurance plan must address improved use of tools and techniques. The standard stresses traceability, continuous improvement and a strategy that emphasizes prevention of errors, not correction.

4.1.1.2 NASA-GB-A201 Discussion

Guidebook NASA-GB-A201 [10] provides guidance to NASA managers responsible for software acquisition and development and for establishing software assurance requirements. This document reinforces the position stated in NASA-STD-2201-93 [7] that the NASA software assurance standard and other NASA standards should be applied to software tools and COTS software used for development, fabrication, manufacturing process control, testing, or acceptance of deliverable software or hardware.

4.1.1.3 NASA-STD-8739.8 Discussion

NASA-STD-8739.8 [8] replaces NASA-STD-2201-93 [7]. This standard specifies the software assurance requirements for software developed or acquired and maintained by NASA and for open source software, GOTS software, MOTS software, and COTS software when included in a NASA system. This standard applies to use of new and existing (e.g., reuse, legacy, heritage) software products and components. This standard is compatible with all software life cycle models (e.g., waterfall, spiral, evolutionary, incremental, package-based), and addresses all software life cycle processes, including acquisition, supply, development, operation, and maintenance. This standard provides a consistent, uniform basis for defining the requirements for software assurance programs to be applied and maintained throughout the life of that software, that is, from project conception, through operations and maintenance, until the software is retired.

Managers, working with software assurance personnel identify the appropriate software class and the level of software assurance effort to apply. Tailoring the implementation of software assurance requirements is acceptable commensurate with the program/project classification as well as size, complexity, criticality, and risk.

Section 5 of this standard addresses the software quality assurance requirements of the acquirer from program/project initiation through retirement of the software. Section 5.4.1.2 specifies that, as part of software development, the acquirer software assurance manager shall verify that the provider has developed and maintained processes for assurance of COTS, GOTS, and MOTS software addressing both the basic acquired software and any modifications or applications written to adopt them into the intended system. Section 5.4.1.5 specifies that, as part of software development, the acquirer software assurance manager shall assure that both deliverable and any designated non-deliverable software development products have proper configuration management.

Section 6 of this standard addresses requirements of software assurance for the actual provider of the software products. The provider designs, develops, implements, tests, operates, and maintains the software products and has software assurance requirements relative to those

software engineering activities. Section 6.1.1 states that the provider shall plan, document, and implement a software assurance program for software development, operation, and maintenance activities. This includes documentation of software assurance procedures, processes, tools, techniques, and methods to be used. Section 6.1.2 states that the software assurance program shall include processes for assurance of COTS, MOTS, and GOTS software addressing both the basic acquired software and any modifications or applications written to adopt them into the intended system. Section 6.8.3 states that software assurance personnel shall be trained in relevant software engineering design methods and languages, processes, development environments, tools, test techniques, and other software engineering and assurance methods needed to stay current with the engineering environment and products they must assure.

Appendix A of this standard provides a complex software assurance classification assessment developed by NASA to identify and evaluate the characteristics of software in determining the software's classification and level of software assurance to be applied. The process for assessing and classifying software consists of four steps:

Step 1 - Safety-Critical Determination

Ascertain if the software has safety implications for the system, property external to the system, or to the environment and if human life is a risk factor. The software is considered safety-critical if it resides in a safety-critical system as determined by a hazards analysis (i.e., hazard severity is catastrophic, critical, moderate or negligible), processes data or analyzes trends that lead directly to safety decisions, or provides full or partial verification or validation of safety-critical systems. If the software is safety-critical, then full software assurance is required independent of any further classification steps and the project must adhere to NASA-STD-8719.13B [9].

Step 2 - Software Class Determination

Determine the software engineering class of software (Class A-E) based on NASA Procedural Requirements (NPR) 7150.2 [112].

Class A, human-rated software systems, applies to all space flight software subsystems developed and/or operated by or for NASA, to support human activity in space and that interact with NASA human space flight systems. Examples of Class A software include guidance, navigation and control; life support; crew escape; automated rendezvous and docking; failure detection; isolation and recovery; and mission operations.

Class B, non-human space rated software, is flight and ground software that must perform reliably in order to accomplish primary mission objectives. Examples include propulsion systems; power systems; guidance, navigation and control; fault protection; thermal systems; command and control ground systems; planetary surface operations; hazard prevention; primary instruments; or other subsystems that could cause the loss of scientific data return from multiple instruments.

Class C, mission support software, includes flight or ground software that is necessary for the scientific data return from a single (non-critical) instrument, or is used to analyze or process

mission data, or other software for which a defect could adversely impact attainment of some secondary mission objectives or cause operational problems for which potential workarounds exist. Examples include prelaunch integration and test, mission data processing and analysis, analysis software used in trend analysis and calibration of flight engineering parameters, and primary/major science data collection and distribution systems. Class C software must be developed carefully, but validation and verification effort is generally less intensive than for Class B.

Class D, analysis and distribution software, is non-space flight software developed to perform science data collection, storage, and distribution; or perform engineering and hardware data analysis. A defect in Class D software may cause rework but has no direct impact on mission objectives or system safety. Examples of Class D software include but are not limited to software tools; analysis tools, and science data and distribution systems.

Class E, development support software, is non-space software developed to explore a design concept; or support software or hardware development functions such as requirements management, design, test and integration, configuration management, documentation, or perform science analysis. A defect in Class E software may cause rework but has no direct impact on mission objectives or system safety. Exploratory software is now a subset of Class E and is not distributed for use outside the development and usage group, either within a NASA center or externally, and is not used to operate equipment or facilities that are safety-critical. No software assurance for Class E software is required.

Classes F, G and H are designated by the Chief Information Officer (CIO) and software assurance is only performed upon request or as designated by the CIO.

Class F, general purpose computing software (multi-NASA center/multi-project), includes software used in support of voice, wide area network, local area network, video, data centers, application services, messaging and collaboration, and public web. A defect in Class F software is likely to affect productivity of multiple users across several geographic locations and may possibly affect mission objectives or system safety including cost, schedule, or technical objectives for any work NASA performs. Examples of Class F software include the NASA web portal and software supporting NASA’s financial program such as time and attendance system, travel management, business warehouse and payroll.

Class G, general purpose computing software (single-NASA center/single project), includes software used in support of a single NASA center or a single project in support of voice, local-area network, video, data centers, application services, messaging and collaboration, and public web. A defect in Class G software is likely to affect productivity of a single user or small group of users but generally will not affect mission objectives or system safety. Examples of Class G software include the NASA Headquarters' Corrective Action Tracking System and NASA Headquarters' New User Request System.

Class H, general purpose desktop software, includes software for Wintel, Mac, and UNIX desktops as well as laptops. A defect in Class H software may affect the productivity of a single user or small group of users, but generally will not affect mission objectives or system safety.

However, a defect in desktop IT-security-related software (e.g., antivirus) may lead to loss of functionality and productivity across multiple users and systems. Examples of Class H software include typical desktop applications such as Microsoft Word, Excel, and PowerPoint and Adobe Acrobat.

Step 3 - Software Classification Scoring Process

Using a complicated Excel-based spreadsheet, called the Software Classification Score Sheet, expand upon the classification and obtain a criticality score based on project/mission characteristics, the tasks the software will perform, and any software unique characteristics. The total score is used as input for determining the level of software assurance in Step 4.

Step 4 - Software Assurance Level Determination

Based on the hazard assessment from Step 1, which determines whether the software is safety-critical, the software class from Step 2, the software classification score from Step 3, and additional potentials for mission failure, resource investment, and impact to equipment and environment, the software assurance effort is determined.

Appendix C of this standard is a requirements compliance matrix to document and check compliance to this NASA standard for use in the approval/review process.

4.1.1.4 NASA-STD-8719.13B Discussion

NASA-STD-8719.13B [9] provides guidance on the creation and assurance of safe software. The focus of the document is on safety-critical software; however, the document also states that the information can be used in the development of mission-critical software.

Section 5.11 discusses measures to use to prevent the inadvertent introduction of errors due to the use of software tools. The primary approach discussed is identification of all project tools that could potentially impact safety-critical software, the degree of impact, and mitigation strategies in the project plan. The process and criteria used to select, approve, and control the tools shall be described in the project plan. Installation of upgrades to approved tools, withdrawal of previously approved tools, and identification of limitations of the tools should be identified.

Appendix A.4.11 describes an example of the qualification process of several different types of tools including compiler, operating system, libraries, language, code generators, and development environment. The example is somewhat sparse on details but follows the guidelines in Section 5.11.

4.1.1.5 NASA-GB-8719.13-1 Discussion

NASA-GB-8719.13-1 [12] provides guidance on use of the NASA software safety standard, NASA-STD-8719.13B [9].

Chapter 11 of the document addresses programming languages and the tools associated with software development in detail. The discussion of tools covers compilers, editors, integrated

development environments (IDEs), and other tools. One of the more important sections discusses automatic code generation as would be found in the various “visual” programming languages and tools.

The section on IDE provides general guidelines for selection including that it is easy to learn, well integrated or easy to integrate with other tools, defaults to enforcing standards, and is well documented. Similar guidelines are provided for editors, compilers and linkers, and debuggers.

Section 11.3 discusses other types of Computer-Aided Software Engineering (CASE) tools including tools for structured analysis, structured design, code generators, and others. It provides three classifications for CASE tools. Upper CASE tools are used early in the development process when initial design, data and process requirements, and the interface design are determined. Lower CASE tools are primarily those that generate or modify code. The third category is integrated CASE (I-CASE) which integrates the upper and lower CASE tools and may add additional functionality.

This section also discusses visual programming languages, visual programming environments, and code generation from design models. The discussion is general, describing how each functions and some advantages and disadvantages, but provides little guidance on selection of tools. The summary of the section states that automatic code generation is unreliable for safety-related software and that automatically generated code should be subject to the same inspection, analysis, and testing as hand-generated code.

4.1.1.6 NASA-GB-001-96 Discussion

NASA-GB-001-96 [15] is a manager's guidebook that defines the activities required of NASA software projects and defines life-cycle models and activity-related methods. This guidebook provides specific guidance to software project managers and team leaders in selecting appropriate life cycles and methods to develop a tailored plan for a software engineering project. This guidebook summarizes the common elements of the overall NASA software engineering process; describes the NASA software engineering process in detail, including summary descriptions of required activities and products; recommended methods for performing those activities; and discusses running and closing out software projects. The guidebook introduces a concept know as a process asset library (PAL) which contains reusable assets from other projects.

Four developmental life cycle models and one maintenance life cycle model are discussed from most simple and familiar to most complex and least familiar:

> Waterfall developmental life cycle model - This model is a once through do each step once approach which first determines user needs then defines requirements, designs the system, implements the system, tests, fixes, and delivers the system.
>
> Incremental developmental life cycle model - This model is an incremental (multi-build) approach that determines user needs and defines a subset of the system requirements, then performs the rest of the development in a sequence of builds. The first build

incorporates part of the planned capabilities; the next build adds more capabilities, and so on, until the system is complete.

Evolutionary developmental life cycle model - This model, like the incremental model, develops a system in builds, but differs in acknowledging that the user needs are not fully understood and not all requirements can be defined up front. User needs and system requirements are partially defined up front, and then are refined in each succeeding build. The system evolves as the understanding of user needs and the resolution of issues occurs.

Package-based developmental life cycle model - This model is used for system development based largely on the use of COTS and GOTS products and reusable packages.

Legacy system maintenance life cycle model - This model is used to apply fixes or minor enhancements to an operational system. Major enhancements require a waterfall or incremental model. It is similar to the waterfall model; however, the architectural design has already been established.

The guidebook emphasizes that COTS software, GOTS software, reusable software, and non-developed software must undergo some level of verification and validation (V&V) and preparation for integration. Prototyping one of these products should occur early in the life cycle to evaluate the product's capabilities and performance and whether it meets already defined functionality and performance requirements. If prototyping proves successful, then plan for its interfaces to other COTS products or to the rest of the system. The quality and reliability of vendor support will be crucial to successful implementation. This discussion is the extent of the guidebooks guidance, regulation, or review/approval of software tools or COTS software.

4.1.2 Civil Aviation Industry Documents Discussion

4.1.2.1 FAA-STD-026A Discussion

FAA-STD-026A [16] establishes and communicates requirements to be applied during the development and support of computer software and related documentation associated with National Airspace System (NAS) acquisition. It applies to software as a stand-alone product or as a subsystem of a specified product. It applies to software delivered to the FAA including non-developmental software (NDS) as well as software developed as an essential component of performing contract requirements (such as tools developed during contract performance). This standard represents the FAA approved tailoring of the IEEE/EIA 12207 [17, 18, 19] series of standards for the procurement of computer software products and services.

This standard states that all software not developed under the contract (including, but not limited to, tools and commercially available software applications, i.e., COTS software) is regarded as NDS. All NDS shall be identified and shall be addressed within required software development activities and documentation. Validation that the NDS performs its required functions and operates correctly within the software system design shall be accomplished.

Appendix A of this FAA standard identifies 17 primary and 8 supplemental documents required to be prepared during software development. These required documents, called data item descriptions (DIDs), are based on the data requirements of the IEEE/EIA 12207 [17, 18, 19] series standards. Appendix B of the FAA standard contains a mapping of this standard's data requirements to IEEE/EIA 12207.1 [18]. The remainder of FAA-STD-026A [16] is comprised of 25 DID templates which describe the required content of the 25 documents and general instructions for preparing the document. All NDS shall be identified and addressed within these 25 DID documents.

4.1.2.2 RTCA/DO-178B Discussion

The RTCA is an association of aeronautical organizations of the United States of America from both government and industry. RTCA/DO-178B [20] provides guidelines for the production of software for airborne systems and equipment. Chapter 4 deals with compilers and test coverage when using optimization. Chapter 6 discusses software verification. Chapter 7 discusses the software configuration management (SCM) process. Chapter 12 deals with the use of previously developed software, software tool qualification and alternate methods of providing software reliability.

Chapter 4, “Software Planning Process,” Section 4.4, discusses the software life cycle environment, including methods and tools used. Methods and tools should be chosen that provide error prevention during the software development process. If the compiler used to generate executable code uses optimization, the correctness of the optimization need not be verified if the test cases give coverage consistent with the software level. Otherwise, the impact of optimization should be determined as discussed in Section 6.4.4.2. Compilers may also produce object code that is not directly traceable to source code, for example, initialization, built-in error detection, or exception handling. The software planning process should provide a means to detect this object code and to ensure verification coverage.

Chapter 6, “Software Verification Process,” states that verification is a technical assessment of the results of both the software development processes and the software verification process. The purpose of the software verification process is to detect and report errors that may have been introduced during the software development processes. The software verification process is completed through a combination of reviews, analyses, and the development and execution of test cases and procedures. This chapter does not mention software tools that could be used during the verification process; however, Chapter 12 does and refers back to Chapter 6 for verifying the tools.

Section 7.2.9 discusses the software life cycle environment control. The objective of software life cycle environment control is to ensure that the tools used to produce the software are identified, controlled, and retrievable. The tools are defined by the software planning process and identified in the software life cycle environment configuration index. Guidance for the use of software tools includes:

a. Configuration identification should be established for the executable object code (or equivalent) of the tools used to develop, control, build, verify, and load the software.

b. The SCM process for controlling qualified tools should comply with the objectives associated with Control Category 1 for software development tools and Control Category 2 for software verification tools.

c. Unless paragraph (b) above applies, the SCM process for controlling the executable object code (or equivalent) of tools used to build and load the software (compilers, assemblers, linkage editors) should comply with the objectives associated with Control Category 2, as a minimum.

The chapter then goes on to describe the SCM process objectives of Control Categories 1 and 2 with Category 1 requiring more processes than Category 2 as shown in the following table. Since tools used to build and load the software are software development tools, paragraph (c) appears to apply to unqualified software development tools. ISL does not agree that unqualified software development tools should be subject to less strict software configuration management processes than qualified software development tools. The RTCA will be contacted as part of Task 2 of this research project, which is the analysis of standards, guidance, and review/approval practices, for clarification of the meaning of paragraph (c) above.

Table 4.1 – SCM Process Objective Control Categories

SCM Process Objective	CC1	CC2
Configuration Identification	•	•
Baselines	•	
Traceability	•	•
Problem Reporting	•	
Change Control - integrity and identification	•	•
Change Control - tracking	•	
Change review	•	
Configuration Status Accounting	•	
Retrieval	•	•
Protection against Unauthorized Changes	•	•
Media Selection, Refreshing, Duplication	•	
Release	•	
Data Retention	•	•

Chapter 12, “Additional Considerations”, contains a significant amount of information on software tool qualification for satisfying certification requirements. Subsection 12.1 is devoted to a discussion of the issues associated with use of previously developed software. If new development tools are used, they must be qualified as discussed in Subsection 12.2. Subsection 12.2 states that qualification of a tool is needed when processes of RTCA/DO-178B [20] are eliminated, reduced, or automated by the use of a software tool without its output being verified as specified in Chapter 6. The use of software tools to automate activities of the software life cycle processes can help satisfy system safety objectives insofar as they can

enforce conformance with software development standard and use automatic checks. The objective of tool qualification is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced or automated. This subsection states that only "deterministic" tools can be qualified, i.e., tools which produce the same output for the same input when operating in the same environment. This appears to preclude the use of any "nondeterministic" tool since it cannot be qualified and all tools must be verified or qualified. The RTCA will be contacted as part of Task 2 of this research project, which is the analysis of standards, guidance, and review/approval practices, for clarification of the meaning of the "deterministic" statement.

Chapter 12 states that there are two types of tools, software development tools and software verification tools which are described below.

Software development tools are those whose output is part of airborne software and thus can introduce errors. For example, a tool which generates source code directly from low-level requirements would have to be qualified if the generated source code is not verified as specified in Chapter 6. Qualifying a software development tool is extensive compared to software verification tools. Subsection 12.2.1 states that software development tools need to be treated at a level equivalent to the airborne software and that the tool meets its tool-specific operational requirements. A trial period may be needed during which a verification of tool output is performed and tool-related problems are analyzed, recorded and corrected to demonstrate that the tool meets its operational requirements.

Software verification tools are those that cannot introduce errors, but may fail to detect them. For example, a static analyzer, that automates a software verification process activity, should be qualified if the function that it performs is not verified by another activity. Type checkers, analysis tools and test tools are other examples. Software configuration management and quality assurance applies to software tools to be qualified. Subsection 12.2.2 states that software verification tools can be qualified by just confirming that the tool meets its operational requirements. A software tool may be qualified only for use on a specific system. Tool qualification data must be generated, a tool qualification plan must be generated and followed and tool operational requirements must be specified to describe the tool's operational functionality including all functions and technical features.

4.1.2.3 FAA Order 8110.49 Discussion

FAA Order 8110.49 [21] clarifies software review and approval processes, determines the level of FAA involvement, discusses approval of software changes made in the field to reduce aircraft maintenance time, and discusses the qualification of software tools. Much of the information in Chapters 4, 7, and 9 is relevant to software tools used in the development of digital instrumentation and control (I&C) software in nuclear systems.

Chapter 4, "Software Conformity Inspection," Section 4.3.a(1), notes that in some cases, special purpose software is used for environmental qualification testing. When this is the case, the manufacturer must verify, validate, and control the configuration of the special purpose test software. The test software should be included as part of the test setup conformity conducted

before the qualification testing. In Section 4.3.a(4), this chapter states that the FAA aviation safety engineers should establish that any software development tools or software verification tools that require qualification have been qualified. However, if the tool qualification activities are not completed at the time of conformity, the tools and supporting data should have their configuration documented. The statement in Section 4.3.a(4) is interesting in that the software tool qualification requirement is not conditional on their ability to detect or fail to detect errors in the final plant safety-related digital I&C software.

Chapter 7, "Approval of Airborne Systems and Equipment Containing User-Modifiable Software (UMS)," Section 7-6, supplements RTCA/DO-178B [20], Section 5.2.3, and discusses requirements for UMS. UMS is software within an airborne system approved for user modification. Users (such as airlines and operators) may modify UMS within the specified modification constraints and with approved modification procedures without involvement by the certification authority. Section 7-6 requires that the non-modifiable software components be protected from modifiable components to prevent interference with the safe operation of the non-modifiable software components. To enforce this protection, tools are allowed to make changes to the modifiable component provided the following information is provided to the certification authority for approval:

(1) Plans for controlling tool version;
(2) Plans for controlling tool usage;
(3) Plans for qualifying or verifying the tool (RTCA/DO-178B [20] Section 12.2 and Chapter 9 or this order); and
(4) Procedures for modifying the tool.

Software forming a component of the tool and used in the protective function should be developed to the software level of the most severe failure condition of the system, as determined by a system safety assessment. Section 7-7 states that if software tools are to be used for modifying UMS, the certification plans should identify tool qualification plans or verification procedures to ensure that the tool has modified the UMS to approved procedures and constraints, and it has not affected the non-modifiable software or protection mechanisms.

Chapter 9, "Qualification of Software Tools Using RTCA/DO-178B [20]," discusses qualification requirements for software tools that are used to eliminate, reduce, or automate a process during software development. This chapter clarifies the intent of RTCA/DO-178B [20], Section 12.2 and its application. Areas of RTCA/DO-178B [20] clarified in this chapter of FAA Order 8110.49 [21] include:

(1) When a tool should be qualified
(2) Justification for the different criteria for qualifying software development tools and software verification tools
(3) Which criteria apply to software development tools and which apply to software verification tools
(4) Data to be produced for software development tools and for software verification tools
(5) Acceptance criteria for tool operational requirements

(6) Tool determinism
(7) Tool partitioning assurance and evidence
(8) Tool configuration control

According to this chapter, not all software tools require qualification. Qualification of a tool is only required if one of the processes described in RTCA/DO-178B [20] is eliminated, reduced, or automated and the tool has not been verified as specified in RTCA/DO-178B [20] Section 6. This chapter considers that there only two types of tools: software verification tools and software development tools. There are different qualification guidelines for each tool type. Verification tools are tools that cannot introduce errors but may fail to detect them. Development tools are tools whose output is part of airborne software and thus can introduce errors. The remainder of the chapter clarifies which qualification criteria of RTCA/DO-178B [20] Section 12.2 apply to each type of tool, what data needs to be transmitted to the FAA or made available to the FAA, and guidelines for configuration control of the software tool.

4.1.2.4 FAA Order 8110.49 Change 1 Discussion

FAA Order 8110.49 Change 1 [22] replaces the 2003 version of FAA Order 8110.49 [21] and guides aircraft certification service (AIR) field offices and designated engineering representatives (DER) on how to apply RTCA/DO-178B [20] for approving software used in airborne computers.

Change 1 to the 1993 version of FAA Order 8110.49 [21] incorporates policy from Notice 8110.110 [113]. Notice 8110.110 [113] was written to supplement RTCA/DO-178B [20] to give the FAA assistance in approving airborne software, and to help the FAA ensure that an applicant establishes appropriate processes and procedures that result in compliance to RTCA/DO-178B [20] objectives. Change 1 also updates document references to reflect current versions, where applicable and revises chapter 3 to reflect current policy regarding use of DERs in technical standard order (TSO) authorization.

This order addresses four new topics discussed in Chapters 13 through 16. Chapter 13 discusses properly overseeing suppliers, Chapter 14 discusses software problem reporting, Chapter 15 discusses assurance of airborne system databases and aeronautical databases, and Chapter 16 discusses the management of software development or verification environments.

The discussion of software tools in Chapters 4, 7, and 9 remain unchanged from the 2003 version.

The policies in new Chapter 13 apply when an applicant uses suppliers and sub-tier suppliers to perform system and software development, verification, and certification activities. The applicant should create oversight plans and procedures that will ensure all suppliers will comply with all regulations, policy, guidance, agreements, and standards that apply to the certification program. The applicant's planning documents, such as certification plans and plans for software aspects of certification (PSAC), should describe how the applicant will have visibility into their suppliers' and sub-tier suppliers' activities. This includes COTS software component

suppliers and vendors. The applicant should submit these plans to the FAA for review and approval.

The policies in new Chapter 14 apply when an applicant's suppliers and sub-tier suppliers will be responsible for managing problems detected during the development of aircraft systems implemented with software. The applicant should discuss in their software configuration management plan, or other appropriate planning documents, how they will oversee their supplier's and sub-tier supplier's software problem reporting process. These plans should describe any tools that the applicant's suppliers or sub-tier suppliers plan to use for the purpose of recording action items or observations for the applicant to review and approve prior to entering them into the applicant's problem reporting system.

The policies in new Chapter 15 apply when the applicant's airborne systems and equipment is utilizing aeronautical databases or airborne system databases. This chapter has no relevance to software tools.

The policies in new Chapter 16 apply when the applicant is using a software development or verification environment that may not be completely representative of the target computer. The applicant should establish and maintain configuration control of the environment, and implement a structured problem reporting system for the environment available to users of the environment. The software development plan and software verification plan should include an explanation of how the software development or verification environment will be used to show compliance with RTCA/DO-178B [20] objectives. If development tools are being used in the integrated environment, then verification should also be performed in the integrated environment.

4.1.2.5 DOT/FAA/AR-06/54 Discussion

DOT/FAA/AR-06/54 [23] discusses a software verification tools assessment study (SVTAS) performed by the NASA Langley Research Center. The SVTAS was a research effort to investigate criteria for effectively evaluating structural coverage analysis tools for use on projects intended to comply with RTCA/DO-178B [20].

RTCA/DO-178B [20] is the primary means used by aviation software developers to obtain FAA approval of airborne computer software. RTCA/DO-178B [20] describes software life cycle activities and design considerations, and enumerates sets of objectives for the software life cycle processes. The SVTAS considers one particular set of RTCA/DO-178B [20] objectives, namely, the verification objectives for structural coverage analysis.

In recent years, a number of commercial tools have been developed to ease the burden of structural coverage analysis. When automated tools are used for credit in the verification of software in compliance with RTCA/DO-178B [20] objectives, those tools are subject to qualification criteria unless the tool's output has been formally verified. Qualification requirements differ depending on whether a tool is a development tool or a verification tool. The SVTAS is focused specifically on structural coverage analysis tools, which are verification tools per RTCA/DO-178B [20].

Despite various attempts to clarify definitions related to structural coverage analysis, misunderstandings about fundamental aspects of structural coverage have continued to arise about how to consistently interpret and implement analysis criteria in tools and how to evaluate those tools for qualification purposes. The intent of the SVTAS was to identify some of the areas of misunderstanding and develop evaluation criteria with the goal of facilitating consistent assessment of structural coverage analysis tools.

The SVTAS was conducted in two phases. During the first phase of the study, commercial structural coverage tools and related policy and guidance were surveyed, issues that might contribute to inconsistencies in the automation of structural coverage analysis were identified, and resolutions for each issue were suggested. The STVAS concluded that FAA Order 8110.49 [21], RTCA/DO-178B [20], and RTCA/DO-248B [27] are the primary and auxiliary FAA-related documents providing policy and guidance on structural coverage tool qualification. The results of these phase 1 activities led to the conclusion that a test suite was a possible method of evaluating structural coverage analysis tools.

During the second phase of the study, a prototype test suite was developed to explore whether a test suite would be an effective approach to increase objectivity and uniformity in the application of the qualification criteria to structural coverage analysis tools. Test suite requirements to meet Level A, B, and C software structural coverage objectives were defined. The test suite requirements included tests for all high-order languages and source and object code and addressed statement coverage, decision coverage, and modified condition decision coverage. Because development and implementation of a full test suite was beyond the scope of the STVAS project, a subset of those requirements was implemented in a prototype test suite consisting of only Ada source code. The prototype test suite was evaluated using three different commercially marketed structural coverage analysis tools to determine the effectiveness of the test suite. The prototype test suite identified anomalies in each of the three tools. Some of the anomalies were false-negatives where the tool incorrectly reported that coverage was not attained when coverage was in fact attained. Some of the anomalies were false-positives where the tool incorrectly reported that coverage was attained when coverage was in fact not attained. Some of the anomalies were situations where the tool crashed, had difficulty, or limitations within the tool prevented coverage from being properly evaluated.

The results of the STVAS demonstrate the potential for a full test suite consisting of source code or object code in any language and consisting of many constructs to help evaluate whether a structural coverage analysis tool is compatible with RTCA/DO-178B [20] coverage requirements.

4.1.2.6 CAST-13 Discussion

CAST-13 [24] clarifies RTCA/DO-178B [20] Section 12.2.1.b as it applies to automatic code generation (ACG) tools. This paper proposes a list of candidate objectives in RTCA/DO-178B [20] that could potentially be alleviated, when applicants are qualifying ACG tools, provided that the applicants supply relevant rationale and justification for each objective's alleviation. This paper also provides a roadmap to potentially reduce the ACG tool's software level relative to the level of the airborne software.

ACG tools are applications that execute on ground computers using commercial off-the-shelf operating systems and directly translate software specifications into source code, thereby potentially eliminating or automating several software development and verification activities. A typical ACG contains two entities, (1) a basic library of symbols with associated source code and (2) logic and architecture that chooses an appropriate symbol based on low-level or high-level requirements and inserts the associated symbols' source code into the airborne software.

When ACG tools do not eliminate, reduce, or automate processes necessary to be performed on airborne software, they do not need to be qualified. When tools eliminate, reduce, or automate life cycle process activities on the airborne software, without their output being verified, the tools should be qualified.

If an ACG tool needs to be qualified, the software level of the tool needs to be determined. Where there is no justification to reduce the software level of the tool, the tool should be qualified to the same software level and objectives as that of the airborne software. However, some of the RTCA/DO-178B [20] objectives could be alleviated or removed if it can be demonstrated that incompatibility problems with the ACG tool are obvious and readily detected.

If justification of a lower software level exists, the verification objectives and activities for the reduced software level should be satisfied. Justification of a lower software level should address the consequences of a tool's software level reduction on the airborne software.

The primary issue for an ACG tool is the production of source code that does not comply with its requirements, but can still be compiled without any error detected and is executable. If the tool is designed and developed in such a way that the ACG tool's architecture and logic can produce unintended, non-deterministic, or erroneous code, then a software level reduction is not possible. Otherwise a level reduction is possible if it can be demonstrated that potential errors in the ACG tool's output can be detected by additional verification on the code produced or if it can be demonstrated that potential errors and failure conditions in the airborne software as a result of the ACG tool's software level reduction or objective's alleviation cannot contribute to the failures of the airborne software. The applicant can propose a reduction in the ACG software level based on a safety assessment, software architecture, tool partitioning, additional verification, and/or by other means.

4.1.2.7 FAA AC 20-148 Discussion

FAA Advisory circular (AC) 20-148 [25] provides one acceptable means of compliance for reusable software component (RSC) developers, integrators, and applicants to gain (1) FAA acceptance of a software component that may be only a part of an airborne system's software application, and (2) credit for the reuse of a software component in follow-on systems and certification projects, including full or partial credit for compliance to the objectives of RTCA/DO-178B [20].

The reuse concept in this AC may apply to verification and development tools. Applicants and tool developers must discuss with the FAA the details of each reusable tool qualification project.

Because tools differ from airborne software, there are other concerns to address when trying to reuse tool qualification data. The FAA plans to address tool reuse in future guidance.

Section 11 of this AC discusses common software reuse issues. If qualified tools are used to develop or verify the RSC, the RSC developer must consider reuse of those tools during RSC development and acceptance. For development tools, the tool qualification plan and the tool accomplishment summary must document portions of the tool qualification the applicant must complete. For verification tools, the PSAC and the software accomplishment summary (SAS) must document portions of the tool qualification the applicant must complete. The RSC developer must provide tool plans, tool operational requirements, and the tool accomplishment summary to the applicant for all tools used in getting acceptance of the RSC.

4.1.2.8 CAST-22 Discussion

Advisory Circular 20-148 [25] states that the reuse concept may apply to verification and development tools and that applicant and tool developers must discuss with the FAA the details of each reusable tool qualification project. AC 20-148 [25] states that the FAA plans to address tool reuse in future guidance. Position paper CAST-22 [26] provides this guidance.

CAST-22 [26] documents the process for reusing tool qualification data across company boundaries (i.e., to approach tools as a kind of reusable component that may be reused with minimal certification authority involvement on the subsequent projects). This paper builds upon the guidance of AC 20-148 [25] with specific focus on tool-unique aspects. If the tool is not used across company boundaries, this approach may not be desirable (i.e., Chapter 12 of Order 8110.49 [21] should be consulted instead).

Section 3 of CAST-22 [26] contains general guidelines for reusable tool qualification. When reusable software verification and development tools are initially planned, the boundaries and limitations of what is reusable and what will be project specific should be established during the first acceptance of the reusable tool qualification package. The tools should be packaged and qualified in such a way that some of the tool qualification data may be reused on other projects. Reusable tools must be discussed in several documents including the reusable tool qualification plan (RTQP), reusable tool accomplishment summary (RTAS), reusable tool configuration index (RTCI), and reusable tool data sheet (RTDS).

Section 4 of CAST-22 [26] discusses development tool guidelines. Reusable development tools should follow the guidance of AC 20-148 [25] for reusable software components. Where a PSAC, software configuration index (SCI), SAS, Data Sheet, etc. are described in AC 20-148 [25], an RTQP, RTCI, RTAS, RTDS, etc. should be used instead (i.e., the appropriate tool qualification data should be substituted for the RSC data). The initial reusable tool qualification must be done in the context of an actual certification project and must have the documented agreement of all the stakeholders. For each objective that the tool automates, replaces, or supplements, the amount of credit being sought (full, partial, or no credit), assumptions, means of compliance, tool limitations, and remaining activities to be completed by the tool user, system developer, and/or applicant must be described in the reusable tool documents.

Section 5 of CAST-22 [26] discusses verification tool guidelines. Reusable verification tools should follow the guidance of AC 20-148 [25] for reusable software components. The initial reusable tool qualification must be done in the context of an actual certification project and must have the documented agreement of all the stakeholders. The verification tool should have a RTQP, RTCI, RTAS, RTDS, and supporting reusable tool qualification data packaged separately from the project documents. Since verification tool objectives are not listed in RTCA/DO-178B [20], a table is included in CAST-22 [26] to describe 15 objectives for reusable verification tool qualification. The goal of the table is to provide a set of reusable verification tool qualification objectives that can then be used to communicate full, partial, or no qualification credit in a verification tool reuse project. As with development tools, for each objective that the tool automates, replaces, or supplements, the amount of credit being sought (full, partial, or no credit), assumptions, means of compliance, tool limitations, and remaining activities to be completed by the tool user, system developer, and/or applicant must be described in the reusable tool documents. Reusable verification tool documentation also includes a reusable tool operational requirements document, reusable tool qualification data, reusable tool user's guide, and a reusable tool data sheet that concisely describes the tool's functions, limitations, interface requirements, compliance concerns, assumptions, configuration, supporting data, open problem reports, tool characteristics, and other relevant information.

4.1.2.9 RTCA/DO-248B Discussion

RTCA/DO-248B [27] provides no new or additional guidance beyond the guidance in RTCA/DO-178B [20]. This document provides clarification of the guidance material in RTCA/DO-178B [20], resolves content errors, and resolves inconsistencies between RTCA/DO-178B [20] and any other relevant civil aviation standards. The clarification material is comprised of 12 errata that correct typographical errors, 76 frequently asked questions (FAQs) providing short and concise responses to questions posed to certification authorities or others who provide interpretation of RTCA/DO-178B [20], and 15 discussion papers (DPs) providing clarification of certain sections of RTCA/DO-178B [20] in cases where the clarification requires more than a short answer to a question.

FAQ #4 clarifies what is meant by COTS software. The question was whether COTS software includes software designed or having option-selectable functionality. The answer is that COTS software is only commercially available applications sold by vendors through public catalog listings and COTS software is not intended to be customized or enhanced. Contract negotiated software developed for a specific application is not considered COTS software. As long as the COTS software is not changed as supplied from the vendor, it is considers COTS software. If COTS software has option-selectable functionality, then the COTS software must meet the RTCA/DO-178B [20] guidelines detailing how to accommodate and verify option-selectable software. When the customer changes COTS software, in the context of RTCA/DO-178B [20], it must be reclassified as previously developed software and fulfill all objectives of previous developed software for certification.

FAQ #31 clarifies compiler validation. The question was how does verification of products relate to compiler acceptability. Verification planning needs to take into account compiler features.

The compiler is considered acceptable once all of the verification objectives are satisfied. However, the compiler is only acceptable for that product and not necessarily for other products. Satisfying all the verification objectives for a certain product does not qualify the compiler as a development tool. Compilers often undergo validation to show compliance with a language standard but compiler validation does not mean the compiler is qualified as a development tool. However, a product may be verified even though its software is developed using a compiler with no validation status.

FAQ #42 clarifies the term “structural coverage” and whether a compiler can be used to simplify the analysis. Structural coverage can be demonstrated by analyzing object code instead of source code as long as analysis can be provided which demonstrates that the coverage analysis conducted at the object code level will be equivalent to the same coverage analysis at the source code level. A compiler can be used to simplify the object or source code structural coverage analysis. When utilizing complier features to simplify analysis, one relies on the compiler to behave as expected. Therefore, one may need to qualify the compiler features being used as a verification tool.

FAQ #59 questions what type of non-flight software is covered by RTCA/DO-178B [20]. The only type of non-flight software covered by RTCA/DO-178B [20] is software used in tools, including development tools and verification tools used for development or verification of airborne software. Section 12.2 can be used for guidance to assess if a tool needs to be qualified, and to define the qualification process and qualification data.

FAQ #61 deals with development tools. The question is what constitutes a development tool and when should it be qualified. Section 12.2 defines developments tools as those whose output is part of airborne software and thus can introduce errors. Some examples of development tools are compilers, automated code generators, linkers, graphical user interface (GUI) builders, automated database construction, and graphical modeling tools that generate source code. If the output of a development tool is fully verified, then the tool does not have to be qualified to be used. Tool qualification is needed if processes defined in RTCA/DO-178B [20] are eliminated, reduced or automated by the use of a software tool without its output being verified as specified in Section 6.

FAQ #62 deals with test software. The question concerns the requirements for flight test analysis software and ground-based test software. Flight test analysis software and ground-based test software are software tools used to monitor or analyze performance or compliance data of a system. Flight test analysis software or ground-based test software can be used as a verification tool to automate some verification activities such as requirements-based testing. When used as a verification tool that eliminates, reduces, or automates a verification activity without its output being verified, then the RTCA/DO-178B [20] guidelines for tool qualification are applicable. When the flight test analysis software or ground-based test software is not used for a verification activity, such as when it is used to satisfy a system regulatory requirement, then qualification of the tool is outside the scope of RTCA/DO-178B [20].

DP #1 deals with verification tool selection. The paper states that there are some subtleties related to tool performance that may cause unforeseen problems. It is important that the

functional requirements of the tool should accurately reflect the corresponding objective. When selecting a tool, the first process should be how well the required objectives of RTCA/DO-178B [20] can be satisfied by the tool. The RTCA/DO-178B [20] objectives must be readily understood. If they are not, it is not possible to correctly determine that the tool has implemented the objectives correctly. The interpretation of a tool's output is based upon the user's understanding of the tool's function. If an incorrect assumption has been made about the tool's function, the interpretation of its output may be equally flawed. When selecting a tool it is essential to understand its limitations so that additional manual efforts may augment the tool to complete the verification process.

DP #11 deals with tool qualification using its service history. The tool's service history can be used to support or replace certain objectives for tool qualification in Sections 12.2 and 12.3.5 of RTCA/DO-178B [20]. The use of service history allows some aspects of Section 12.2 to be accomplished based on data that may have been gathered during the tool's usage. Tool qualification using the tool's service history may be useful when the tool is in use but has not been qualified. Product service history concerns the use of embedded software in airborne environment. Tool service history concerns the use of a software development or verification tool. Product service history is concerned with the operation and number of flight hours. Tool service history is concerned with the tool's compliance with that tool's operational requirements during its usage. The tool qualification plan should explain the methods which will be used to qualify the tool, including those objectives and guidance which will be met using a service history approach. The DP continues with a table showing whether service history can be used to satisfy the objectives or guidelines for each subparagraph of RTCA/DO-178B [20] Section 12.2. Not all objectives of Section 12.2 can be met using tool service history. When using product service history for tool qualification, the guidance of Section 12.3.5 will be used to show compliance with Section 12.2.

4.1.2.10 RTCA/DO-178C Discussion

Since 1992, the aviation industry and certification authorities around the world have used the considerations in RTCA/DO-178B [20] as an acceptable means of compliance for software approval in the certification of airborne systems and equipment. As experience was gained in the use of RTCA/DO-178B [20], questions arose regarding the document's content and application. Some of these questions were addressed in the development of RTCA/DO-248B [27]. However, RTCA/DO-248B [27] only provided clarifications of RTCA/DO-178B [20] and did not provide any additional guidance for compliance. Advances in hardware and software technology resulted in software development methodologies and issues that were not adequately addressed in RTCA/DO-178B [20]. RTCA/DO-178C [28] is an update to RTCA/DO-178B [20]. One of the updates is related to software tools. Section 12.2 has been rewritten to provide better guidance for determining the impact of tool use which, in turn, determines the tool qualification requirements. Also, all the tool qualification objectives and activities were moved to a separate document, RTCA/DO-330 [31]. RTCA/DO-178C [28] has not yet been endorsed by the FAA; Order 8110.49 Change 1 [22] still endorses RTCA/DO-178B [20].

Section 2.5 of RTCA/DO-178C [28] states that COTS software is a software-related issue that should be considered by the system life cycle processes. Section 2.5 further states that if deficiencies exist in the software life cycle data of COTS software, the data should be augmented to satisfy the objectives of RTCA/DO-178C [28].

Section 3 of RTCA/DO-178C [28] discusses the software life cycle processes, definition, and transition criteria used in the civil aviation industry. Software life cycle processes include the planning process, the development processes (requirements process, design process, coding process, and integration process), and integral processes (verification process, configuration management process, quality assurance process, and the certification liaison process). Integral processes are performed concurrently with the planning and development processes. The civil aviation industry high-level software life cycle processes discussed in this chapter are simpler than either the IEEE Std. 1074-2006 [4] or IEEE Std. 12207-2008 [2] software life cycle processes and are intended to be compatible with any life cycle model selected for use by the applicant.

Section 4 of RTCA/DO-178C [28] discusses the objectives and activities of the software planning process. This process produces the software plans and standards that direct the software development processes and the integral processes. Section 4.1.c states that one of the objectives of the software planning process is to ensure that the software life cycle environment, including the methods and tools to be used for the activities of each software life cycle process, has been selected and defined. Section 4.2.c states that one activity of the software planning process is to choose methods and tools that aid error prevention and provide defect detection in the software development process. Section 4.2.f states that another activity of the software planning process is when multiple-version dissimilar software is used in a system to mitigate software failures, the process should choose the methods and tools to achieve the dissimilarity necessary to satisfy the system safety objectives. Section 4.2.i states that another activity of the software planning process is that when user-modifiable software is planned, related processes, tools, environment, and data items substantiating the design should be specified in the software plans and standards. Section 4.2.j.3 states that another activity of the software planning process is when parameter data items are planned, the processes to develop, verify, and modify parameter data items, and tool qualification should be addressed. Section 4.3.b.3 states that the activities for the software plans should define the transition criteria by specifying availability of tools, methods, plans, and procedures.

Section 4.4.1.b states that the use of tools or combinations of tools as parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together. Section 4.4.1.d states that if certification credit is sought for use of the tools in combination, the sequence of the options should be examined and specified in the appropriate plan. Section 4.4.1.e states that if optional features of software tools are chosen for use in a project, the effects of the options should be examined and specified in the appropriate plan. This is especially important for compilers and autocode generators. Section 4.4.1.f states that known tool problems and limitations should be assessed and those issues which can adversely affect airborne software should be addressed.

With respect to compiler and language considerations, Section 4.4.2.a states that some compilers have features intended to optimize performance of the object code. If the test cases give coverage consistent with the software level, the correctness of the optimization need not be verified. Otherwise, the impact of these features on structural coverage analysis should be determined (Section 6.4.4.2). Section 4.4.2.b says the implementation of certain compiler features may produce object code that is not directly traceable to the source code, for example, initialization, built-in error detection, or exception handling. The software planning process should provide a means to detect this object code and to ensure verification coverage, and should define the means in the appropriate plan. Section 4.4.2.c states that if a new compiler, linkage editor, or loader version is introduced, or compiler options are changed during the software life cycle, previous tests and coverage analyses may no longer be valid. The verification planning should provide a means of reverification that is consistent with Sections 6 and 12.1.3.

Section 5 of RTCA/DO-178C [28] discusses the objectives and activities of the software development processes including the software requirements process, the software design process, the software coding process, and the software integration process. During the software design process when user-modifiable software is included in the project, the non-modifiable component should be protected from the modifiable component to prevent interference in the safe operation of the non-modifiable component. This protection can be enforced by hardware, by software, by the tools used to make the change, or by a combination of the three. If the protection is provided by a tool, the tool should be categorized and qualified as defined in Section 12.2. Compiling, linking and loading are part of the software integration process. Section 5.4.3.d states that use of autocode generators should conform to the constraints defined in the planning process.

Section 6 of RTCA/DO-178C [28] discusses the objectives and activities of the software verification process. The purpose of the software verification process is to detect and report errors that may have been introduced during the software development processes. Software verification process objectives are satisfied through a combination of reviews, analyses, the development of test cases and procedures, and the subsequent execution of those test procedures. Verification independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified. A tool may be used to achieve equivalence to the human verification activity.

Section 7 of RTCA/DO-178C [28] discusses the objectives and activities of the SCM process. Software life cycle environment tools are defined by the software planning process and identified in the software life cycle environment configuration index. Configuration identification should be established for the executable object code and of the tools used to develop, control, build, verify, and load the software. The SCM process for controlling tools should comply with the objectives associated with Control Category 1 or 2 depending on whether the tool was qualified, the life cycle process that used the tool, and the software level assigned to the tool.

Data is produced during the software life cycle to plan, direct, explain, define, record, or provide evidence of activities. This data enables the software life cycle processes, system or equipment

certification, and post-certification modification of the software product. Section 11 of RTCA/DO-178C [28] discusses the characteristics, form, configuration management controls, and content of the software life cycle data. Data should be unambiguous, complete, verifiable, consistent, modifiable, and traceable. Data is verifiable if it can be checked for correctness by a person or tool. The PSAC is the primary means used by the certification authority for determining whether an applicant is proposing a software life cycle that is commensurate with the rigor required for the level of software being developed. This plan should include a description of specific considerations that may affect the certification process including alternative methods of compliance, tool qualification, previously developed software, option-selectable software, user-modifiable software, deactivated code, COTS software, field-loadable software, parameter data items, multiple-version dissimilar software, and product service history.

Section 11 states that the software development plan (SDP), the software verification plan (SVP), the software configuration management plan, and the software quality assurance plan should identify methods and tools to be used, the coding methods, programming language, coding tools to be used, and the hardware platforms for the tools to be used. The SVP should also include a description of the assumptions made by the applicant about the correctness of the compiler, linkage editor, or loader. Section 11 also states that the software requirements standards, the software design standards, and the software code standards should include any constraints on the use of tools used for requirements development, design, and coding. Section 11.15 discusses the software life cycle environment configuration index. This index identifies the configuration of the software life cycle environment and should identify the tools to be used during the development of the software, identify the software testing and analysis tools, and identify qualified tools and their associated tool qualification data. Section 11.16 discusses the software configuration index. This index identifies the configuration of the software product and should identify procedures, methods, and tools for making modifications to the user-modifiable software, if any, and should identify instruction for building the executable object code including instructions for compiling and linking.

Section 12 provides guidance on additional considerations where objectives and/or activities may replace, modify, or add to some or all of the objectives and/or activities defined in the rest of this document. Section 12.1.3 discusses changes of an application or development environment. If a new development environment uses software tools, the tool may need to be qualified per Section 12.2. Using a different autocode generator or a different set of autocode generator options may change the source code; therefore, the impact of these changes should be analyzed. If a different compiler or options are used, previous verification may not be valid and should not be used for the new application. Section 12.2 discusses tool qualification. The discussion in RTCA/DO-178C [28] has been significantly simplified from RTCA/DO-178B [20]. Most of the details regarding tool qualification are now contained in a separate document, RTCA/DO-330 [31]. The content of Section 12.2 in this document provides a high-level summary of tool qualification as a supplement to RTCA/DO-330 [31]. Section 12.2.1 states that qualification of a tool is needed when processes of this document are eliminated, reduced, or automated by use of a software tool without its output being verified. The purpose of tool qualification is to ensure that the tool provides confidence at least equivalent to that of the

process(es) eliminated, reduced, or automated. A tool is qualified only for use on a specific system where the intention to use the tool is stated in the PSAC. Section 12.2.2 states that if tool qualification is needed, the impact of the tool use in the software life cycle processes should be assessed in order to determine its tool qualification level (TQL). RTCA/DO-178C [28] states that the following criteria should be used to determine the impact of a tool:

Criteria 1: A tool whose output is part of the airborne software and thus could insert an error.

Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:

1. Verification process(es) other than that automated by the tool, or
2. Development process(es) that could have an impact on the airborne software.

Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error.

As discussed in RTCA/DO-178C [28], the following criteria from Sections 2.3.2 and 2.3.3 of RTCA/DO-178C [28] should be used to determine the software level of the tool. The software level of the tool should be consistent with the software level of the software component associated with the tool based upon the failure condition which may result from anomalous behavior of the software as determined by the system safety assessment.

Level A: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft which would result in multiple fatalities, usually with the loss of the airplane.

Level B: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a hazardous failure condition for the aircraft which would reduce the capability of the airplane or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be:

1. A large reduction in safety margins or functional capabilities
2. Physical distress or excessive workload such that the flight crew cannot be relied upon to perform their tasks accurately or completely or
3. Serious of fatal injury to a relatively small number of the occupants other than the flight crew.

Level C: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a

significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to the flight crew, or physical distress to passengers or cabin crew, possibly including injuries.

Level D: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft which would not significantly reduce airplane safety, and which involve crew actions that are well within their capabilities including, for example, a slight reduction in safety margins or functional capabilities, a slight increase in crew workload, such as routine flight plan changes, or some physical discomfort to passengers or cabin crew.

Level E: Software whose anomalous behavior would cause or contribute to a failure of system function with no effect on aircraft operational capability, safety or crew workload. If a software component is determined to be Level E and this is confirmed by the certification authority, no tool qualification is required.

Based on the impact of a tool as specified by one of the above criteria, and the software level as specified by one of the above levels, the TQL can be determined from the following table. TQL-1 is the most rigorous level and TQL-5 is the least rigorous level. The objectives, activities, guidance, and life cycle data required for each TQL are described in RTCA/DO-330 [31]. The life cycle data required for each TQL is assigned as either Control Category 1 or 2 data depending on the software level of the tool and the life cycle process using the tool. The software configuration management processes associated with Control Category 1 or 2 data must then be followed to control the qualified tool.

Table 4.2 – Tool Qualification Level Determination

Software Level	Criteria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5
E	No tool qualification necessary		

Section 12.3 discusses alternative methods of complying with the objectives and activities of this document. Methods include (1) exhaustive input testing, (2) multiple-version dissimilar software verification, (3) software reliability models, and (4) product service history. With respect to item (2), if multiple-version dissimilar software is used, the tool qualification process may be modified, if evidence is available that the multiple software tools used in the software development process are dissimilar. The applicant should show that each tool was obtained from a different developer and show that each tool has a dissimilar design. If separate, dissimilar simulators are used to verify multiple-version dissimilar software versions, then the approach to tool qualification of the simulators may be modified. Unless it can be justified as unnecessary for multiple simulators to be dissimilar, the applicant should provide evidence that

each simulator was developed by a different team, provide evidence that each simulator has different requirements, a different design, and a different programming language, and provide evidence that each simulator executes on a different processor.

4.1.2.11 RTCA/DO-248C Discussion

RTCA/DO-248C [29] provides no new or additional guidance beyond the guidance in RTCA/DO-178C [28] and is an update to RTCA/DO-248B [27]. This document provides clarification of the guidance material in RTCA/DO-178C [28] and RTCA/DO-278A [30]. RTCA/DO-278A [30] is a document that provides guidance similar to RTCA/DO-178C [28] for the communications, navigation, surveillance, and air traffic management (CNS/ATM) ground-based community. RTCA/DO-248C [29] resolves inconsistencies between RTCA/DO-178C [28] and any other relevant civil aviation standards. The clarification material is comprised of 84 frequently asked questions (FAQs) providing short and concise responses to questions posed to certification authorities or others who provide interpretation of RTCA/DO-178C [28], 21 discussion papers (DPs) providing clarification of certain sections of RTCA/DO-178C [20] in cases where the clarification requires more than a short answer to a question, and 13 rationale providing reasoning for the material in RTCA/DO-178C [20]. The rationale section is new to RTCA/DO-248C [29] and does not exist in RTCA/DO-248B [27]. The first 76 FAQs are identical to those in RTCA/DO-248B [27] and were not surveyed.

The new FAQs and DPs are not relevant to software tools, COTS software, or life cycles and will not be discussed. Two of the rationales are relevant.

Section 5.3 of RTCA/DO-248C [29] contains the rationale for the material in Section 3 of RTCA/DO-178C [28] which deals with selecting a life cycle model. Section 3 of RTCA/DO-178C [28] allows for a software life cycle to be defined with any suitable life cycle model(s) to be chosen for software development. Specific transition criteria between one process and the next are not prescribed, rather RTCA/DO-178C [28] states that transition criteria should be defined and adhered to throughout the development life cycle(s) selected. The approach documented in Section 3 of RTCA/DO-178C [28] allows the accommodation of different life cycles.

Section 5.13 of RTCA/DO-248C [29] contains the rationale for removing tool qualification objectives and activities from RTCA/DO-178C [28] and relying on a supplement, RTCA/DO-330 [31]. Supplements were implemented to provide guidelines for technology advances in software engineering. Each supplement adds, modifies, or deletes the RTCA/DO-178C [28] objectives to make the guidance more clear. Tools are widely used to develop, verify, configure, and control software. A tool is a computer program or a functional part thereof, used to help develop, transform, test, analyze, produce, or modify another program, data, or its documentation. Examples are an automated code generator, a compiler, test tools, and modification management tools. RTCA/DO-330 [31] explains the process and objectives for qualifying tools where RTCA/DO-178C [28] Section 12.2 provides the criteria to determine if a tool needs to be qualified.

4.1.2.12 RTCA/DO-330 Discussion

A detailed view of software tools is given in RTCA DO-330 [31]. RTCA DO-330 [31] presents an entire qualification process for various tool qualification levels (TQL) that roughly correspond to safety levels for the NRC where TQL-1 would correspond to safety significant and TQL-5 would correspond to non-safety components. The qualification process presented contains stages similar to software development life cycle stages including tool qualification planning, tool development life cycle processes, tool verification, and so on. A significant amount of information can be obtained from this document.

This document contains a “Tool Life Cycle Process” which is roughly similar to the software life cycle processes found in IEEE Std. 1074-2006 [4] and IEEE Std. 12207-2008 [2]. Included in the life cycle are four stages referred to as “Integral Processes.” The process steps are tool qualification planning, tool development, tool verification, tool configuration management, tool quality assurance, certification liaison process to qualify tools, tool qualification data, and additional considerations for tool qualification. The processes beginning with verification and ending with certification liaison are considered the integral processes.

The document describes the purpose of tool qualification as avoiding the risk that an error in a tool that may have a negative impact on software functionality by having the tool developed and verified using adequate processes. It further describes the process as obtaining confidence in the tool functionality.

The qualification level of a tool is based on the tool use and its potential impact on the software development life cycle. The document references RTCA/DO-178C [28] for determination of the qualification level. The stakeholders in the qualification process, tool user and tool developer, will address where in the development process the tool will be used.

Within the tool life cycle process, the qualification planning process identifies the tools to be used and their impact on the software lifecycle and the development, verification, and integral processes are described. Documents generated during this process include the tool qualification plan, tool development plan, tool verification plan, tool configuration management plan, and the tool quality assurance plan. These documents are similar in content to the documents described in IEEE standards.

The tool development process includes tool requirements, tool design, tool coding, and tool integration. These correspond to the software life cycle processes of IEEE Std. 12207-2008 [2]. The guidance provided on software requirements provides a software tool-specific process similar to that described in IEEE Std. 830-1998 [59]. The tool design process and tool design description document are similar to that described in IEEE Std. 1016-2009 [115]. The tool coding process has no direct correlation to any single IEEE document but is similar to the software construction process described in IEEE Std. 12207-2008 [2]. The tool integration process is also similar to the software integration process of IEEE Std. 12207-2008 [2].

The tool operational verification and validation process is a software tool-specific process similar to the verification and validation process described in IEEE Std. 1012-2012 [79]. Key concepts

in this section include traceability to requirements and testing. Tool testing and test cases are also discussed in this section. Compliance with requirements and robustness are key concepts.

The tool configuration management process described is similar to the processes described in IEEE Std. 1042-1987 [46] and IEEE Std. 828-2012 [75]. It also reinforces the need for configuration management of software tools as described in IEEE Std. 7-4.3.2-2010 [1]. Configuration management activities include configuration identification, baselines and traceability, problem reporting and tracking, change control, change review, configuration status accounting, and archive, retrieval and release.

The tool quality assurance process described in the document has some of the requirements required by 10 CFR Part 50, Appendix B. However, concepts such as independence are not emphasized though they are mentioned. In general, the process described is similar to that described in IEEE Std. 730-2002 [116] without the depth of coverage required in the documentation.

The remaining sections of the document provide more detail in the interface between the qualifier and the qualifying authority as well as additional detail on some of the documentation described above.

Chapter 11, “Additional Considerations for Tool Qualification,” discusses multi-function tools, previously qualified tools (including reuse, changes to the operational environment, and changes to the qualified tool), qualifying COTS software tools, analysis of service history, and alternative methods for tool qualification.

Annex A discusses tool qualification objectives and provides a reference table showing the applicability of each qualification objective by paragraph and its applicability by TQL level. The table also references the documentation required by each objective and the control category. Appendix B provides an example of the qualification process while Appendices C and D provide answers to frequently asked questions.

4.1.2.13 FAA Order 1370.109 Discussion

FAA Order 1370.109 [32] establishes a security software assurance policy for the FAA to protect the confidentiality, integrity, and availability of FAA information systems. The system development lifecycle (SDLC) described in NIST Special Publication (SP) 800-64 [114] is the basis for the software assurance methodology described in this order. This approach to systems development leads to well documented systems that are easier to test and maintain.

This policy includes guidance for utilizing approved tools purchased for Agency-wide use and sets the criteria for evaluating testing tools that identify software deficiencies so that defects are uncovered and remediated. The scope of this policy includes examination of software as well as execution of that software code in various environments and conditions. This policy does not apply to utility software.

This order has some high-level requirements for software tools. First, vendor technologies used to perform software assurance assessments must be referenced in the NIST software

assurance metrics and tool evaluation list. Second, it is the responsibility of the CIO to provide funding for enterprise tools and/or services for software assurance assessments, maintain a list of approved software code services and/or tools, and to utilize approved software code services and/or tools that work in FAA's existing environment.

4.1.3 NRC and Commercial Nuclear Power Industry Documents Discussion

4.1.3.1 RG 1.152 Revision 3 Discussion

Regulatory Guide (RG) 1.152, Revision 3 [34], describes a method acceptable to NRC staff for complying with regulations for "promoting high functional reliability, design quality, and a secure development and operational environment (SDOE) for the use of digital computers in the safety systems of nuclear power plants." Hardware, software, firmware, and interfaces are addressed in this regulatory guide. The regulatory position is based on the waterfall life cycle model with a discussion of each phase of this model.

Section 2.2, "Requirements Phase," discusses pre-developed and COTS software: Specifically, this section states, in part: "Requirements specifying the use of pre-developed software and systems (e.g., reused software and commercial off-the-shelf (COTS) systems) should address the reliability of the safety system (e.g., by using pre-developed software functions that have been tested and are supported by operating experience)." The introduction of unnecessary or extraneous requirements that may result in inclusion of unwanted or unnecessary code should be prevented in the requirements phase.

Section 2.3, "Design Phase," states that with respect to software tools, design items that use pre-developed software should address how this software will not challenge the secure operational environment. Measures should be taken to prevent the introduction of unnecessary design features that may result in unwanted or unnecessary code.

Section 2.4, "Implementation," discusses implementation of COTS software and states that COTS software is likely proprietary and unavailable for review and its behavior may be indeterminate. The developer should ensure that COTS software features do not compromise the secure operational environment that would degrade the reliability of the safety system.

The conclusions reached from reading RG 1.152 [34] are that a secure operational and development environment must be implemented to prevent unnecessary and unauthorized code from being added to the safety system with specific attention given to COTS and pre-developed software.

4.1.3.2 RG 1.168 Revision 1 Discussion

RG 1.168, Revision 1 [38], describes a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to quality assurance programs as applied to software V&V and with respect to conducting audits, inspections, walkthroughs, and technical and management reviews.

There are only a few references to safety-related software tools in this regulatory guide. The first is in Section C.3, paragraph 2 which states that when a licensee acquires a commercial-grade product approved by the NRC staff for implementation in a safety-related system, the applicant or licensee is not relieved of the responsibility of assuring that the V&V and subsequent software quality satisfy the NRC's requirements for reliability including the independence of software V&V by a separate organization not subject to the financial, resource or scheduler limitations of the developer. The extent of independence between the organization responsible for design and the organization responsible for verification and checking of the design must be verified by the applicant or licensee to meet the NRC's requirements contained in 10 CFR Part 50, Appendix B.

Section C.4 states that IEEE Std. 1012-1998 [39] provides guidance for retrospective V&V of software that was not verified under the standard including COTS software. The use of this guidance for the acceptance of pre-existing safety system software not verified during development to the provisions of this regulatory guide or its equivalent is not endorsed. RG 1.152 [34] provides information on the acceptance of pre-existing software.

Finally, Section 6 discusses software tools used in the development of safety system software . As endorsed by RG 1.152 [34], IEEE Std. 7-4.3.2-1993 [69] states: “V&V tasks of witnessing, reviewing, and testing are not required for software tools, provided the software that is produced using these tools is subject to V&V activities that will detect flaws introduced by the tools.” If this cannot be demonstrated, the provisions of this regulatory guide are applicable.

4.1.3.3 RG 1.168 Revision 2 Discussion

Proposed RG 1.168, Revision 2 [41] updates its endorsement of IEEE Std. 1012-1998 [39] to IEEE Std. 1012-2004 [42] as a method that is acceptable to the NRC staff for meeting the requirements of 10 CFR Part 50 as they apply to verification and validation of safety system software. IEEE Std. 1012-2004 [42] describes the process of software V&V, including elements of a software V&V plan, and describes a minimum set of V&V activities for software at different integrity levels.

Proposed Revision 2 of this regulatory guide also updates its endorsement of IEEE Std. 1028-1997 [40] to IEEE Std. 1028-2008 [43] as an approach acceptable to the NRC staff for carrying out software reviews, inspections, walkthroughs, and audits. Reviews and audits are closely associated with V&V activities since technical reviews and audits are frequently conducted by the V&V organization and because the V&V organization normally participates in management reviews. Because of this close connection of the V&V activity with reviews and audits, this proposed regulatory guide addresses IEEE Std. 1012-2004 [42] and IEEE Std. 1028-2008 [43] together.

IEEE Std. 1028-2008 [43] adds an anomaly ranking and reporting scheme which should be beneficial and useful to the applicants and licensees when addressing the anomaly reporting documentation requirements found in IEEE Std. 829-2008 [52].

IEEE Std. 1012-2004 [42] adds a conditional independence option in Annex C. IEEE Std. 1012-2004 [42] also recommends use of the software integrity scheme, while IEEE Std. 1012-1998 [39] did not require the selection of an integrity level. This proposed regulatory guide expands on the topic of independence because the new concept of conditional independence in Annex C, Table C.1 is not acceptable to the NRC. This proposed regulatory guide also provides clarification for organizational mapping as per IEEE Std. 1012-2004 [42], Annex F.

IEEE Std. 1012-2004 [42] adds a new activity called “Security Analysis.” With respect to “Security Analysis,” this proposed regulatory guide finds this new activity consistent with a life-cycle approach and has altered the NRC position in Part C, 7.c, to what is now called a “Secure Analysis.” Clause 4 of IEEE Std. 1012-2004 [42] defines an acceptable four-level method of quantifying software integrity levels, in which level 4 is the highest and level 1 is the lowest. Software used in nuclear power plant safety systems should be assigned integrity level 4 or the equivalent, as demonstrated by a mapping between the applicant or licensee approach and integrity level 4 as defined in IEEE Std. 1012-2004 [42].

There are still only a few references to safety-related software tools in this regulatory guide. The first is in Section C.3, paragraph 3 and has not changed from the previous version 1 of this regulatory guide. This section states that when a licensee acquires a commercial-grade product approved by the NRC staff for implementation in a safety-related system, the applicant or licensee is not relieved of the responsibility of assuring that the V&V and subsequent software quality satisfy the NRC's requirements for reliability including the independence of software V&V by a separate organization not subject to the financial, resource or scheduler limitations of the developer. The extent of independence between the organization responsible for design and the organization responsible for verification and checking of the design must be verified by the applicant or licensee to meet the NRC's requirements contained in 10 CFR Part 50, Appendix B.

Section C.4 states that IEEE Std. 1012-2004 [42] provides guidance for retrospective V&V of software that was not verified under the standard including COTS software. The use of this guidance for the acceptance of pre-existing safety system software not verified during development to the provisions of this regulatory guide or its equivalent is not endorsed. RG 1.152 [34] provides information on the acceptance of pre-existing software. This information is effectively unchanged from the previous version of the regulatory guide.

Finally, Section 6 still discusses software tools used in the development of safety system software. The endorsement of IEEE Std. 7-4.3.2-1993 [69] in the last revision of RG 1.168 [38] has been updated to IEEE Std. 7-4.3.2-2003 [35]. This section states that tools used in the development of safety system software should be handled according to IEEE Std. 7-4.3.2-2003 [35], as endorsed by RG 1.152 [34]. However, the V&V tasks of witnessing, reviewing, and testing are not required for software tools, provided the software that is produced using these tools is subject to V&V activities that will detect flaws introduced by the tools. If this cannot be demonstrated, the provisions of this regulatory guide are applicable to the development of tools.

4.1.3.4 RG 1.169 Discussion

RG 1.169 [44] describes a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to the maintenance of appropriate controls and records of software development activities.

Section C.6 applies to software tools. This section discusses documentation requirements. For safety system software, configuration items or controlled documents should include support software used in development, test code used in testing, and commercial software that are safety system software. Items that may not change but are necessary to ensure correct software production, such as compilers, should also be configuration items, thereby ensuring that all factors contributing to the executable software are understood.

Section C.7 discusses control of purchased materials including COTS software. Contractually developed or qualified commercial software products that are safety system software must be taken under control by an SCM program that complies with IEEE Std. 828-1990 [45] as endorsed by this regulatory guide. This means, that the exact version of the product is identified and controlled according to the change control procedures applied to other configuration items and that its usage is tracked and reported.

Section C.8 discusses development tools. Tools used in the development of safety system software should be handled according to IEEE Std. 7-4.3.2-1993 [69] as endorsed by RG 1.152 [34]. Tools must be taken under control, i.e., treated as a configuration item, by an SCM program operated by the using organization.

4.1.3.5 RG 1.169 Revision 1 Discussion

Proposed RG 1.169, Revision 1 [47] updates the endorsement of IEEE Std. 828-1990 [45] to IEEE Std. 828-2005 [48] as a method that is acceptable to the NRC staff for meeting the requirements of General Design Criterion (GDC) 1 in Appendix A to 10 CFR Part 50 with respect to configuration management plans for safety system software. IEEE Std. 828-2005 [48] describes the software industry approaches to SCM that are generally accepted in the software engineering community.

This proposed revision to RG 1.169 [47] addresses two new areas of IEEE Std. 828-2005 [48]; neither of which is directly relevant to the regulation, guidance or review/approval practice of software tools. The first area is related to the overall software release management objectives that complement the existing standard work found in the software life-cycle project plan. The second area is related to security features or mechanisms that can play a critical role in supporting software security at higher levels of assurance.

Section C.6 still applies somewhat to software tools. This section discusses documentation requirements. For safety system software, configuration items or controlled documents should include support software used in development and test code used in testing. The identification of commercial software items as a configuration item has been deleted from this proposed revision. Still included in this revision as configuration items are items that may not change but

are necessary to ensure correct software production, such as compilers, thereby ensuring that all factors contributing to the executable software are understood.

Section C.7 still discusses control of purchased materials including COTS software. Contractually developed or qualified commercial software products that are safety system software must be taken under control by an SCM program that complies with IEEE Std. 828-2005 [48] as endorsed by this regulatory guide. This means, that the exact version of the product is identified and controlled according to the change control procedures applied to other configuration items and that its usage is tracked and reported. Section C.7 also endorses EPRI TR-106439 [106] for guidance on dedicating commercial-grade digital equipment for use in nuclear power plant safety applications.

Section C.8 still discusses development tools; however, the endorsed IEEE standard has been updated. Tools used in the development of safety system software should be handled according to IEEE Std. 7-4.3.2-2003 [35] as endorsed by RG 1.152 [34]. Tools must be taken under control, i.e., treated as a configuration item, by an SCM program operated by the using organization.

4.1.3.6 RG 1.170 Discussion

RG 1.170 [49] discusses software test documentation for digital computer software used in safety systems of nuclear power plants. This regulatory guide endorses IEEE Std. 829-1983 [50] with exceptions for methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. In particular, the methods are consistent with GDC and the criteria for quality assurance programs of Appendix B as they apply to the documentation of software testing activities.

With respect to software tools used during testing, the only relevant statement in this regulatory guide is an exception to IEEE Std. 829-1983 [50]. The IEEE standard states that not all documentation is necessary for all test phases. This regulatory guide states that if not all test documentation is prepared, that there is a minimum set of test documentation that must accompany all tests including environmental conditions and special controls, equipment, tools, and instrumentation needed for accomplishing the testing.

4.1.3.7 RG 1.170 Revision 1 Discussion

Proposed RG 1.170, Revision 1 [51] updates the endorsement of IEEE Std. 829-1983 [50] to IEEE Std. 829-2008 [52] as a method that is acceptable to the NRC staff for meeting the requirements of GDC in Appendix A to 10 CFR Part 50 with respect to documentation of software testing activities. IEEE Std. 829-1983 [50] provided a linear planning, testing, and reporting perspective. The updated IEEE Std. 829-2008 [52] expands the depth of its function with both a classification of the intensity based on the seriousness of the incorrect behavior consequences and life-cycle time line for the system and software project development.

This proposed RG 1.170, Revision 1 [51] addresses the expansion of IEEE Std. 829-2008 [52] by updating the existing sections starting with the “Test Program” and then adding three new

staff regulatory guidance positions to the regulatory guide. New Staff Regulatory Guidance Position No. 8 points out that repetition is sometimes needed when automated tools are used for testing and that there is a need for making test information easily accessible to the NRC staff to improve the timeliness of a safety conclusion.

With respect to software tools used during testing, Section C.1.c states:

> "The additional information on environmental conditions and special controls, equipment, tools, and instrumentation needed for accomplishing the testing can also be supplemented under the Level Test Plan within Clause 9.3.2 and the Level Test Design, Clause 10.2.2."

Section C.8 discusses an NRC exception to Clause 6.3 of IEEE Std. 829-2008 [52] which states that there is no need for repetition of test information if completely managed by an automated tool with references for tracing the information. The NRC staff takes exception to this clause, as there are particular cases where electronic validation methods, with repetition, provide test information easily accessible for the basis of any safety conclusion. This section further states that this test information should be available within the record management system and that tools used in the development of safety system software should be handled according to IEEE Std. 7-4.3.2-2003 [35], as endorsed by RG 1.152 [34].

4.1.3.8 RG 1.171 Discussion

RG 1.171 [53] discusses software unit testing for digital computer software used in safety systems of nuclear power plants. This regulatory guide endorses IEEE Std. 1008-1987 [54] with exception as it applies to software unit testing. This IEEE standard defines a method for planning, preparing for, conducting, and evaluating software unit testing.

The only statement of relevance to software tools in this regulatory guide is that the documentation used to support software unit testing must include as a minimum environmental conditions and special controls, equipment, tools, and instrumentation needed for the accomplishment of testing.

4.1.3.9 RG 1.171 Revision 1 Discussion

Proposed RG 1.171, Revision 1 [55] discusses software unit testing for digital computer software used in safety systems of nuclear power plants. This regulatory guide still endorses IEEE Std. 1008-1987 [54] with exception as it applies to software unit testing. IEEE Std. 1008-1987 [54] was reaffirmed in 2002. Since 1987, various software-related standards have been developed or updated and Revision 1 of RG 1.171 [55] has been updated to reference these later standards and to be consistent with these standards and guidance.

Revision 1 of RG 1.171 [55] states that Section 1.1 of IEEE Std. 1008-1987 [54] mandates the use of the Test Design Specification and the Test Summary Report documents, which can be found in IEEE Std. 829-2008 [52], Clauses 10, "Level Test Design," and 17, "Master Test Report," respectively. In addition, IEEE Std. 1008-1987 [54] either incorporates additional information into these two documents or indicates the need for additional documentation.

Regardless of whether these two documentation formats are used, Revision 1 of RG 1.171 [55] requires that the documentation used to support software unit testing should include information necessary to meet regulatory requirements as applied to software test documentation. As a minimum, this information should include special conditions and controls, equipment, tools, and instrumentation needed for the accomplishment of testing.

4.1.3.10 RG 1.172 Discussion

RG 1.172 [56] discusses software requirements specifications (SRSs) for digital computer software used in safety systems of nuclear power plants. This regulatory guide endorses IEEE Std. 830-1993 [57] with exception as a method of complying with GDC 1 and the criteria for quality assurance programs in Appendix B as they apply to the development of software requirements specifications. The endorsed IEEE Std. 830-1993 [57] describes current practice for writing software requirements specifications for a wide variety of systems. It is not specifically aimed at safety applications; however, it does provide guidance on the development of software requirements specifications that will exhibit characteristics important for developing safety system software.

This regulatory guide states that Sections 4.3.2.2 and 4.3.2.3 of IEEE Std. 830-1993 [57] discuss the use of specification or representation tools to generate requirements. The NRC requires that traceability should be maintained between these representations and the natural language descriptions of the software requirements that are derived from systems requirements and system safety analyses.

4.1.3.11 RG 1.172 Revision 1 Discussion

Proposed RG 1.172, Revision 1 [58] discusses SRSs for digital computer software used in safety systems of nuclear power plants. This regulatory guide updates its endorsement of IEEE Std. 830-1993 [57] to IEEE Std. 830-1998 [59] even though IEEE Std. 830-1998 [59] has been superseded by IEEE Std. 29148-2011 [60]. The regulatory guide endorses IEEE Std. 803-1998 [59] with exception as a method of complying with GDC 1 and the criteria for quality assurance programs in Appendix B to 10 CFR Part 50 as they apply to the development of software requirements specifications.

There are only minor changes between IEEE Std. 830-1998 [59] and IEEE Std. 830-1993 [57]. This version of RG 1.172 [58] specifically addresses Annex B of IEEE Std. 830-1998 [59] and corrects three other perspectives associated with the software attributes of a software requirements specification. None of the updates in Revision 1 of RG 1.172 [58] involve software tools.

The only reference to software tools in Revision 1 of RG 1.172 [58] is in a discussion of Sub-Clauses 4.3.2.2 and 4.3.2.3 of IEEE Std. 830-1998 [59]. These two sub-clauses discuss the use of specification or representation tools to generate requirements. This version of the regulatory guide continues to require that traceability be maintained between these representations and the natural language descriptions of the software requirements that are

derived from systems requirements and system safety analyses to meet the requirements in GDC 1 of Appendix A to 10 CFR Part 50.

4.1.3.12 RG 1.173 Discussion

RG 1.173 [61] describes a method acceptable to the NRC staff for satisfying the Commission's regulations with respect to software development processes.

This regulatory guide endorses IEEE Std. 1074-1995 [62] which describes, in terms of inputs, development, verification or control processes, and outputs, a set of processes and constituent activities that are commonly accepted as composing a controlled and well-coordinated software-development process. It describes inter-relationships among activities by defining the source activities that produce the inputs and the destination activities that receive the outputs. IEEE 1074-1995 [62] can be used as a basis for developing specific software life cycle processes that are consistent with regulatory requirements, as applied to software, for controlling and coordinating the design of safety system software.

Section C.1.3 of this regulatory guide states that if pre-existing software (i.e., reusable software or COTS software) is incorporated into a safety system developed under the method described by this regulatory guide, an acceptance process must be included at an appropriate point in the life cycle model to establish the suitability of the pre-existing software for its intended use. Revision 3 of RG 1.152 [34] and EPRI TR 106439 [106] discuss acceptance processes for commercial-grade (COTS) software evaluation.

4.1.3.13 RG 1.173 Revision 1 Discussion

Proposed RG 1.173, Revision 1 [6] describes a method acceptable to the NRC staff for satisfying the Commission's regulations with respect to software development processes. This regulatory guide updates the endorsement of IEEE Std. 1074-1995 [62] to IEEE Std. 1074-2006 [4]. IEEE Std. 1074-2006 [4] provides guidance acceptable to the NRC staff for describing, in terms of inputs, development, verification or control processes, and outputs, a set of processes and constituent activities that are commonly accepted as composing a controlled and well-coordinated software project life cycle process.

In response to a new section called “Determine Security Objectives” in IEEE Std. 1074-2006 [4], Revision 1 of RG 1.173 [6] has also added a new Subsection Part C.1.d, “Secure Analysis,” and acknowledges that planning for security and confirming the security accreditations is necessary and part of a secure analysis. RG 1.173, Revision 1 [6], has also added a clarification statement under Part C.4.d, “System Transitions” to highlight one of the existing planning activities in IEEE Std. 1074-2006 [4].

Revision 1 of RG 1.173 [6] does not change its position on software tools or commercial software. Section C.1.c of this regulatory guide still states that if pre-existing software (i.e., reusable software or COTS software) is incorporated into a safety system developed under the method described by this regulatory guide, an acceptance process must be included at an appropriate point in the life cycle model to establish the suitability of the pre-existing software for its intended use. Revision 3 of RG 1.152 [34] provides information on the acceptance of

preexisting software and with the July 17, 1997 NRC endorsement, additional detailed information on acceptance processes appear in EPRI TR 106439 [106].

4.1.3.14 Digital I&C Interim Staff Guidance 06 Discussion

Interim Staff Guidance (ISG)-06 [63] describes the licensing process that may be used in the review (against licensing criteria – the Standard Review Plan, NUREG-0800) of license amendment requests associated with digital Instrumentation and Control (I&C) system modifications in operating plants originally licensed under Part 50.

The ISG clarifies the NRC position that safety-related software, developed under a high-quality process, can be qualified under the commercial dedication processes of EPRI-TR-106439 [106]. It also iterates that "verification and validation activities common to software development in digital systems to be a critical characteristic that can be verified as being performed correctly following the completion of the software development by conducting certain dedication activities such as audits, examinations, and tests." With respect to software tools, this position implies that COTS and pre-developed tools can be commercially dedicated using similar procedures.

4.1.3.15 NUREG-0800 Branch Technical Position (BTP) 7-14 Discussion

BTP 7-14 [64] in Chapter 7 of NUREG-0800, the Standard Review Plan (SRP), provides the acceptance criteria for review of the structure and content of documentation associated with the software development process of safety-related software. Since, according to several references, software tools should be reviewed with the same rigor as the software they are designed to produce, these acceptance criteria would be the same criteria used to review documentation associated with development of software tools.

With respect to software tools, BTP 7-14 [64] states that the resource characteristics that the software development plan should exhibit include methods/tools and standards. Methods/tools require a description of the software development methods, techniques, and tools to be used. The approach to be followed for reusing software should be described. The plan should identify suitable facilities, tools, and aids to facilitate the production, management, and publication of appropriate and consistent documentation and for the development of the software. It should describe the software development environment, including software design aids, compilers, loaders, and subroutine libraries. The plan should require that tools be qualified with a degree of rigor and level of detail appropriate to the safety significance of the software that is to be developed using the tools. Methods, techniques, and tools that produce results that cannot be verified to an acceptable degree or that are not compatible with safety requirements should be prohibited, unless analysis shows that the alternative would be less safe.

4.1.3.16 NUREG-0800 Appendix 7.1-D Discussion

NUREG-0800, Appendix 7.1-D [65], discusses the application of the criteria in IEEE Std. 7-4.3.2-2003 [35] to computer-based safety systems. The criteria contained in IEEE Std. 7-4.3.2-2003 [35], in conjunction with requirements in IEEE Std. 603-1991 [37], establish minimum functional and design criteria for computers used as components of a safety system. Although

intended for digital safety systems, the criteria of IEEE Std. 7-4.3.2-2003 [35] can be applied to any digital I&C system.

One of the first concerns of this document in the area of safety system criteria is that of the single-failure criterion. Software tools are not discussed in this section; however, special care should be taken when reviewing software tool usage, especially compilers and linkers, to ensure that the single-failure criterion is not violated. For example, diverse systems that have been compiled with a single compiler may subject the code output to a common-mode failure mechanism.

Section 5.3 of this appendix discusses software tools and directs the reviewer to IEC 60880-2 [100] Part 2 and IEEE Std. 7-4.3.2-2003 [35], Clause 5.3.2. Additional guidance for interpretation of Clause 5.3.2 is provided in Section 5.3.2 of the appendix. IEC 60880-2 [100] directs that:

> "If, however, it cannot be demonstrated that defects not detected by software tools or introduced by software tool will be detected by verification and validation (V&V) activities, the software tool should be designed as safety-related software itself, with all the attendant regulatory requirements for safety software."

IEEE Std. 7-4.3.2-2003 [35], Clause 5.3.2 provides specific guidance for software tools that can be found in the appropriate section below. Section 5.3.2 of Appendix 7.1-D reiterates the guidance cited in Section 5.3 and continues to point out that tool-operating history may be used to provide additional confidence in the suitability of tool. It continues by referencing BTP 7-14 [64] as included above. The section concludes by encouraging reviewers to thoroughly evaluate tool usage and make several points about the problems with tool evaluation.

It should be noted that Section 5.3.5 of the appendix does not specifically mention that software tools should be controlled under a configuration management plan but it is required by Section 5.3.2 and IEEE Std. 7-4.3.2-2003 [35], Clause 5.3.2.

4.1.3.17 NUREG/CR-6263 Discussion

NUREG/CR-6263 [67] aggregates guidance from numerous international standards and software engineering documents to evaluate guidance for developing high-integrity software. It begins in Volume 1 with an executive summary, a review of the process used to evaluate candidate guidance, and a review of the research methods used. It should be noted that significant advances have been made since this document was written and many of the deficiencies noted in the report have been addressed and, to some extent, have been mitigated. In particular, the progress of object-oriented programming has virtually eliminated the deficiencies noted in the area of standardization.

Table ES-1, located in Volume 1, provides several research points or guidelines related to software tools under the heading Software Coding: Development Environment. Several of the guidelines listed are the subject of this study and have been addressed in the discussion of other documents. Other guidelines, such as G5-3 – the development environment should consist of a minimum number of different tools, appear somewhat arbitrary and may need

further evaluation. The technical basis for this guideline, found in Volume 2, Section 5.2.2.B3, cites EPRI Report NP-6780 [117], Chapter 10, "Man-Machine Interface Systems." A search of the EPRI database cannot locate this document. However, an apparent updated version of this document, EPRI Technical Report TR-016780 [118], does contain this particular guideline listed in Chapter 10, Section 6.1.3.9, but the rationale for the statement does not address this requirement. A better guideline may be that different tools should be used on redundant systems to reduce the possibility of common-mode failure.

The remaining guidelines appear to be valid and should be incorporated in this study, as long as the guidelines can be supported with other documentation and where the rationale in the source document validates the requirement.

4.1.3.18 NUREG/CR-6421 Discussion

NUREG/CR-6421 [68] discusses a proposed approval process for COTS software to be used in reactor safety systems. Given the requirement of IEEE Std. 7-4.3.2-2010 [1] that software tools should be developed under the quality assurance process of 10 CFR Part 50, Appendix B or should be commercially dedicated, the criteria of commercial dedication in this document can be applied.

In Section 1.3, based on the definition of critical software, software tools used to develop software for systems that are important to safety can be classified as critical software.

Section 2.3 provides a special note concerning compilers, linkers, and operating systems. The discussion focuses on the difficulty of commercial dedication of these tools. By extension, many of the tools that are being considered for this study will have similar difficulties.

Tables 1 and 2 of NUREG/CR-6421 [68] discuss safety categories and COTS software usage categories. The IEC 61226 safety categories are defined in Section 4.1.5.4 of this report.

Table 1 shows by example some familiar reactor systems and where they would be placed in the IEC 61226 [98] scheme. An approximate equivalence to Regulatory Guide 1.97, Revision 3 [126], instrument signal categories is also shown. Regulatory Guide 1.97, Revision 3 [126] prescribes the instrument variables to be monitored. However, Regulatory Guide 1.97, Revision 4 [127] supersedes Regulatory Guide 1.97, Revision 3 [126] and endorses IEEE Std. 479-2002 [132] because the increased use of digital instrumentation systems in advanced nuclear power plant designs requires a more flexible standard which recognizes the advantage of providing performance-based criteria for use in selecting variables. Although IEEE Std. 479-2002 [132] has been superseded by IEEE Std. 479-2010 [133], both versions establish flexible, performance-based criteria for the selection, performance, design, qualification, display, and quality assurance of accident monitoring variables.

Table 2 summarizes the possible classifications of COTS products according to the IEC 61226 [98] categories. The current version of IEC 61226 [98] does not discuss software tools. However, the version of IEC 61226 [98] that was in a preliminary status at the time NUREG/CR-6421 was written classified software that directly produces executable modules used in A, B, or C applications (software tools such as compilers, linkers, automatic configuration managers, or

the like) as IEC 61226 [98] safety class A, B, or C depending on the level of software produced and whether there exists a diverse alternative or another software tool that verifies the output. For example, if the tool is used to produce class A modules, it will be classified as either class A or class B. It further states that CASE systems, or other support systems that indirectly assist in the production of A, B, or C applications, are classified as support usage.

Table 3 formalizes the decision process detailed in Table 2. It states the criteria used to determine the safety category classification of COTS software.

- Criterion 1 states that if the COTS product is used directly in a system important to safety, the COTS safety category is determined by the criteria of IEC 61226[1] [98].
- Criterion 2 states that COTS that directly produces or controls the configuration of an executable software product that is used in a system important to safety, and no method exists to validate the output of the COTS product, the COTS safety category is the same as that of its output, except that category C software may be produced by COTS products of the unclassified category. COTS software that directly produces category A or B software that is validated by other means is category B or C, respectively.
- Criterion 3 states that COTS that supports production of category A, B, or C software, but does not directly produce or control the configuration of such software modules, falls under the safety category “unclassified.”
- Criterion 4 states that if the COTS product has no impact on category A, B, or C software or systems, it falls under the safety category “unclassified”. Category C and unclassified products have relatively low reliability requirements. Therefore, software that produces category C or unclassified software may be of standard commercial quality.

The proposed acceptance process is based on the classification scheme described in Section 2 and on a set of acceptance criteria derived from the standards described in Section 3. It is broken into two phases: a preliminary qualification phase, and a detailed qualification phase. The preliminary qualification phase applies to all COTS products, regardless of the ultimate safety categorization. This phase is concerned with understanding system safety requirements, understanding the COTS product's proposed role in a system important to safety, unambiguously identifying the COTS product, and determining the rigor of subsequent qualification procedures. The detailed qualification phase activities vary in rigor and content depending upon the result of the preliminary phase.

These criteria, the acceptance criteria discussed in Section 4, and the expanded explanations given in Table 5 for category A software, Table 6 for category B software, and Table 7 for category C software, determine the level of rigor necessary to approve the many types of COTS software tools under consideration. It may be possible to generically approve some types of

[1] The original text of NUREG/CR-6421 [68] references IEC 1226 which has been withdrawn.

software tools under the guidance of this document; however, a majority of the COTS software tools will require individual analysis.

4.1.4 Institute of Electrical and Electronics Engineers Documents Discussion

4.1.4.1 IEEE Std. 7-4.3.2-2003 Discussion

Endorsed by RG 1.152, Revision 3 [34], IEEE Std. 7-4.3.2-2003 [35] discusses software tools in Section 5.3.2. However, the introduction contains this statement: "This standard does not address justification for the selection of software tools and acceptance criteria for compilers, operating systems, and libraries." In Section 3.1.42, it defines a software tool as "a computer program used in the development, testing, analysis, or maintenance of a program or its documentation." Examples include comparator, cross-reference generator, decompiler, driver, editor, flowcharter, monitor, test case generator, and timing analyzer.

Section 5.3.2 contains two requirements to determine the suitability of software tools. Meeting either or both requirements is considered confirmation. The first is that a test tool validation program shall be developed to provide confidence that the necessary features of the software tool function as required. The second is that the software tool shall be used in a manner such that defects not detected by the software tool will be detected by V&V activities. It continues to state that operating experience may be used to provide additional confidence in the tool. It is the opinion of the reviewer that while these requirements are valid they are not adequate.

4.1.4.2 IEEE Std. 7-4.3.2-2010 Discussion

IEEE Std. 7-4.3.2-2010 [1] is the latest version of the standard endorsed in RG 1.152, Revision 3 [34] and discussed in NUREG 0800, Appendix 7.1-D [65]. Software tools are discussed in Section 5.3.2. Two requirements are discussed in the section. These requirements greatly expand on the earlier version and improve the validity of the requirements. As in the earlier version, meeting one or both of the requirements is considered confirmation of the suitability of the tools. The first is that the output of the software tool is to be subject to the same level of verification and validation as the safety-related software. The second is that the tool should be developed under the quality assurance process of 10 CFR Part 50, Appendix B or it should be commercially dedicated. A third requirement is that software tools used in the life cycle process of safety-related software should be controlled under configuration management. The wording of these requirements removes the ambiguity that was inherent in the earlier version.

4.1.4.3 IEEE Std. 603-1991 Discussion

IEEE Std. 603-1991 [37] establishes minimum functional design criteria for the power, instrumentation, and control portions of nuclear power generating station safety systems. The basis for discussion in this standard is how the instrumentation and control systems should function during normal operations and design-basis events.

This standard establishes two very important concepts. The first is the division between safety and non-safety systems in nuclear power plant instrumentation and control systems. Even though this document is directed primarily at analog control systems, the distinction between the

systems is still valid. The second important concept put forth in this document is that of the single-failure criterion. Although IEEE Std. 603-1991 does not discuss software tools, ISL believes that the single-failure concept should be extended to software tools because of the possibility that a single software tool may be used during several portions of the software development life cycle. Care must be taken to ensure that this practice does not violate the single-failure criterion.

4.1.4.4 IEEE Std. 603-1998 Discussion

IEEE Std. 603-1998 [70] is a revision to IEEE Std. 603-1991 [37] that contains several changes unrelated to this study. The purpose of this revision is to clarify the application of this standard to computer-based safety systems and to advanced nuclear power generation station designs. This revision provides guidance for the treatment of electromagnetic interference (EMI) and radio-frequency interference (RFI), clarifies definitions (e.g., “Class 1E”), and updates references.

IEEE Std. 603-1998 [70] still establishes minimum functional design criteria for the power, instrumentation, and control portions of nuclear power generating station safety systems but still does not discuss software development. All discussion of digital computers, software, or firmware is referred to IEEE Std. 7-4.3.2-1993 [69]. As such, there are no specific requirements for software tools; however, the two concepts established in the previous version are still valid. The first is the division between safety and non-safety systems in nuclear power plant instrumentation and control systems. The second is the concept of single-failure criterion.

4.1.4.5 IEEE Std. 603-2009 Discussion

IEEE Std. 603-2009 [71] is a revision to IEEE Std. 603-1998 [70] that contains a change that may be of interest to this study. The relevant change is that the definition of common-cause failure has been updated to read “loss of function to multiple structures, systems, or components due to a shared root cause.” Although IEEE Std. 603-2009 does not discuss software tools, ISL believes that the single-failure concept is important to an understanding of software tools as the same tool may be used in development of software for multiple systems thus making a fault in the tool a potential source of a common-cause failure. All references to IEEE Std. 7-4.3.2-1993 [69] have been updated to IEEE Std. 7-4.3.2-2003 [35].

4.1.4.6 IEEE Std. 828-1990 Discussion

As described in RG 1.169 [44], IEEE Std. 828-1990 [45] presents methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Therefore, despite the fact that software tools are not specifically mentioned in the standard, the software configuration management plan documentation required by this standard, with the exceptions listed in RG 1.169 [44], can be used to determine if software tools have been developed under an Appendix B software quality assurance process. This allowance is explicitly explained in Regulatory Position No. 8 of RG 1.168 [38].

4.1.4.7 IEEE Std. 828-1998 Discussion

IEEE Std. 828-1998 [74] is a revision to IEEE Std. 828-1990 [45]. The standard presents methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. There are no significant changes that affect software tools.

4.1.4.8 IEEE Std. 828-2005 Discussion

IEEE Std. 828-2005 [48] is a revision to IEEE Std. 828-1998 [74]. IEEE Std. 828-2005 [48] is endorsed by RG 1.169, Revision 1 [47] and is described as a minor revision performed to ensure consistency between this standard, IEEE Std. 12207.1-1997 [18], and the IEEE Software Engineering Body of Knowledge (SWEBOK) project. Side-by-side comparison shows no significant differences. It should be noted that RG 1.169, Revision 1 [47] states:

> Tools used in the development of safety system software should be handled according to IEEE Std. 7-4.3.2-2003 [35], "IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," issued 2003, as endorsed by RG 1.152, Revision 3 [34]. In particular, an SCM program operated by the using organization that complies with IEEE Std. 828-2005 [48] should take control of tools (i.e., tools should be treated as configuration items). SCM Plans should identify that software tools used in the production of safety system software be considered as a configuration item.

4.1.4.9 IEEE Std. 828-2012 Discussion

IEEE Std. 828-2012 [75] is a revision to IEEE Std. 828-2005 [48]. In keeping with modifications to other IEEE standards, this revision shifts the focus of the standard from software to system and software processes. This shifts the life cycle conformance from IEEE Std. 1074-2006 [4] to IEEE Std. 12207-2008 [2] and IEEE Std. 15288-2008 [5]. Requirements listed for previous versions associated with software tools are not affected by this change.

4.1.4.10 IEEE Std. 829-1983 Discussion

As described in RG 1.170 [49], IEEE Std. 829-1983 [50] describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Therefore, despite the fact that software tools are not specifically mentioned in the standard, the test documentation required by this standard, with the exceptions listed in RG 1.170 [49], can be used to determine if software tools have been developed under an Appendix B software quality assurance process.

4.1.4.11 IEEE Std. 829-1998 Discussion

IEEE Std. 829-1998 [78] is a revision to IEEE Std. 829-1983 [50]. Software tools are not specifically mentioned in this updated standard and the changes implemented in IEEE Std. 829-1998 [78] do not affect software tools.

4.1.4.12 IEEE Std. 829-2008 Discussion

IEEE Std. 829-2008 [52] is endorsed by RG 1.170, Revision 1 [51]. This guide describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Therefore, despite the fact that software tools are not specifically mentioned in the standard, the test documentation required by this standard, with the exceptions listed in RG 1.170, Revision 1 [51], can be used to determine if software tools have been developed under an Appendix B software quality assurance process.

Clause 6.3 of IEEE Std. 829-2008 [52] states that there is no need for repetition of test information if completely managed by an automated tool with references for tracing the information. In RG 1.170, Revision 1 [51], the NRC staff takes exception to this clause as there are particular cases where electronic validation methods with repetition provide test information easily accessible for the basis of any safety conclusion. This test information should be available within the record management system.

4.1.4.13 IEEE Std. 830-1993 Discussion

As described in RG 1.172 [56], IEEE Std. 830-1993 [57] describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Software tools for development of software requirements are discussed briefly in Section 4.3.2.3. The discussion primarily focuses on three categories of tools – object oriented, process oriented, and behavior oriented. Despite the limited discussion of software tools, the software requirements specification documentation required by this standard, with the exceptions listed in RG 1.172 [56], can be used to determine if software tools have been developed under an Appendix B software quality assurance process.

Additionally, this standard can be used to reverse engineer pre-existing or COTS software to recreate missing documentation as recommended by IEEE Std. 7-4.3.2-2010 [1].

4.1.4.14 IEEE Std. 830-1998 Discussion

IEEE Std. 830-1998 [59] is a revision to IEEE Std. 830-1993 [57] and is endorsed by RG 1.172, Revision 1 [58]. As described in RG 1.172, Revision 1 [58], this guide describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. The NRC clarifies in RG 1.172, Revision 1 [58], that when tools are used to present requirements, traceability must be maintained between the representations and the natural language description of the software requirements. Despite the limited discussion of software tools, the software requirements specification documentation required by this standard, with the exceptions listed in RG 1.172, Revision 1 [58], can be used to determine if software tools have been developed under an Appendix B software quality assurance process. There are no other significant changes that affect software tools. This standard has been superseded by IEEE Std. 29148-2011 [60].

4.1.4.15 IEEE Std. 1008-1987 Discussion

As described in the original version of RG 1.171 [53] and in Revision 1 to RG 1.171 [55], IEEE Std. 1008-1987 [54] describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Despite the limited discussion of software tools, the unit testing documentation required by this standard, with the exceptions listed in RG 1.171, Revision 1 [55], can be used to determine if software tools have been developed under an Appendix B software quality assurance process. Additionally, the unit testing processes described can be used to test pre-developed and COTS software tools for commercial dedication. RG 1.171, Revision 1 [55] notes that references in this document to IEEE Std. 829-1983 [50] should be updated to reference IEEE Std. 829-2008 [52] as endorsed by RG 1.170, Revision 1 [51].

4.1.4.16 IEEE Std. 1012-1998 Discussion

As described in RG 1.168, Revision 1 [38], IEEE Std. 1012-1998 [39] describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Despite the limited discussion of software tools, the verification and validation documentation required by this standard, with the exceptions listed in RG 1.168, Revision 1 [38], can be used to determine if software tools have been developed under an Appendix B software quality assurance process. Additionally, the processes described can be used to test pre-developed and COTS software tools for commercial dedication.

One important statement is included in Section 7.4.6, “Tools, Techniques, and Methods.” This statement should be incorporated into our basic requirements.

> “Tools that insert code into the software shall be verified and validated to the same rigor as the highest software integrity level of the software. Tools that do not insert code shall be verified and validated to assure that they meet their operational requirements. If partitioning of tool functions can be demonstrated, only those functions that are used in the V&V processes shall be verified to demonstrate that they perform correctly for their intended use.”

A recommendation for independence is listed in Annex C under technical independence. It states that technical independence requires the independent V&V effort to use its own set of test and analysis tools separate from the developers’ tools. As this annex is only informative, this is not a requirement but may be a concept to keep in mind during further analysis.

4.1.4.17 IEEE Std. 1012-2004 Discussion

IEEE Std. 1012-2004 [42] amplifies the statement quoted above by specifying four levels of software integrity. Software integrity is defined as “a range of values that represent software complexity, criticality, risk, safety level, security level, desired performance, reliability, or other project-unique characteristics that define the importance of the software to the user and acquirer.” It should be noted that RG 1.168, Revision 2 [41], takes exception to Table B.1, “Assignment of Software Integrity Levels,” and Table B.3, “Graphic Illustration of the Assignment

of the Software Integrity Levels." The NRC notes that the description "critical consequences" is not acceptable as a description for Level 4. The standard goes on to state "Tools that insert or translate code (e.g., optimizing compilers, auto-code generators) shall be assigned the same integrity level as the integrity level assigned to the software element that the tool affects." This adds code translation and a few examples to the older requirement. Additional guidance in RG 1.168, Revision 2 [41], notes that Annex D, "V&V of Reuse Software" is not acceptable to the NRC.

4.1.4.18 IEEE Std. 1012-2012 Discussion

IEEE Std. 1012-2012 [79] has not been reviewed by the NRC staff. As such, any guidance that conflicts with the IEEE Std. 1012-2004 [42] shall defer to the endorsed revision. This revision of the standard widens the scope of V&V activities to include system life cycle activities as described in IEEE Std. 12207-2008 [2]. The methodology of the standard scope revision places common V&V activities in Clause 7, system V&V activities in Clause 8, software V&V activities in Clause 9, and hardware V&V activities in Clause 10. The activities described in these clauses may be combined as required to fit the application under development or review.

Software integrity as described in IEEE Std. 1012-2004 [42] has also been expanded to cover hardware and system integrity levels. These integrity levels are used to determine the V&V tasks, activities, rigor, and the level of intensity of the V&V to be performed. The standard explains the assignment of integrity levels for reuse components (including software) as follows:

> The integrity level assigned to reused, COTS, and GOTS components shall be in accordance with the integrity-level scheme adopted for the system element into which COTS or GOTS may be integrated for the project. The reused COTS or GOTS component shall be evaluated for use in the context of its application. The design, development, procedural, and technology features implemented in the system can raise or lower the assigned integrity levels.

This revision of the standard expands on the guidance for tools that insert or translate code with the statement "If the tool cannot be verified and validated, then the output of the tool shall be subject to the same level of V&V as the software element."

Tables B.1 and B.3 have not been modified in this revision. Therefore, the exceptions noted in RG 1.168, Revision 2 [41] should be considered in effect.

It should be noted that Annex D has very few changes from IEEE Std. 1012-2004 [42]. The guidance of RG 1.168, Revision 2 [41], that states that Annex D is not acceptable to the NRC should still apply.

4.1.4.19 IEEE Std. 1028-1997 Discussion

As described in RG 1.168, Revision 1 [38], IEEE Std. 1028-1997 [40] describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Despite the limited discussion of software tools, the software review documentation required by this standard, with

the exceptions listed in RG 1.168, Revision 1 [38], can be used to determine if software tools have been developed under an Appendix B software quality assurance process. Additionally, the processes described can be used to review pre-developed and COTS software tools for commercial dedication.

4.1.4.20 IEEE Std. 1028-2008 Discussion

IEEE Std. 1028-2008 [43] is endorsed with exceptions by RG 1.168, Revision 2 [41]. According to RG 1.168, Revision 2 [41], IEEE Std. 1028-2008 [43] provides guidance acceptable to the NRC staff for carrying out software reviews, inspections, walkthroughs, and audits subject to certain provisions. The primary focus of the changes in this revision was to clarify the difference between inspections and walkthroughs. In this revision, process improvement is mandatory for inspections and process improvement is no longer required for walkthroughs. No significant changes affecting software tools were noted in the survey of this document. Annex A of this revision is not endorsed by the NRC.

4.1.4.21 IEEE Std. 1042-1987 Discussion

As described in RG 1.169 [44], IEEE Std. 1042-1987 [46] describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Despite the limited discussion of software tools, the configuration management documentation required by this standard, with the exceptions listed in RG 1.169 [44], can be used to determine if software tools have been developed under an Appendix B software quality assurance process. Additionally, the processes described can be used to review pre-developed and COTS software tools for commercial dedication. This standard has been withdrawn due to obsolescence. Software configuration management is discussed in IEEE Std. 828-2005 [48] and IEEE Std. 828-2012 [75].

4.1.4.22 IEEE Std. 1074-1995 Discussion

As described in RG 1.173 [61], IEEE Std. 1074-1995 [62] describes methods acceptable to the NRC staff for complying with parts of the NRC's regulations for achieving high functional reliability and design quality in software used in safety systems. Despite the limited discussion of software tools, the software life cycle process documentation required by this standard, with the exceptions listed in RG 1.173 [61], can be used to determine if software tools have been developed under an Appendix B software quality assurance process. Additionally, the processes described can be used to review pre-developed and COTS software tools for commercial dedication.

It is important to recognize that this standard does not advocate or specify a single software life cycle model. The processes described are valid for any number of different models; therefore, no models are described in the standard. In this respect, the life cycle model is treated the same as in IEEE Std. 12207-2008 [2] and IEEE Std. 15288-2008 [5]. As described in IEEE Std. 15288-2008 [5], “The life cycle model comprises one or more stage models, as needed. It is

assembled as a sequence of stages that may overlap and/or iterate, as appropriate for the system-of-interest's scope, magnitude, complexity, changing needs and opportunities."

The life cycle processes described in this standard are similar to, but not the same as, those described in IEEE Std. 12207-2008 [2] or IEEE Std. 15288-2008[5]. The differences between the standards can be attributed to different groups preparing the standards and the fact that IEEE Std. 1074-1995 [62] is primarily focused on development of a software life cycle process while the other two standards are more focused on utilizing the life cycle processes.

4.1.4.23 IEEE Std. 1074-1997 Discussion

IEEE Std. 1074-1997 [80] is not currently endorsed by the NRC. IEEE Std. 1074-1997 [80] is very similar to IEEE Std. 1074-1995 [62]. There are no significant changes to this version that affect software tools. In IEEE Std. 1074-1997 [80], software life cycle activities were rearranged into more logical groupings (called activity groups) such as placing all planning activities into the new Project Planning Activity Group. IEEE Std. 1074-1997 [80] has replaced the term "process" with the term "activity group" to identify collections of activities and to avoid misinterpreting the "process" collections as actual processes and trying to execute them. IEEE Std. 1074-1997 [80] added a new activity group, "manage risks," based on the importance of risk management. Finally, IEEE Std. 1074-1997 [80] added a Software Importation Activity Group because of the recognition that software can be acquired from other sources for use in the system being developed.

4.1.4.24 IEEE Std. 1074-2006 Discussion

IEEE Std. 1074-2006 [4] is endorsed by RG 1.173, Revision 1 [6]. IEEE Std. 1074-2006 [4] is very similar to IEEE Std. 1074-1995 [62] and IEEE Std. 1074-1997 [80]. There are no significant changes to this version that affect software tools. The focus of IEEE Std. 1074-2006 [4] is more clearly centered on a single process for a given project. In IEEE Std. 1074-2006 [4], the term "compliance" was changed to "conformance" to reflect international standards usage. IEEE Std. 1074-2006 [4] added release management activities in recognition of the importance of release management. Due to the emerging importance and attention to software security, IEEE Std. 1074-2006 [4] added two security-related activities: determine security objectives and confirm security accreditation. RG 1.173, Revision 1 [6] acknowledges that planning for security and confirming the security accreditations is necessary and part of a secure analysis. However, IEEE Std. 1074-2006 [4] is not endorsed in RG 1.173, Revision 1 [6], as being appropriate for compliance with 10 CFR 73.54.

4.1.4.25 IEEE Std. 1209-1992 Discussion

IEEE Std. 1209-1992 [82] addresses the evaluation and selection of CASE tools to support the software life cycle process described in IEEE Std. 1074-1991 [119]. Written from the perspective of a software tool user, the standard presents evaluation and selection criteria that address program characteristics visible to the user. The functional aspects of the tools and the process by which the tools were developed are not addressed. If followed, the evaluation and selection process described results in a set of documentation that can be used to determine the

reliability of the tools during the NUREG 0800 review process. This standard has been withdrawn due to obsolescence. Adoption and selection of CASE tools are discussed in IEEE Std. 14102-2010 [83], IEEE Std. 14471-2010 [84], and IEEE Std. 1175.4-2008 [85].

4.1.4.26 IEEE Std. 1462-1998 Discussion

IEEE Std. 1462-1998 [86] addresses the evaluation and selection of CASE tools to support the software product quality model described in ISO/IEC 9126:1991 [120]. In this standard, high-level quality factors are described and, in some cases, are broken down into lower-level characteristics. The primary areas for evaluation include how the CASE tool meets the needs of the life cycle processes used; how the CASE tool relates to the overall tool environment; how the CASE tool itself performs; and other criteria. The evaluation process recommends that characteristics from each area should be selected and should then be prioritized. The standard presents an extensive list of characteristics that can easily be converted into criteria for review of software tools during a NUREG 0800 review process.

4.1.4.27 IEEE Std. 12207-2008 Discussion

IEEE Std. 12207-2008 [2] integrates ISO/IEC Std. 12207:1995 [90] and its two amendments and was coordinated with the parallel revision of ISO/IEC Std. 15288:2002 [92]. The purpose of the standard is to provide a software-specific life cycle process that is to be used in conjunction with the system life cycle process described in IEEE Std. 15288-2008 [5]. Clause 6 of the standard describes the system life cycle processes of IEEE Std. 15288-2008 [5] with emphasis given to software while Clause 7 describes the life cycle processes normally found only in software development. Annex D provides the alignment of the ISO/IEC Std. 12207:1995 [90] and ISO/IEC Std. 15288:2002 [92] processes.

There are very few direct references to development of software tools but the processes described can be used in conjunction with the tailoring procedures of Annex A for the software tools development process.

Annex G describes the relationship of IEEE Std. 12207-2008 [2] to other IEEE Standards. IEEE Std. 1074-2006 [4], endorsed by RG 1.173, Revision 1 [6], is cited as meeting the requirements of Clause 6.2.1, Lifecycle Model Management Process.

4.1.4.28 IEEE Std. 14102-2010 Discussion

IEEE Std. 14102-2010 [83] defines both a set of processes and a structured set of CASE tool characteristics for use in the technical evaluation and the ultimate selection of a CASE tool. Though it is targeted at users and developers, the documentation produced during the evaluation and selection process can provide significant insight into the quality of the software tools used.

The format of the standard is similar to that used in IEEE Std. 1462-1998 [86] with updates for newer technologies. The processes described include preparation, structuring, evaluation, and selection. The characteristics described coincide with those in IEEE Std. 1462-1998 [86].

4.1.4.29 IEEE Std. 15288-2004 Discussion

IEEE Std. 15288-2004 [91] establishes a common framework for describing the life cycle of systems created by humans. It defines a set of processes and associated terminology. These processes can be applied at any level in the hierarchy of a system's structure. Selected sets of these processes can be applied throughout the life cycle for managing and performing the stages of a system's life cycle.

Software tools or tools in general, are only discussed in this standard in regards to how they are used to support the processes described. However, the processes described can be used in a software tools development environment to provide evidence that the tool was developed in a highly functional methodology.

4.1.4.30 IEEE Std. 15288-2008 Discussion

IEEE Std. 15288-2008 [5] is a revision to IEEE Std. 15288-2004 [91]. This standard establishes a common framework for describing the life cycle of systems created by humans. There are no significant changes to this version that affect software tools.

4.1.4.31 IEEE Std. 29148-2011 Discussion

IEEE Std. 29148-2011 [60] provides guidance for the execution of the IEEE Std. 15288-2008 [5] and IEEE Std. 12207-2008 [2] processes that deal with requirements engineering. This international standard also provides normative definition of the content and recommendations for the format of the information items, or documentation, that result from the implementation of these processes.

The use of a requirements management tool is encouraged because such a tool facilitates a cumbersome and complex project of maintaining requirements traceability and configuration control. When defining a verification plan based on system requirements, IEEE Std. 29148-2011 [60] states that it is a good practice to identify the generic name of analytical/computer tools and to ensure that the analysis methods and tools, including simulations, are acceptable for the provision of objective, proof-of-requirements compliance.

4.1.5 International Electrotechnical Commission Documents Discussion

4.1.5.1 IEC 60880 Ed. 2.0 Discussion

IEC 60880 Ed. 2.0 [95] discusses software tools in Chapter 14 with references to amplifying information found in Chapter 15, "Qualification of Pre-Developed Software." It begins with a brief overview of the use of software tools and their benefits. Selection of tools is discussed focusing on tools that can directly or indirectly affect the quality of the software being produced. Several requirements are listed along with permissible modifications allowing reduction of reliability requirements when adding diversity and defense in depth. Requirements for tools are addressed by topics such as the software engineering environment, tool qualification, configuration management, translators and compilers, application data tools, and automation of testing. These requirements consist of specific measures on how to achieve the more general

requirements. One topic to note is that of diversity. IEC 60880 Ed. 2.0 [95] states that diversity can be achieved by using separate, dissimilar compilers to create software outputs. When one considers that approximately 60% of software errors are traceable to specifications, this position is difficult to support; however, the remaining 40% may benefit. Despite this, IEC 60880 Ed. 2.0 [95], Chapter 14 contains a significant amount of information that can be used to develop guidelines for software tool approval.

4.1.5.2 IEC 62138 Ed. 1.0B Discussion

IEC 62138 Ed. 1.0B [97] provides requirements for the software of computer-based I&C systems performing functions of safety category B or C as defined by IEC 61226 [98]. It complements IEC 60880 Ed. 1.0 [99] and IEC 60880-2 Ed. 1.0 [100], which provide requirements for the software of computer-based I&C systems performing functions of safety category A.

It is also consistent with, and complementary to, IEC 61513 Ed. 1.0 [101]. Activities that are mainly system level activities (for example, integration, validation, and installation) are not addressed exhaustively by this standard. Requirements that are not specific to software are deferred to IEC 61513 Ed. 1.0 [101].

The primary focus of this document is to describe the software development process in terms of the software safety life cycle. This life cycle is similar to the software development life cycle described in IEEE 12207-2008 [2] with an emphasis on the stages that can affect software safety.

Selection and use of software tools for safety level C systems are discussed in Section 5.1.4 while tools for safety level B systems are discussed in Section 6.1.4. These sections provide several guidelines for the selection of software tools. In addition to the guidelines, these sections advise that the compatibility of a tool with other tools in use should be considered along with the tool quality. They also recommend that a well-known tool with extensive operating experience is preferable to an untried tool with little or no operating experience as long as the well-known tool meets other selection criteria. When dealing with COTS software, operating experience is often the deciding factor in determining a tool's quality. Additional evidence may be based on tool certification, certification of the suppliers, and tests.

4.1.5.3 IEC 62340 Ed. 1.0B Discussion

IEC 62340 Ed 1.0B [102] is a second tier IEC standard that addresses aspects to overcome common-cause failures (CCFs). This standard provides an overview of the relevant requirements for new and replacement I&C systems that perform functions important to safety in nuclear power plants (NPPs) if their failure would be unacceptable with respect to the plant safety design. IEC 61513 Ed. 1.0 [101] is the top level document that provides general requirements for I&C systems and equipment that are used to perform functions important to safety in NPPs. IEC 62340 Ed. 1.0B [102] supplements IEC 61513 Ed. 1.0 [101] and related standards with requirements to reduce and overcome the possibility of CCF of I&C functions of category A.

This standard presents requirements related to the avoidance of CCF of I&C systems that perform category A functions and the implementation of independent I&C systems to overcome CCF. Means to achieve protection against CCF are discussed in this standard in relation to susceptibility to internal plant hazards and external hazards; propagation of physical effects in the hardware; and avoidance of specific faults and vulnerabilities within the I&C systems.

This standard explains the strategy to cope with CCF. I&C systems that perform category A functions are designed with appropriate redundancy combined with voting mechanisms to meet single failure criterion. I&C systems with this design can fail if two or more redundant channels fail concurrently. The CCF can occur if a latent fault is systematically incorporated in some or all redundant channels and if by a specific event this fault is triggered.

Latent faults may originate from any phase of the life cycle and can be caused by human errors or from manufacturing processes. Faults can be caused by errors in the requirements specifications; inadequate specification of the hardware design limits against environmental conditions; or technical design faults from internally induced mechanisms. A latent fault requires that the fault is not identified by self-supervision or periodic testing and that the concerned components do not fail spontaneously but fail when being activated by a common trigger in some or all redundancies.

Faults in the requirements specification can be overcome through functional diversity. Functional diversity serves to ensure that the main plant safety targets are met, in spite of the possible existence of latent faults related to errors from the requirements specification. The analysis of the design-basis accidents (DBAs) and of the relevant design-basis events (DBEs) which can be caused by failures of the I&C or related subsystems provides the requirements specification from which any need for the application of functional diversity will arise.

Design measures to prevent coincidental failure of I&C systems include designing I&C systems with independence, functional diversity, means to avoid failure propagation through communication paths, design features to prevent system failure due to maintenance activities, integrity, protection from external dates or messages, physical separation, and environmental robustness.

Tolerance against postulated latent software faults requires that digital I&C systems performing Category A functions be designed according to IEC 61513 Ed. 1.0 [101] to operate internally without dependence on the demand profile. The application software shall be designed to be tolerant of invalid input signals. Invalid or faulty input shall be identified on-line. Invalid signals used by one function in a system which performs multiple functions shall not affect other functions with undisturbed input signals. The software shall take safe action even in response to multiple coincident failures of input signals, thereby avoiding DBEs.

Avoiding system failure due to maintenance during operation requires that simultaneous maintenance activities be restricted to a single redundant system. When performing maintenance during power operation, the effects of the maintenance should be analyzed to prevent other I&C systems not subject to maintenance from failing. Hardware replacement

components must have adequate qualification and verification to ensure the reliability of the I&C safety systems is not reduced.

This standard does not discuss software tools but is relevant because single-failure criterion is one of the most important criterion imposed on reactor designs and the concept of single-failure and common-cause failures can be extended to the development and use of software tools.

4.1.5.4 IEC 61226 Ed. 3.0B Discussion

IEC 61226 Ed 3.0B [98] establishes a method of classification of the information and command functions for nuclear power plants, and the I&C systems and equipment that provide those functions, into categories that designate the importance to safety of the function. The resulting classification then determines relevant design criteria.

The design criteria are the measures of quality by which the adequacy of each function in relation to its importance to plant safety is ensured. In this standard, the criteria are those of functionality, reliability, performance, environmental durability (including seismic) and QA. These design criteria shall be defined independently from the technology of the equipment.

This standard is relevant to software tools because the classification of COTS software, as described in NUREG/CR-6421 [68], uses the classification categories of IEC 61226 Ed 3.0B [98].

This standard is applicable to all the information and command functions and the I&C systems and equipment that provide those functions. The functions, systems and equipment under consideration provide automated protection, closed or open loop control and information to the operating staff. They keep the NPP conditions inside the safe operating envelope and provide automatic actions, or enable manual actions, that prevent or mitigate accidents, or that prevent or minimize radioactive releases to the site or wider environment. The I&C functions that fulfill these roles safeguard the health and safety of the NPP operators and the public.

This standard extends the classification strategy presented in IAEA Safety Guide NS-G-1.3 [135], and establishes the criteria and methods to be used to assign the I&C functions of a NPP to one of three categories A, B and C, depending on their importance to safety, or to an unclassified category for functions with no direct safety role. I&C functions falling within the boundary of the safety systems will generally be assigned to category A or B. I&C functions defined as safety related will generally be assigned to categories B or C.

Category A

Category A denotes the functions that play a principal role in the achievement or maintenance of NPP safety to prevent DBE from leading to unacceptable consequences. This role is essential at the beginning of the transient when no alternative actions can be taken, even if hidden faults can be detected. These functions play a principal role in the achievement or maintenance of the non-hazardous stable state. If specified manual actions are provided to reach the non-hazardous stable state, factors such as the availability of redundant, validated, information sources, sufficient duration of the grace time for operator evaluation of alternative

sources of information, and whether the manual actions are the only possibility for mitigation of this sequence of events to preserve NPP safety, have to be considered.

Category A also denotes functions whose failure could directly lead to accident conditions which may cause unacceptable consequences if not mitigated by other category A functions. Category A functions have high reliability requirements. Consequently, it may be necessary to limit their functionality and complexity.

Category B

Category B denotes functions that play a complementary role to the category A functions in the achievement or maintenance of NPP safety, especially the functions required to operate after the non-hazardous stable state has been achieved, to prevent a DBE from leading to unacceptable consequences, or mitigate the consequences of a DBE. The operation of a category B function may avoid the need to initiate a category A function. Category B functions may improve or complement the execution of a category A function in mitigating the consequences of a DBE, so that plant or equipment damage or activity release may be avoided or minimized.

Category B also denotes functions whose failure could initiate a DBE or worsen the severity of a DBE. Because of the presence of a category A function to provide the ultimate prevention of or mitigation of the consequences of a DBE, the safety requirements for the category B function need not be as high as those for the category A function. This allows, if necessary, the category B functions to be of higher functionality than category A functions in their method of detecting a need to act or in their subsequent actions.

Category C

Category C denotes functions that play an auxiliary or indirect role in the achievement or maintenance of NPP safety. Category C includes functions that have some safety significance, but are not category A or B. They can be part of the total response to a DBA but not be directly involved in mitigating the physical consequences of the accident, or be functions necessary for beyond design-basis accidents.

4.1.6 International Atomic Energy Agency Documents Discussion

4.1.6.1 IAEA NS-G-1.1 Discussion

The main focus of safety guide IAEA NS-G-1.1 [138] is on the preparation of documentation that is used for an adequate demonstration of the safety and reliability of computer-based systems important to safety. This safety guide applies to all types of software: pre-existing software or firmware (such as an operating system), software to be specifically developed for the project, or software to be developed from an existing pre-developed equipment family of hardware or software modules. The safety guide addresses proper planning, documentation, selection, and qualification of software tools in several life cycle phases.

Section 3 discusses the need for diversity in the design of computer-based systems. The reliability of computer-based systems can be enhanced by using diversity to reduce the potential for software common-cause failures. Diversity of methods, languages, tools, and personnel should be taken into consideration. With respect to the use of automated tools, all tools used should be mutually compatible. The tools should be qualified to a level commensurate with their function in the development of the software and in the safety demonstration. The techniques used to gain confidence in the tools should be specified and documented. Well established methods, languages, and tools for software development should be used. Methods, languages, and tools that are still at the research stage should not be used.

Section 4 provides recommendations on the planning phase of the system development project, and describes the structure and content of the associated documentation, including the development plan, the quality assurance program description, the verification and validation plan, and the configuration management plan.

Development Plan

The tools to be used should be identified in the development plan. The tools should be selected to facilitate the proper application of the methods, standards, and procedures selected. Planning in advance for the entire project will help in the selection of an integrated set of tools. The tools should be properly qualified for their function in the development, management, or verification of the system. The correctness of their output should be ensured by a tool certification process, by means of cross-verification or by reverse engineering.

Tools should be used since they relieve staff from manual error prone tasks such as programming and verification. They help to achieve a reproducible level of quality, in part guaranteed and demonstrated by the evidence provided by their past usage.

Documents to be produced during each phase should be identified and their contents specified. The schedule for the documents should be established and times should be identified when project review is to be conducted. These are products of a managerial task that include the assessment of the adequacy of the facilities and tools available. A plan should be prepared to ensure that those personnel involved in the development activities are competent in the application of the relevant standards, procedures, and methods, in the use of design, programming and analysis tools and methods, and in configuration management and change control practices. Records of their competence should be maintained. If new members are added to the team, they should be closely supervised until they have demonstrated their competence to their line managers.

Quality Assurance

The quality assurance program description should be prepared and implemented by the licensee and should be available for regulatory review (and possibly approval) before the project begins. A software quality assurance plan should be produced at the outset of the project. This plan should cover external suppliers and should include at least the following:

(1) Identification of the hardware and software items to which the quality assurance program description applies and of the governing standards, procedures, and tools to be used on the project.

(2) The procedures for qualifying tools

Verification and Validation Plan

The team or teams performing the verification and validation should be identified in the plan. The verification and validation tasks should be allocated between the teams. The teams performing verification and validation should be independent of the development team. The amount and type of independent verification and validation should be justified with respect to the safety class of the system; for example, financial independence may not be required for safety-related systems. Independence includes:

(1) Technical independence: The work should be done by different people, using different techniques and tools.

(2) Management independence: The work should be led and motivated by different people. The verification and validation team and the development team should have different management lines. Official communication between independent teams should be recorded.

(3) Financial independence: There should be a separate budget with restrictions on transfer of financial resources between development and verification and validation.

Configuration Management Plan

All items of software development, such as compilers, development tools, configuration files and operating systems, should be under configuration management control.

Sections 5 to 15 are dedicated to individual phases of the development life cycle including the computer system design phase, the software requirements specification phase, the software design phase, the software implementation phase, the verification and analysis phase, the computer system integration phase, the validation phase, the installation and commissioning phase, the operations phase, and the post-delivery modification phase. Only those phases that contain information relevant to software tools will be discussed.

Section 5 covers the computer system requirements specification phase. The computer system requirements should be independent of the implementation. They should deal with the computer system as a black box. A formalism for specifications that is understandable to all parties concerned, with a well defined syntax and semantics, should be used to capture the results of that work. This formalism should be understandable to the licenser, suppliers, and designers of the computer system and the software specifications. The formalism can be supported by tools that help in the validation of the completeness and consistency of the requirements. Tools used to analyze the consistency and completeness of the computer

system requirements should be validated to a level of confidence proven equivalent to the level required from the design process.

Section 7 covers the software requirements specification phase. Software requirements are the subset of the computer system requirements that will ultimately be implemented as computer programs. Specifiers should be aware of the capabilities of the computers, tools and similar existing systems so that it is feasible to fulfill the software requirements in the software design and the software implementation. If the computer system requirements are sufficiently detailed, and their documentation is sufficiently formal, and if parts of the computer system design and of the code are generated by tools, then a separate software requirement document may be unnecessary for those parts. However, those parts of such computer system requirements from which code is generated or reused should be regarded as a statement of software requirements against which subsequent code should be verified. In addition, any separately compiled modules that are included by the code generator should be supported by separate documents for the software requirements.

Section 8 covers the software design phase. At the design stage of the software development, choices of a technological nature may need to be made. Examples of such implementation constraints are the needs to ensure diversity and the attributes required of the programming languages, compilers, subroutine libraries and other supporting tools. This information should be provided in the software documentation. Implementation constraints should be justified or be traceable to higher-level requirements or constraints.

Section 9 covers the software implementation phase. Code can be produced from the system specifications in various ways that are essentially combinations of two distinct approaches: (1) the classical development process through the stages of specifications and design or (2) the use of code generating tools which receive as input an application oriented description of the system with a high-level language. The choice between these two approaches depends on the tools and resources available to the parties engaged in the project and should take into account, in particular, trade-offs between design and the demonstration of the dependability of tools. The production of code that can be demonstrated to be a correct implementation of the software specifications is a major issue. The code should be verifiable against these specifications. If verification is made by human inspection, the code should be readable, adequately annotated, and understandable. Validated tools should be used since they relieve programmers from tasks of the code production process that are prone to manual error, such as programming and verification. A suitable set of tools should be selected for the implementation. These may include translators and code generators, debuggers, linkers, and testing and other verification tools. Thoroughly tested translators that are available and maintainable throughout the system lifetime should be used. If no thoroughly tested translator is used, additional analysis and verification, manual or by the use of other tools, should demonstrate that the translation is correct. Any tools used in the software implementation phase should be compatible with other tools used in other phases of the development. The choice of all the tools used should be justified and documented. The process by which the tools have been validated should also be documented.

Section 9 also covers software diversity during implementation. For example, independent programming teams, diverse solution spaces, and different working environments, tools and run time support systems (operating systems, compilers) can be used. All these aspects and techniques should be considered before a choice is made, if software diversity is to be used. The choice and its justification should be properly documented, with account taken in particular of the extra complexity introduced by these techniques.

Section 10 covers verification and analysis phases. Verification is the process of ensuring that the products of each development phase (including any pre-existing software) satisfy the requirements of the previous phase. Analysis is the process of checking that the products of each development phase are internally consistent and properly apply the chosen techniques. The objective of the verification team should be to provide a full and searching examination that is commensurate with the required reliability. Consideration should be given to the means by which the testing of exception conditions (such as divide by zero, out of range) is to be performed. This might require the use of specialized test tools (such as fault injection tools, in-circuit emulators). Personnel who are planning and conducting the tests should be given appropriate training in the use of test tools, procedures, and techniques. These personnel should be independent from the development team.

Section 10 states that the tools employed in producing software belong to the two following broad categories:

(1) Software development tools, whose output (possibly transformed) becomes part of the program implementation and which can therefore introduce errors. Code generators, compilers, and linkers are examples.

(2) Software verification tools, which cannot introduce errors (but may fail to detect them). Tools for static analysis and test coverage monitors are examples.

For a tool of either category to be employed, there should be a precise definition of the tool's functionality. For a software development tool, the domain of applicability should be precisely known, and for a software verification tool, the analyses or checks it performs should be well defined. In all cases, a tool should have sufficient safety dependability to ensure that it does not jeopardize the safety of the end product. Therefore, a software development tool whose output is used without further review should be of the highest dependability level. This may be relaxed if its output is subjected to verification or if reverse engineering is used. For a software verification tool, the requirements may also be relaxed somewhat, because its output will be closely scrutinized.

Since verification compares the results of one phase with the results of a previous phase, preparation for verification can begin as soon as that previous phase is complete. Such preparation should be documented as the first part of the verification results. This documentation should consist of tool assessment criteria. During a verification process, verifiers should record sufficient information about the process that it could be repeated with confidence of achieving the same results. If any verification tools are used, they should be

completely identified (including version and configuration) and any input or set-up parameters used should be recorded.

Section 15 covers the post-delivery modification phase. During the operational phase, it is necessary to ensure that the functional safety of the computer-based system is maintained both during and after modifications. Adequate change control procedures, which deal specifically with plant safety issues, should be in place. Experts, independent of the designers and developers of the modification, should assess the adequacy of the proposed modification and of its implementation. The changes that should be covered by the change procedures include software changes, hardware changes and tool changes.

4.1.7 Electric Power Research Institute Discussion

4.1.7.1 EPRI NP-5652 Discussion

The need to ensure availability of quality replacement parts for today's operating nuclear power plants in a timely manner is a concern of the industry. Since numerous replacement parts are now being purchased as commercial-grade items (CGIs) because they are no longer available as products designed and manufactured under a 10 CFR Part 50, Appendix B-compliant quality assurance program, increased attention has been focused on their procurement and use. EPRI NP-5652 [104] is intended to alleviate these and other difficulties, such as lack of consistent controls in procurement practices, regulatory inspection findings against utility control of commercial-grade items, and lack of qualified replacement parts due to discontinuation of 10 CFR Part 50, Appendix B quality assurance programs. It facilitates industry uniformity in procuring and dedicating CGIs for use as basic components, i.e., in nuclear safety-related applications.

The key points of NP-5652 [104] from the perspective of this study are the basic premise of commercial-grade dedication and the four acceptance methods for dedication. The basic premise consists of six parts. The first part is verification that the design accurately reflects the requirements for the CGI and that the product delivered is the item that was specified. The second part is a technical evaluation to ensure that the CGI meets the design requirements. The third part is a part number verification. With respect to software tools, this would be the version number or build number of the tool. The remaining parts include consistency with regulatory requirements, uniformity of the CGI supplied, and the documentation required for acceptance. This document also presents four acceptance methods that are summarized below. Note that NRC Generic Letter (GL) 91-05 [121] promulgated the NRC staff position on these NP-5652 [104] provisions and it took exception to part of the EPRI definition of critical characteristics, i.e., that critical characteristics, when verified, provide reasonable assurance that the item received is the item specified. The NRC position was part number verification alone was not sufficient and that critical characteristics, all of which must somehow be verified, provide reasonable assurance that the item will perform its intended safety function. This position was codified in the 1995 revision to 10 CFR Part 21.

The four acceptance methods of NP-5652 [104] were conditionally endorsed by the NRC in GL 89-02 [122], with restrictions on the use of Methods 2 and 4 alone. Further clarification of the

NRC staff position on this guidance was promulgated by the NRC in GL 91-05 [121] as stated above. In 1995, the NRC's new definitions of *basic component, commercial-grade item*, *dedication*, and *critical characteristic* were codified in a revised 10 CFR Part 21 along with other new dedication-related requirements in Part 21, i.e., that the dedicating entity is responsible for reporting under Part 21, that the dedicating entity must maintain auditable records of dedications, and that the dedication must be performed under the applicable criteria of 10 CFR, Part 50, Appendix B. Note that the overall intent of a dedication process is to meet the requirements, primarily of Appendix B Criterion III, "Design Control" (in particular, the review for suitability of application and design verification), and of Criterion VII, "Control of Purchased Material, Equipment, and Services" (in particular, qualification of suppliers and examination of products upon delivery).

Method 1 consists of special tests and inspections. These tests can (and should when warranted) include pre- and post- installation tests. Pre-installation tests would include receipt testing. Method 1 may be performed by the manufacturer, the licensee, or a third-party dedicating entity, as a vendor or licensee subcontractor.

Method 2 consists of commercial-grade survey of the supplier. Note that NRC GL 89-02 [122] states that NP-5652 [104] Method 2 should not be employed as the basis for accepting items (read: sole basis) from suppliers with undocumented commercial quality control programs or with programs that do not effectively implement their own necessary controls. Likewise, Method 2 should not be employed as the basis for accepting items from distributors unless the survey includes the part manufacturer(s) and the survey confirms adequate controls by both the distributor and the part manufacturer(s). It also states that the commercial-grade survey should be item- and critical characteristic-specific, that the survey confirms that the supplier verifies all critical characteristics, and that the supplier has implemented an effective commercial quality assurance (QA) and quality control program. Methods 1 and 2 would be more difficult for software tools but may be applicable.

Method 3 is source verification. In the traditional source verification, the purchaser or its agent personally verifies at the factory the critical characteristics of the actual item(s) being purchased or directly observes the supplier's verification of the critical characteristics. For software tools, this could include specification reports and actual software source code. NP-5652 [104] recommends this method for single-use or infrequent procurement and would it be applicable to most software tools. This method may require the purchaser's specification of hold points to enable the source verifier's direct verification or observation of the supplier's verification of critical characteristics, but it allows for independent purchaser verification of critical characteristics that may not be readily verifiable in the finished product without reliance on the supplier's routine process.

Method 4 consists of acceptable supplier/item performance records. This would be the most reliable source of information for software tools from commercial vendors. However, NRC GL 89-02 states that Method 4 should not be employed alone unless (a) the established historical record is based on industry-wide performance data that is directly applicable to the item's critical characteristics and the intended safety-related application; and (b) the manufacturer's measures

for the control of design, process and material changes have been adequately implemented as verified by audit (multi-licensee team audits such as those of the Nuclear Procurement Issues Council (NUPIC) are acceptable). Note that the NRC staff has accepted commercial-grade surveys to verify those manufacturer attributes, including surveys performed by NUPIC, whose audit and surveys the NRC periodically observes. Note also that control of design and material changes for hardware corresponds roughly to software configuration management.

These methods stem from a single generic method consisting of verification of the safety function of the item, determination that the item meets the criteria for CGI, identifying critical characteristics, documenting results, and selection of the acceptance method or combination of methods. More specifically, the EPRI commercial-grade dedication guidance defines two phases for the dedication process, technical evaluation and acceptance. In the technical evaluation phase, the dedication plan is developed to include determination of the item's safety functions, other application-specific requirements including ambient and process design-basis service conditions, the known vulnerabilities and/or deficiencies of the item based on industry operating experience, and applicable regulatory requirements and guidance. From this information, the critical characteristics are derived, plant application-relevant verification methods and acceptance criteria specified. In the acceptance phase, one or more of the four acceptance methods is/are used to execute the dedication plan and actually perform the verification of the critical characteristics.

The NRC's position has been that as part of the technical evaluation phase of the dedication process, one or more of the most appropriate acceptance methods of the four EPRI NP-5652 [104] methods should be selected for verification of each critical characteristic along with the most plant application-relevant verification procedure (e.g., type of test or inspection), and the most application-relevant acceptance criteria.

4.1.7.2 EPRI TR-102260 Discussion

EPRI TR-102260 [105] evaluates industry activities that occurred since the publication of NP-5652 [104] and provides update information that can be used to reduce engineering and procurement costs associated with commercial-grade dedication. Key points that should be considered when using this document in discussing software tools is that its primary focus is on materials and equipment and that no specific discussion is provided for digital equipment or software. The specific purpose of the document is to provide detailed guidance on:

- The definition of the concept of "Reasonable Assurance" and demonstration of implementation guidance on achieving reasonable assurance.
- The clarification of the intent of EPRI Report NP-5652 [104] with respect to the verification of original design requirements versus verification of safety-related functions.
- The EPRI Report NP-5652 [104] generic process application and acceptance method implementation.

- The application or enhancement of Method 4, “Acceptable Supplier/Item Performance Record” in EPRI Report NP-5652 [104] to the various utility scenarios encountered in accepting CGIs for safety-related application.
- The application considerations identified through solicitation of “Lessons Learned” scenarios.
- The acceptance and dedication of CGIs by 10 CFR Part 50, Appendix B suppliers, third party suppliers, and Nuclear Steam Supply System (NSSS) suppliers.

As part of the clarification of NP-5652 [104], this document provides a list of requirements from American National Standards Institute (ANSI) Standard N18.7, paragraph 5.2.13. One of these requirements states:

> “In those cases where the QA requirements of the original item cannot be determined, an engineering evaluation shall be conducted by qualified individuals to establish the requirements and controls. This evaluation shall assure that interfaces, interchangeability, safety, fit, and function are not adversely affected or contrary to applicable regulatory or code requirements.”

This statement will be instrumental in forming the basis for commercial-grade dedication of software tools where the vendor either cannot or will not provide documentation of the QA requirements.

Regarding the evaluation discussed above, GL 91-05 stated that in addition to form, fit, and function, manufacturing process and materials should also be considered along with a verifiable reconciliation of changes. These provisions have some relevance to software tools as well in that the tool developer’s process should be evaluated and in terms of the analogs to materials that would be applicable to software and/or tools. Control of design and material changes would be analogous to software configuration management.

The term “reasonable” is defined by the document as a level of confidence that is justifiable but not absolute. Reasonable assurance of performance must be based on facts, actions, or observations. However, the decision that reasonable assurance has been attained is inherently subjective and the judgment of reasonability may vary between different observers. Maintaining a methodology based on the guidance of this document may help to reduce the subjective nature of the judgment.

In clarification of the statement “Item received is the item specified,” this document states that the term “item specified” should be viewed in a broad, generic sense and should not be literally taken to mean that each and every technical requirement of a given item communicated (i.e., specified) in a procurement document needs to be verified to meet 10 CFR 50, Appendix B, Criterion VII. What is most important is that the identified critical requirements have been met. Additionally, care should be taken not to oversimplify the process by simply checking part or version numbers as a means of acceptance.

The document also clarifies how technical information can be obtained for CGIs. These include published or controlled documents such as supplier catalogs, supplier technical manuals, product specifications, and national codes and standards cited by the supplier. Other suggested methods include contacting the suppliers' representatives by telephone or arranging meetings.

Also clarified in this document is the difference between a critical characteristic for design and a critical characteristic for acceptance. The primary difference is that the critical characteristics for acceptance need only be based on the complexity, intended safety-related function, and performance of the item. An example of this difference for a compiler would be a requirement that it should be able to compile 1,000 lines of code per minute. While this would be a critical design requirement for the compiler it would not affect the safety-related function and would therefore not be a critical requirement for acceptance.

The document also clarifies when Method 2, "Commercial Grade Survey of Suppliers," should not be used (as the sole basis for a dedication) by incorporating the NRC staff position promulgated in GL 89-02 cited above:

(1) Acceptance Method 2, "Commercial-Grade Survey of Supplier," should not be employed as the basis for accepting items from suppliers with undocumented commercial quality control programs or with programs that do not effectively implement their own necessary controls.

(2) Method 2 should not be employed as the basis for accepting items from distributors unless the survey includes the part manufacturer(s) and the survey confirms adequate controls by both the distributor and the part manufacturer(s).

The document also contains an extensive section on the use of Method 4, "Acceptable Supplier/Item Performance Record," covering when it can and when it should not be used. The primary items in this discussion include item complexity and documentation of supplier/item performance. It can be argued that any software tool is a complex piece of software. Because of this, the document recommends caution in the use of performance records. Additionally, it recommends extreme caution if adequate documentation of the item's performance cannot be produced. Both of these positions, as well as others, are supported by examples and discussion that need not be reproduced here.

4.1.7.3 EPRI TR-106439 Discussion

EPRI TR-106439 [106] provides a framework for the treatment of commercial digital equipment and has been endorsed by the NRC in a safety evaluation report.

Commercial dedication of software tools is not directly addressed in this report. Many of the steps described in the dedication process would be difficult if not impossible to accomplish. Vendors that produce COTS software tools would likely not be interested in the time and expense of the commercial dedication process given the relatively small market. This is especially true when one considers the number of choices that developers have for compilers, linkers, IDEs, and other tools. Because of this, the burden of commercial dedication of software

tools will fall to the vendors developing the final software, who will be using the tools, or to the licensee.

Section 4.2 discusses defining and verifying critical characteristics. Several characteristics are discussed as they relate to software and would apply to software tools as well. However, a majority of the characteristics discussed would not apply to software tools. The final report of this study will need to develop an acceptable set of critical characteristics for safety-related software tools. This set of characteristics may possibly be based on the characteristics listed in IEEE Std. 1209-1992 [82] and IEEE Std. 1462-1998 [86].

One of the key concepts discussed in this document is that of engineering judgment. Engineering judgment must be based on quantifiable critical characteristics and adequate evidence. The documentation of the judgment should be sufficient to allow a comparably qualified individual to come to the same conclusion. For software tools these documents should include any and all documentation from the original manufacturer, results of any testing performed by the tool user (software developer), and the results of any audits, surveys, or monitoring that may have been performed at a minimum.

Another requirement of this document, and many others, is configuration management. This can be difficult in COTS software tools because the original manufacturer (e.g., Microsoft) may include updates to software tools along with regular operating system updates. It is imperative that these “automatic” updates are accounted for and that as much information about the update as possible be included in the software tool documentation set.

4.1.7.4 EPRI TR-107339 Discussion

EPRI TR-107339 [107] gives more detailed information on how to conduct the evaluations described in TR-106439 [106]. For the purposes of this study, the primary focus is in the commercial dedication process applied to digital, software-based equipment and systems.

Section 5.1 reiterates the studies of software-based systems that show that a large fraction of problems with software is attributable to problems in the requirements specifications. It recommends a controlled procedure or methodology for development of system requirements similar to that described in EPRI TR-108831 [123]. Based on the system requirements, a set of critical requirements can then be generated for the commercial-grade dedication process or evaluation of the software tools used.

There is one important point to note in the discussion of commercial-grade dedication of the software portions of instrumentation and control systems as described in this and other documents. That is that the difficulty in using testing as a tool to determine the quality of the software is that testing does not normally take into account software behavior under abnormal conditions and events. This differentiates system software from software tools in that software tools are normally used under “ideal” conditions. Any abnormal event or condition that occur during testing or use of software tools can be eliminated simply by retesting or reprocessing.

4.1.7.5 EPRI TR-1025243 Discussion

TR-1025243 [108] does not meet the requirements of 10 CFR Part 50, Appendix B, or 10 CFR Part 21; however, it can provide some insight into acceptance of commercial-grade computer programs via the dedication process. This report provides methodology that can be used to perform safety classification of non-process computer programs, such as design and analysis tools, that are not resident or embedded (installed as part of) plant systems, structures, and components. The report also provides guidance for using commercial-grade dedication methodology to accept commercially procured computer programs that perform a safety-related function.

Included in this document is a clarification of "commercial-grade items" as related to computer programs. The document references the NRC Safety Evaluation Report (SER) of TR-106439 [106] in which it states:

> "The second sentence of the new Part 21 definition of CGI excludes "items where the design and manufacturing process requires many in-process inspections and verifications to ensure that defects or failures to comply are identified and corrected (i.e., one or more, critical characteristics of the item cannot be verified)." The staff considers verification and validation activities common to software development in digital systems to be a critical characteristic that can be verified as being performed correctly following the completion of the software development by conducting certain dedication activities such as audits, examinations, and tests."

When this wording in 10 CFR Part 21 was first promulgated in a 1995 rulemaking, it generated much controversy. In a public meeting between the cognizant NRC staff and the Nuclear Management Resources Council (NUMARC), now the Nuclear Energy Institute (NEI), and other industry stakeholders, the NRC staff acknowledged that the first part of the wording that excluded from eligibility for dedication items for which the design and manufacturing process requires many in-process inspections and verifications to ensure that defects or failures to comply are identified and corrected could have better expressed the staff's intent. The staff explained that its basic position was more accurately stated in the parenthetical statement that followed, "(i.e., one or more, critical characteristics of the item cannot be verified)." Dedication under the conditions in question could be acceptable, but would require the use of EPRI NP-5652 Method 3, source verification. The staff explained that the exclusory wording was not intended to exclude EPRI acceptance Method 3 of NP-5652, source verification, in which in-process inspections and verifications by a purchaser's representative, or by the manufacturer and witnessed by the purchaser's representative, provided that all critical characteristics could be verified either during or after manufacture. Rather it was intended only to exclude from eligibility for dedication CGIs for which one or more critical characteristics were not or could not be verified during or after manufacture. This position is reflected in the statement quoted above from the NRC SER that accepted TR-1025243. In addition, there is presently a rulemaking process in progress, which ostensibly is supposed to clarify this position among others in 10 CFR Part 21.

The focus of TR-1025243 [108] is “commercially procured computer programs for use in safety-related design and analysis applications.” Although these types of computer programs are outside the scope of this study, the methods presented can be used in the commercial-grade dedication of any computer programs.

The document presents three scenarios of procuring and using computer programs in safety-related applications. Scenario A is a computer program procured as a basic component. Scenarios B and C consider computer programs procured as commercial items. Scenario B focuses on the output of the computer program while scenario C focuses on the program itself.

The document is divided into eight sections. Sections 1 and 2 contain introductory material. Sections 3 through 5 cover general practices applicable to all three of the presented scenarios. Section 3 discusses computer program categories and uses. Section 4 discusses generic technical evaluation and acceptance processes. Section 5 covers functional safety classification of computer programs. Section 6 covers the commercial-grade dedication process of computer programs falling under Scenario C with examples in Section 7. Section 8 contains the references and bibliography.

4.1.7.6 TR 107330 Discussion

EPRI TR-107330 [109] provides a specification that is suitable for evaluating a particular programmable logic controller (PLC) product line as a platform for safety-related applications, establishing a suitable qualification test program, and confirming that the manufacturer has a quality assurance program that is adequate for safety-related applications or is sufficiently complete that it can be brought into conformance. The primary focus of this document is the hardware portion of a PLC system however; it does contain discussion of the software and software tools used in the system.

Two types of software are discussed in this document, application software for a specific implementation and the executive software (operating system) that is resident in the PLC independent of the application. According to the specification, the executive software has been historically more difficult to qualify because of a lack of rigor and/or documentation of the software development process. It continues by noting that this trend has been improving due to the increased use of international standards by PLC vendors. It should not be assumed that this would automatically apply to the software tools developed by the vendor or its suppliers. Compliance with standards must be documented and verified.

The guidelines provided by this document regarding software tools include a list of required features, a list of debugging tools, requirements for configuration management, and requirements to assist in validation of the application program. Key features listed include the ability to perform a bit by bit comparison between the program on the PLC and the program on a programming device and the ability to implement security measures preventing on-line application or executive program changes and unauthorized access. Similar features can be applied to other types of software tools used in non-PLC digital equipment.

4.1.8 National Institute of Standards and Technology Discussion

4.1.8.1 NIST SP 500-234 Discussion

NIST SP 500-234 [110] is a research report presenting V&V guidelines for software developed for the health care environment. In addition to discussing V&V activities performed during software development, the document discusses V&V activities to be performed on COTS software. The document breaks down the V&V activities into tasks that should be performed at each step of the development life cycle. Tasks for reuse of software, COTS software, are also listed for each step providing a step-by-step guide for V&V of COTS software.

The document lists nine V&V activities in Table 2-1. These activities are then discussed in detail in Sections 2.3.1 through 2.3.8. Each section contains reuse specific guidance under the subtitle "Reuse-Specific." These sections will be analyzed further in later tasks. Additional guidance for reuse software is provided in Section 4. This section discusses concerns with using reuse software and assessing software for reuse.

4.1.9 Atomic Energy of Canada Limited Discussion

4.1.9.1 AECL CE-1001-STD Discussion

AECL CE-1001-STD [139] is a Canadian standard jointly prepared by Ontario Power Generation, Nuclear and Atomic Energy of Canada Limited. This standard specifies the software engineering requirements for safety-critical software used in real-time protective, control, and monitoring systems. The scope and format of this standard is similar to IEEE Std. 12207-2008 [2]. The standard defines a minimum set of software engineering processes to be followed in creating and revising the software, a minimum set of outputs to be produced by the processes, and requirements for the content of the outputs. The processes in this standard are simpler than the processes in IEEE Std. 12207-2008 [2] and follow the lifecycle processes described in CE-0100-STD [140]. Chapter 2 of the standard describes a simple sequential life cycle model consisting of development, verification, validation, and support processes.

The standard only discusses software tools in the context of proper identification and documentation in various reports and plans. Section 4.4.2.2(k) of the standard states that the reliability qualification procedure should identify all equipment, tools, and support software required to produce the input and expected result data for the test cases used in the verification and validation processes. Section 4.5.1(d) states that all test procedures should identify all equipment, tools, and support software required to perform the test performed in the verification and validation processes. Section 4.5.3(a) states that all reports should summarize the activities performed and the methods and tools used and Section 4.5.3(j) states that all reports should identify the actual equipment, tools, and support software used in the verification and validation processes.

Section 5.2 discusses the requirements of the software development plan (SDP). The SDP is a comprehensive plan for the management of the software engineering process consisting of the project plan, the documentation plan, the configuration management plan, and the training plan. Section 5.2.1.3 states that the project plan portion of the SDP shall (1) specify facilities, tools,

and aids to be used during development, verification and support; (2) identify personnel required to maintain and manage the facilities, tools, and aids; and (3) identify the qualification requirements for facilities, software tools, and pre-developed software for the target system. Section 5.2.2(h) states that the documentation plan portion of the SDP shall identify facilities, tools, and aids used to produce, manage, and issue documentation. Section 5.2.3(a) states that the configuration management plan portion of the SDP shall uniquely identify all categories of configuration items that form a developmental baseline for software configuration management including support tools (compilers, linkers, operating systems). Section 5.2.4(f) and (g) state that the training plan portion of the SDP shall identify requirements for training courses and materials with respect to the use of software tools and methodologies and identify training facilities, tools, responsibility for training, and the contents of the training records.

Section 5.3 discusses the requirements of the standards and procedures handbook (SPH) which defines the standards and procedures that must be defined and used. The SPH consists of process standards and procedures, documentation standards and procedures, configuration management standards and procedures, and training standards and procedures. Section 5.3.1(k) states that the process standards and procedures shall identify procedures for using tools and facilities in the development, verification, and support processes. Section 5.3.1(l) states that the process standards and procedures shall identify or reference pertinent documents that identify all verification tools, techniques, checklists, procedures, and databases, their proper use, and current versions.

4.2 Documents Not Relevant to Software Tools

4.2.1 Space Industry Documents Discussion

4.2.1.1 NASA-GB-A301 Discussion

NASA-GB-A301 [11] is a second-tier guidebook that defines audits, describes the audit process, and provides a sample checklist that can be tailored for use in an audit. The term “audit” refers to a SQA technique that is used to examine the conformance of a development process to procedures and the conformance of products to standards.

Standards are “established criteria to which software products are compared” including documentation standards, design standards, and coding standards. Procedures are “established criteria to which the development and control processes are compared” including step-by-step directions that are to be followed to accomplish some development or control process. Standards and procedures are requirements for software management, engineering, and assurance; SQA audits verify their existence and assess a project’s compliance with them.

This guidebook describes how to conduct an audit, including planning, visits to software providers, audit reporting, and provider follow-up to correct problems. The guidebook describes the best times to schedule audits within each life cycle phase and discusses how to tailor a generic checklist to meet the project’s need and the desired qualities of an auditor. The guidebook only mentions a single software tool; a standards checker for code that can be used by the auditor if the auditor verifies that the project runs the checker.

This guidebook does not address software tool qualification, verification, validation, regulation, or approval/review practices. A detailed survey of this document was not performed or warranted since the document was not relevant to software tool regulation, guidance, or review/approval practice.

4.2.1.2 NASA-GB-001-94 Discussion

NASA-GB-001-94 [13] is a guidebook describing the use of one specific software engineering tool, i.e., software measurements (a.k.a. metrics). It describes the purpose and importance of measurement, presents specific procedures and activities of a measurement program, and clarifies the role that measurement can and must play in the goal of continual, sustained improvement for all software production and maintenance efforts throughout NASA. The guidebook includes an introduction to software measurement and is aimed at organizations beginning or improving a measurement program. Measurement provides the only mechanism available for quantifying a set of characteristics about a specific environment or for software in general. Increased understanding leads to better management of software projects and improvements in the software engineering process.

This guidebook highly recommends using a COTS relational database management system (DBMS) to support the organization's measurement program. The guidebook states that the time and effort required to develop custom tools outweigh their benefit. Verification, validation or qualification of this COTS DBMS is not discussed.

Chapter 5 of the guidebook contains a section devoted to a discussion of Tools and Automated Aids. Tools and automated aids are built to facilitate software management, development, maintenance, or data collection processes. The chapter refers to three such tools including a cost estimation aid, a management aid that compares actual measured values with baseline estimates, and design aids driven by experimental results indicating beneficial design approaches. In addition, the chapter states that tools that are more sophisticated may use historical information for managing and for analysis. An example of such a tool is the software management environment (SME). The tool encapsulates experience (i.e., data, research results, and management knowledge) gained from past development projects to assist software development managers in their day-to-day management and planning activities. This tool does not impact final plant software and has no direct relevance to this software tool research project other to acknowledge its use by NASA.

This guidebook also mentions software tools as being an important part of software measurement; however, the guidebook only describes how to use software measurement tools to establish a database, collect information, and analyze the information that can be used in future software engineering applications to better plan and resource load the project. The guidebook does not mention tool qualification, verification, validation, regulation, guidance, or approval/review practice and thus has no relevance to the software tool research project.

4.2.1.3 NASA-GB-001-95 Discussion

NASA-GB-001-95 [14] is a guidebook for defining, operating, and implementing a working software process improvement program. Both software process and software product are emphasized in the guidebook. Improvements in software process result in measurable improvements to the software product. This guidebook addresses the needs of the NASA software community by offering a framework based on an evolutionary approach to quality management tailored for the software business. It presents an overview of concepts pertaining to software process improvement and the three-phase software process improvement approach (understanding, assessing, and packaging). In the *Assessing Phase,* specific objectives for improvement are set, one or more changes are introduced into the current process, and the changes are then analyzed to assess their impact on both product and process. Change may include introducing a method, tool, or management approach. In the *Packaging Phase,* the products developed by the analysts are stored by the support staff into a repository (i.e., an experience base) and are provided to the developers upon request. Packaging typically includes standards, policies, and handbooks; training; and tools.

The guidebook presents the structure of a typical NASA software process improvement program and describes the major components of the software process improvement organization. It gives an overview of each of the three organizational components (developers, analysts, and support staff) and discusses the resources each component requires. The developers have the global responsibility for using, in the most effective way, the packaged experience base to deliver high-quality software. The analysts produce and provide baselines, tools, lessons learned, and data, parameterized in some form in order to be adapted to the characteristics of a project. The support staff sustains and facilitates the interaction between developers and analysts by saving and maintaining the information, making it efficiently retrievable, and controlling and monitoring access to it. They use tools that assist in collecting, validating, and redistributing data and reusable experience.

Although this guidebook mentions the use of software tools as part of the process improvement program, it does not discuss software tool requirements, qualification, verification, validation, regulation, guidance, or review/approval practices. A detailed survey of this document was not performed or warranted since the document was not relevant to software tool regulation, guidance, or review/approval practice.

4.2.2 Civil Aviation Industry Documents Discussion

4.2.2.1 FAA AC 90-111 Discussion

FAA AC 90-111 [33] provides a validation process for two types of software tools used to develop instrument flight procedures (IFP). An IFP is a charted flight path defined by a series of navigational fixes, altitudes, and courses provided with lateral and vertical protection from obstacles from the beginning of the path to a point from which a landing can be completed. The two types of software tools are aiding tools and expert tools. An aiding tool is a software system that implements some aspect of instrument flight procedure design, but does not integrate all of the procedure design functions in a single system. An expert tool is a software system that

provides a highly integrated instrument procedure design environment in which many or all criteria-specific computations are carried out automatically.

Aiding and expert software tools used to develop IFPs are not part of airborne software; therefore, this advisory circular is not relevant to guidance, regulation, or review/approval of software tools used in the development of software in airborne systems.

4.2.3 NRC and Commercial Nuclear Power Industry Documents Discussion

4.2.3.1 RG 1.153 Revision 1 Discussion

RG 1.153, Revision 1 [36] discusses the GDC from Appendix A to 10 CFR Part 50 that are applicable to the power, instrumentation, and control portions of nuclear power plant safety systems. This regulatory guide also states the requirements of 10 CFR 50.55a and 10 CFR 50.49 and describes a method acceptable to the NRC staff for complying with NRC regulations with respect to the design, reliability, qualification, and testability of the power, instrumentation, and control portions of safety systems of nuclear plants.

This regulatory guide states that conformance with the requirements of IEEE Std. 603-1991 [37] provides a method acceptable to the NRC staff for satisfying the NRC's regulations with respect to the design, reliability, qualification, and testability of the power, instrumentation, and control portions of the safety systems of nuclear power plants. IEEE Std. 603-1991 [37] establishes minimum functional and design requirements for the power, instrumentation, and control portions of safety systems for nuclear power plants.

Section 1.2 of IEEE Std. 603-1991 [37] references IEEE 7.4.3.2-1982 [124]; however, RG 1.152, Revision 3 [34] endorses IEEE 7.4.3.2-1993 [35]. Therefore, RG 1.152, Revision 3 [34] constitutes an acceptable method of meeting the regulatory requirements for digital computers. Section 5.8.1 of IEEE Std. 603-1991 [37] references IEEE Std. 497-1981 [125]; however, Revision 3 of RG 1.97 [126] provides an acceptable method to meet the regulations for accident monitoring instrumentation.

This regulatory guide does not discuss software tool requirements, qualification, verification, validation, regulation, guidance, or review/approval practices.

4.2.3.2 NUREG/CR-6101 Discussion

NUREG/CR-6101 [66] discusses development of safety-related software. Section 2 provides an in-depth discussion of terms associated with software safety, reliability, robustness, and other measurable and non-measurable traits as a foundation for subsequent sections. It also provides a discussion of the software life cycle process and examples of how life cycle processes fit into several different life cycle models.

Section 3 describes activities in the life cycle processes that can affect reliability and safety. Since the report is written from the perspective of an assessor, a majority of the discussion concerns the reports that should be provided for audit and the reports that should be generated during reviews. The various specification and design phase plans are presented in condensed

form and reference their respective IEEE standards (e.g., IEEE Std. 1074-1991 [119] for software development plans and IEEE Std. 1228-1994 [128] for software safety plans). Section 4 reviews the life cycle activities discussed in Section 3 and covers questions that should be asked by the assessors.

The Appendix to NUREG/CR-6101 [66] discusses some of the technical aspects of concepts presented in the report. No specific discussions of software tools are found in this document, however, the concepts presented concerning review and assessment of the various steps of the life cycle process can be used in reviewing the documentation provided for the software tool development process.

4.2.4 Institute of Electrical and Electronics Engineers Documents Discussion

4.2.4.1 IEEE Std. 279-1971 Discussion

IEEE Std. 279-1971 [72] addresses criteria for protection systems for nuclear power generating stations. These criteria establish minimum requirements for the safety-related functional performance and reliability of protection systems for stationary land-based nuclear reactors producing steam for electric power generation.

This document discusses general, high-level protection system functionality including redundancy, single failure criteria, channel out of service for testing, and the functional reliability of the protection system under normal and abnormal conditions. As expected from a 1971 document, computer software and software tools are not discussed in the standard. This standard has been withdrawn due to obsolescence.

4.2.4.2 IEEE Std. 323-2003 Discussion

IEEE Std. 323-2003 [73] describes the basic requirements for qualifying Class 1E equipment and interfaces that are to be used in nuclear power generating stations. As such, it is a standard focused on hardware rather than software and contains no useful information concerning software tools.

4.2.4.3 IEEE Std. 1540-2001 Discussion

IEEE Std. 1540-2001 [88] describes the risk management processes that should be employed during the software development life cycle. No specific mention is made to the development of software tools. However, the risk management processes described can be used in any development process to improve the quality of the final software product. This standard has been superseded by IEEE Std. 16085-2006 [89].

4.2.4.4 IEEE Std. 16085-2006 Discussion

IEEE Std. 16085-2006 [89] is a revision of IEEE Std. 1540-2001 [88] and incorporates ISO/IEC 16085:2006 [129]. The primary function of the standard is to define the risk management processes as they apply to the software life cycle defined in ISO/IEC 12207:1995 [90] and

ISO/IEC 15288:2002 [92]. Besides the element of risk management in development of software tools, this standard adds little to the study.

4.2.5 National Institute of Standards and Technology Discussion

4.2.5.1 NIST IR 4909 Discussion

NIST IR 4909 [111] examines the contents of a software quality assurance standard for nuclear applications and includes recommendations for the documentation of software systems. The study consists of a comparison of ASME NQA-2a-1990 [130] and IEEE Std. 730.1-1989 [131]. The executive summary specifically points out that the recommendations of the report may not be valid if CASE tools are used. Taking this into consideration, and the fact that the IEEE Std. 730.1-1989 [131] has been superseded by active IEEE Std. 730-2002 [116], it is the opinion of this reviewer that this document provides little or no additional useful input to the current study.

5 Baselines

During the survey of various international standards, regulatory guidance documents, and documents describing “best practices,” several basic guidelines were stated in similar forms. These guidelines cover a majority of the software tools that are of concern for this study. Additional guidelines, that were limited to one or two documents, were not included in our baseline either because they did not apply to software tools used in the development of software important to safety in a nuclear environment or because the technical basis did not meet the standards of the survey team.

The baseline requirements and guidance presented in this section are divided into two subsections. The first subsection describes the requirements presented by international standards and regulatory documentation. The second describes guidance and review/approval practice from other documentation. The baseline requirements and baseline guidance and review/approval practices are organized according to the applicable Institute of Electrical and Electronics Engineers (IEEE) 12207-2008 [2] process.

The purpose of the tool qualification process is to obtain confidence in the tool functionality. The tool qualification effort varies based upon the potential impact that a tool error could have on the system safety and upon the overall use of the tool in the software life cycle process. The higher the risk of a tool error adversely affecting system safety, the higher the rigor required for tool qualification {RTCA DO-330 [31], Section 2.0}.

The determination of which tools should be subject to regulatory guidance is based on several factors. These include:

1. whether the tools have a direct effect on the final software output and result in the creation of software errors (e.g., autocode generators);
2. whether the tools have an indirect effect on the final software output and result in the creation of software errors (e.g., failure to resolve quality assurance (QA) audit and review anomalies);
3. whether they are used in the verification and validation (V&V) process and may fail to detect errors;
4. or
    a. if the use of the tool can be used to justify bypassing other verification and validation processes which could result in failure to detect errors or
    b. if they can be used to justify elimination of software constructs that would have been used in fault handling which could result in failure to react properly to errors.

5.1 Requirements Baseline

The output of software tools used in the life cycle process of safety-related software shall be confirmed as suitable for use in safety-related systems by one or both of the following methods. The output shall be subjected to the same level of verification and validation as the safety-related software or the tool shall be developed using the same or an equivalent high-quality life cycle process as required for the software upon which the tool is being used. If this is not possible, the tool shall be commercially dedicated to provide confidence that the necessary features of the software tool function as required. {IEEE Std. 7-4.3.2-2010 [1], Section 5.3.2} The following requirements can assist in this process.

5.1.1 Organizational Project-Enabling Processes

1. All project tools that could potentially impact safety-critical software, the degree of impact, and mitigation strategies shall be discussed in the project plan. {NASA-STD-8719.13B [9], Clause 5.11.1.1}

2. The process and criteria used to select, approve, and control the tools shall be described in the project plan. Installation of upgrades to approved tools, withdrawal of previously approved tools, and identification of limitations of the tools should be identified. {NASA-STD-8719.13B [9], Clause 5.11.1.2}

3. Tools to be used should be identified in the development plan. {IAEA NS-G-1.1 [138], Section 4.6}

4. The entire project should be planned in advance to select an integrated set of tools. {IAEA NS-G-1.1 [138], Section 4.6}

5. All software tools shall be identified and shall be addressed within required software development activities and documentation. Validation that the tool performs its required functions and operates correctly within the software system design shall be accomplished. {FAA-STD-026A [16], Clause 4.4}

6. The software tool should be used in a manner such that defects not detected by the software tool will be detected by V&V activities. {IEEE Std. 7-4.3.2-2003 [35], Clause 5.3.2.b}

7. If it cannot be demonstrated that defects not detected by software tools or introduced by software tools will be detected by V&V activities, the software tool should be designed as safety-related software itself, with all the attendant regulatory requirements for safety software. {NUREG 0800 Appendix 7.1-D, Clause 5.3.2; IEEE Std. 7-4.3.2-2003 [35], Clause 5.3.2.b}

5.1.2 Project Processes

1. Software tools should support the development activities that contribute to the correctness of software and system design. {IEC 62138 Ed. 1.0B [97], Clause 5.1.4}

2. The equipment families used for the development of an Instrumentation and Control (I&C) system should be associated with software tools that can reduce the risk of introducing faults in new application software. {IEC 62138 Ed. 1.0B [97], Clause 5.1.4}

3. The equipment families used for the development of an I&C system should be associated with software tools that can reduce the risk of introducing faults in the configuration of their pre-developed software and in the design of the system. {IEC 62138 Ed. 1.0B [97], Clause 5.1.4}

4. Evidence should be provided regarding the quality of the software tools that might introduce faults in software or in system design, and regarding their ability to produce correct results. {IEC 62138 Ed. 1.0B [97], Clause 5.1.4}

5. Many modern equipment families are provided with extensive application-oriented development tools that enable plant or system engineers to specify their requirements using graphical techniques. The tools may automatically translate the graphical programs into executable application software. When these tools are of adequate quality, this approach should be used to reduce the risk of faults. {IEC 62138 Ed. 1.0B [97], Clause 4.1}

6. The software tools that might introduce faults in software or in system design shall be selected and used according to documented procedures and rules aiming at reducing this risk. Evidence shall be provided regarding their quality and their ability to produce correct results. Their use shall be traced so that the tools, if any, which were used to generate a given item or information may be identified. {IEC 62138 Ed. 1.0B [97], Clause 6.1.4}

7. The software tools that might lead to faults already existing in software or in system design being overlooked should be selected and used in a way which reduces this risk. Their use should be traced. {IEC 62138 Ed. 1.0B [97], Clause 6.1.4}

8. When a tool or tool version that has the potential to introduce faults in software or in system design is substituted with another, reasonable precautions shall be taken to ensure that this does not have adverse effects on the correctness of the software and the system design. {IEC 62138 Ed. 1.0B [97], Clause 6.1.4}

9. When activities of the software development life cycle process are automated using software tools, the activities automated shall be documented including the documentation of the inputs and outputs relevant to the life cycle phase. {IEC 60880 Ed.2 [96], Clause 5.4.6}

10. It shall be possible to identify all translation tools and tool versions used to produce each executable entity. {IEC 60880 Ed.2 [96], Clause 5.6.12}

11. Criteria for the selection and evaluation of tools for the software engineering environment should be developed and prioritized to allow trade-offs prior to use. The criteria should be structured by software quality characteristics as defined in ISO/IEC

9126:1991 [120] including functionality, reliability, usability, efficiency, modifiability, and portability. The criteria may include other items like licensing effort and resources required to use a tool, the rigor of the quality plan under which the tool was developed, vendor tool history, and alternatives to tool use. {IEC 60880 Ed.2 [96], Clause 14.3.1.2}

12. Tools used to provide diversity, i.e., compilers used for the development of multiple version dissimilar software systems, should be demonstrated to be dissimilar. This may be achieved by showing that: {IEC 60880 Ed.2 [96], Clause 14.3.1.5}

    a. each tool was obtained from a different supplier (for example, one tool could be developed and the other tool could be purchased off the shelf); or

    b. the tools have different input and/or output languages; or;

    c. the tools have dissimilar requirements and design processes.

13. A tool qualification strategy shall be produced and the tools shall be qualified in accordance with that strategy. The strategy shall consider the reliability requirements of the tool and the type of the tool. {IEC 60880 Ed.2 [96], Clause 14.3.2.1}

14. The qualitative reliability requirements of a tool shall be determined considering: {IEC 60880 Ed.2 [96], Clause 14.3.2.2}

    a. the consequences of a fault in the tool;

    b. the probability that a tool causes or induces faults in the software implementing the safety function;

    c. what other tools or processes mitigate the consequences of a fault in the tool.

15. Documentation associated with computer-aided software engineering (CASE) tools, data in these tools, compilers, test tool, test data, simulations, emulations, utilities, configuration management tools, databases and data files, and other software shall include: {FAA-STD-026A [16], Clause 10.2.3.3}

    a. Specific names, identification numbers, version numbers, release numbers, and configurations:

    b. Rationale for the selected software;

    c. Reference to user/operator manuals or instructions for each item, as applicable;

    d. Identification of each software item and document as Government-furnished, as item that will be delivered to the support organization, as item the support organization is known to have, as item the support organization must acquire, or other description of status;

e. If items must be acquired, information about a current source of supply, including whether the item is currently available and whether it is expected to be available at the time of delivery;

f. Information about vendor support, licensing, and data rights, including whether the item is currently supported by the vendor, whether it is expected to be supported at the time of delivery, whether licenses will be assigned to the support organization, and the terms of such licenses;

g. Security and privacy considerations, limitations, or other items of interest.

16. Diversity of methods, languages, tools, and personnel should be taken into consideration to reduce the potential for software common-cause failures. {IAEA NS-G-1.1 [138], Section 3.13}

17. Tools should be selected to facilitate the proper application of the methods, standards, and procedures selected. {IAEA NS-G-1.1 [138], Section 4.6}

5.1.3 Technical Processes

1. Software tool review criteria should conform to the characteristics described in IEEE Std. 14102-2010 [83]. {IEEE Std. 14102-2010 [83], Clause 10}

2. Software tools should be evaluated and selected in accordance with the processes described in IEEE 14102-2010 [83]. The documentation of this evaluation should be used by the approving authority as evidence of the tool's acceptability. The tool requirements defined by the licensee or software vendor and the critical characteristics and sub-characteristics of the tool shall be reviewed against the requirements for the safety-related software being approved. Particular attention should be given to the weighting factors assigned to the various characteristics and sub-characteristics by the licensee or software vendor.[2] {IEEE Std. 14102-2010 [83], Clause 10}

3. All tools used should be mutually compatible. The tools should be qualified to a level commensurate with their function in the development of the software and in the safety demonstration. The techniques used to gain confidence in the tools should be specified and documented. {IAEA NS-G-1.1 [138], Section 3.25}

4. Well established methods, languages, and tools for software development should be used. Methods, languages, and tools that are still at the research stage should not be used. {IAEA NS-G-1.1 [138], Section 3.29}

5. Tools used to analyze the consistency and completeness of the computer system requirements should be validated to a level of confidence proven equivalent to the level required from the design process. {IAEA NS-G-1.1 [138], Section 5.9}

[2] The process for defining software tool requirements is described in Clause 7.2 of IEEE 14102-2010 [83]. Critical characteristics of software tools are described in Clause 10.2 to 10.5 of IEEE 14102-2010 [83].

6. Any separately compiled modules that are included by an automatic code generator should be supported by separate documents for the software requirements. {IAEA NS-G-1.1 [138], Section 7.4}

7. A well-known tool with extensive operating experience is preferable to an untried tool with little or no operating experience as long as the well-known tool meets other selection criteria. {IEC 62138 Ed. 1.0B [97], Clause 5.1.4.1}

8. When dealing with COTS software tools, operating experience is often the deciding factor in determining a tool's quality. Additional evidence may be based on tool certification, certification of the suppliers, and tests. {IEC 62138 Ed. 1.0B [97], Clause 5.1.4}

9. The compatibility of a tool with other tools in use should be considered along with the tool quality. {IEC 62138 Ed. 1.0B [97], Clause 5.1.4.1}

10. Translators/compilers shall not remove without warning defensive programming or error-checking features introduced by the programmer. {IEC 60880 Ed.2 [96], Clause 14.3.4.2}

11. The use of compiler optimization should be avoided. It shall not be used if it produces object code that is excessively difficult to understand, debug, test, and validate. {IEC 60880 Ed.2 [96], Clause 14.3.4.3}

12. Where optimization is used, tests, verification and/or validation shall be performed on the optimized code. {IEC 60880 Ed.2 [96], Clause 14.3.4.4}

13. Libraries which are used in the target system shall be considered as sets of pre-developed software components. Those components of the library used shall be evaluated, qualified, and used in accordance with the requirements of pre-developed software. {IEC 60880 Ed.2 [96], Clause 14.3.4.5}

14. Tests, verification and/or validation shall be carried out to ensure that any additional code (assembly instructions) introduced by the translator which is not directly traceable to source line statements (for example, error checking code, error and exception handling code, initialization code) is correct. {IEC 60880 Ed.2 [96], Clause 14.3.4.6}

5.1.4 Software Implementation Processes

1. Tools that insert or translate code (e.g., optimizing compilers, auto-code generators) shall be assigned the same integrity level and verified and validated to the same rigor as the highest software integrity level assigned to the software element that the tool affects. {IEEE Std. 1012-2004 [42], Clause 4; IEEE Std. 1012-2012 [79], Clause 5}

2. Tools that do not insert code shall be verified and validated to assure that they meet their operational requirements. {IEEE Std. 1012-1998 [39], Section 7.4.6}

3. If a tool cannot be verified and validated, then the output of the tool shall be subject to the same level of V&V as the software element. {IEEE Std. 1012-2012 [79], Clause 5}

4. If tools do not insert code and partitioning of tool functions can be demonstrated, only those functions that are used in the V&V processes shall be verified to demonstrate that they perform correctly for their intended use. {IEEE Std. 1012-1998 [39], Clause 7.4.6}

5. Integrity-level determination quantifies the complexity, criticality, risk, safety level, security level, desired performance, reliability, or other system-unique characteristics based on the importance of the system to the user and acquirer. The degree of rigor and intensity in performing and documenting the task shall be commensurate with the integrity level. {IEEE Std. 1012-2012 [79], Clause 5}

6. Validated tools should be used since they relieve programmers from tasks of the code production process that are prone to manual error, such as programming and verification. {IAEA NS-G-1.1 [138], Section 9.10}

7. A suitable set of tools should be selected for the implementation including translators, code generators, debuggers, linkers, testing, and other verification tools. {IAEA NS-G-1.1 [138], Section 9.11}

8. Thoroughly tested translators that are available and maintainable throughout the system lifetime should be used. If no thoroughly tested translator is used, additional analysis and verification, manual or by the use of other tools, should demonstrate that the translation is correct. {IAEA NS-G-1.1 [138], Section 9.12}

9. Any tools used in the software implementation phase should be compatible with other tools used in other phases of the development. {IAEA NS-G-1.1 [138], Section 9.13}

5.1.5 Software Support Processes

1. A test-tool validation program should be developed to provide confidence that the necessary features of the software tool function as required. {IEEE Std. 7-4.3.2-2003 [35], Clause 5.3.2.a}

2. Technical independence requires the independent V&V effort to use its own set of test and analysis tools separate from the developers' tools. {IEEE Std. 1012-2012 [79], Annex C.1}

3. A software quality assurance plan should be produced at the outset of the project including identification of tools to be used on the project and the procedures for qualifying the tools. {IAEA NS-G-1.1 [138], Section 4.11}

4. The teams performing verification and validation should be independent of the development team using different techniques and tools. {IAEA NS-G-1.1 [138], Section 4.17}

5. All items of software development, such as compilers, development tools, configuration files and operating systems, should be under configuration management control. {IAEA NS-G-1.1 [138], Section 4.20}

6. Software assurance personnel shall be trained in relevant software engineering tools so they stay current with the engineering environment and products they must assure. {NASA-STD-8739.8 [8], Section 5.3.1.3}

7. The quality assurance plan should precisely identify the software tools which may influence the correctness of software and/or system design. {IEC 60880 Ed. 2.0 [95], Clause 5.1.4.4}

8. The quality assurance plan shall distinguish the tools that might introduce faults in software or in system design, from those which only might lead to overlooking already existing faults. {IEC 62138 Ed. 1.0B [97], Clause 6.1.4}

9. Automated validation tools that generate test data, transport, or transform test data and test results and evaluate test results should record a complete test log. This is applicable to module tests as well as to plant simulations. {IEC 60880 Ed.2 [96], Clause 14.3.6.1}

10. Appropriate tools should be used to test and/or simulate the behavior of the executable code loaded in the target system. {IEC 60880 Ed.2 [96], Clause 14.3.6.2}

11. Appropriate tools shall be used to ensure or verify that the right executable code is loaded correctly in the target system. {IEC 60880 Ed.2 [96], Clause 14.3.6.3}

5.1.6 Software Reuse Processes

1. The integrity level assigned to reused, COTS, and GOTS tools shall be in accordance with the integrity-level scheme adopted for the system element in which COTS or GOTS tools are used for the project. The reused COTS or GOTS tools shall be evaluated for use in the context of its application. The design, development, procedural, and technology features implemented in the system can raise or lower the assigned integrity levels. {IEEE Std. 1012-2012 [79], Clause 5}

2. If V&V of reused software tools cannot be accomplished at the appropriate level, then the items may be used so long as the risk associated with this use is recognized and accounted for in the risk mitigation strategy. The V&V effort should assure that the risks are thoroughly understood, properly documented, and properly tracked under the risk analysis tasks. {IEEE Std. 1012-2012 [79], Appendix D.3}

5.2 Guidance, Review and Approval Practice Baseline

Guidance for the use, review, and approval of software tools that are used to develop software for systems important to safety is found in documents from many different industries. Some of this guidance has been organized below to assist in meeting the requirements in the previous section of the report.

5.2.1 Organizational Project-Enabling Processes

1. The software development plan should include the following resource characteristics. {NUREG-0800 BTP 7-14 [64], Section B.3.1.2.3; RTCA/DO-330 [31], Section 4.1}

    a. A description of the software development methods, techniques, and tools to be used;

    b. A description of the approach to be followed for reusing software;

    c. Identify suitable facilities, tools, and aids to facilitate the production, management, and publication of appropriate and consistent documentation and for the development of the software;

    d. Describe the software development environment, including software design aids, compilers, loaders, and subroutine libraries;

    e. Require that tools be qualified with a degree of rigor and level of detail appropriate to the safety significance of the software that is to be developed using the tools;

    f. The tool qualification level is defined; and

    g. The tool operational environment is described.

2. When special purpose software is used for environmental qualification testing, the manufacturer must verify, validate, and control the configuration of the special purpose test software. The test software should be included as part of the test setup conformity conducted before the qualification testing. {FAA Order 8110.49 [21], Section 4-3.a.1}

3. The applicant's planning documents should describe how the applicant will have visibility into their suppliers' and sub-tier suppliers' activities including COTS software tool suppliers and vendors. The applicant should submit these plans to the certification authority for review and approval. {FAA Order 8110.49 Change 1 [22], Section 13-3.b.}

4. A software tool may be qualified only for use on a specific system. Tool qualification data must be generated, a tool qualification plan must be generated and followed and tool operational requirements must be specified to describe the tool's operational functionality including all functions and technical features. {RTCA/DO-178B [20] 12.2}

5. Applicants and tool developers must discuss with the certification authority the details of each reusable tool qualification project. {FAA AC 20-148 [25], Section 2.b}

6. The software planning process should meet the following objectives. {RTCA/DO-178C [28], Clause 4.1}

    a. The software life cycle environment, including the methods and tools to be used for the activities of each software life cycle process, has been selected and defined.

b. Software plans should define the transition criteria by specifying availability of tools, methods, plans, and procedures.

7. Activities for the software planning process include: {RTCA/DO-178C [28], Clause 4.2}

   a. Methods and tools should be chosen that aid error prevention and provide defect detection in the software development process.

   b. Methods and tools should be chosen to achieve dissimilarity necessary to satisfy the system safety objectives when multiple-version dissimilar software is used in a system to mitigate software failures.

   c. If user-modifiable software is planned, related processes, tools, environment, and data items substantiating the design should be specified in the software plans and standards.

   d. When parameter data items are planned, the processes to develop, verify, and modify parameter data items, and tool qualification should be addressed.

8. Activities for the selection of software development environment methods and tools include: {RTCA/DO-178C [28], Clause 4.4.1}

   a. The use of tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together.

   b. If certification credit is sought for use of the tools in combination, the sequence of the options should be examined and specified in the appropriate plan.

   c. If optional features of software tools are chosen for use in a project, the effects of the options should be examined and specified in the appropriate plan. This is especially important for compilers and autocode generators.

   d. Known tool problems and limitations should be assessed and those issues that can adversely affect software should be addressed.

5.2.2 Project Processes

1. Reviewers should thoroughly evaluate tool usage. {NUREG-0800 Appendix 7.1-D [65], Section 5.3.2}

2. COTS software, including software tools such as compilers, linkers, automatic configuration managers, or the like, should be classified and designed as IEC 61226[3] [98] safety class A, B, or C depending on the level of software produced and whether there exists a diverse alternative or another software tool that verifies the output. The

[3] The original text of NUREG/CR-6421 [68] references IEC 1226, which has been withdrawn.

criteria used to determine the safety category classification of COTS software are as follows: {NUREG/CR-6421 [68], Tables 2 and 3}

a. Criterion 1 states that if the COTS produce is used directly in a system important to safety, the COTS safety category is determined by the criteria of IEC 61226[3] [98].

b. Criterion 2 states that if COTS that directly produces or controls the configuration of an executable software product is used in a system important to safety, and if no method exists to validate the output of the COTS product, then the COTS safety category is the same as that of its output, except that Category C software may be produced by COTS products of the unclassified category. COTS software that directly produces Category A or B software that is validated by other means is Category B or C, respectively.

c. Criterion 3 states that COTS that supports production of Category A, B, or C software, but does not directly produce or control the configuration of such software modules, falls under the safety category "unclassified."

d. Criterion 4 states that if the COTS product has no impact on category A, B, or C software or systems, it falls under the safety category "unclassified".

3. Tools used for software development may reduce or eliminate the ability of the vendor to evaluate the output of those tools. Therefore, subsequent testing should be performed to show the software will perform as intended. {NUREG 0800, Appendix 7.1-D, Section 5.3.2}

4. Configuration items or controlled documents should include support software used in development, test code used in testing, and commercial software that are safety system software. {RG 1.169 [44], Section C.6}

5. Contractually developed or qualified commercial software products that are safety system software and tools used in the development of safety system software must be taken under control by an SCM program. {RG 1.169 [44], Section C.7}

6. Tools should be used to support all aspects of the I&C life cycle where benefits result through their use and where tools are available. {MDEP DICWG-02 [3], Position 1}

7. The benefits and risk of using a tool should be balanced against the benefits and risk of not using a tool (i.e., choose tools that limit the opportunity for making errors and introducing faults, but maximize the opportunity for detecting faults). {MDEP DICWG-02 [3], Position 2}

8. The functionality and limits of applicability of all tools should be identified and documented. {MDEP DICWG-02 [3], Position 3}

9. Tools and their output should not be used outside their declared functionality or limits of application without prior justification. {MDEP DICWG-02 [3], Position 4}

10. The qualification process of a tool should take into account experience from prior use. {MDEP DICWG-02 [3], Position 6}

11. Tool parameters used during the development, verification, or validation of software should be recorded in the development records. {MDEP DICWG-02 [3], Position 8}

12. Qualification of a tool is only required if one of the system or software life cycle processes is eliminated, reduced, or automated and the tool has not been verified. {FAA Order 8110.49 [21] and FAA Order 8110.49 Change 1 [22], Chapter 9-1}

13. Tools are allowed to make changes to the modifiable component of User Modifiable Software (UMS) provided the following information is provided to the certification authority for approval: {FAA Order 8110.49 [21] and FAA Order 8110.49 Change 1 [22], Chapter 7-6.a}

    a. Plans for controlling tool version;

    b. Plans for controlling tool usage;

    c. Plans for qualifying or verifying the tool; and

    d. Procedures for modifying the tool.

14. Methods and tools should be chosen that provide error prevention during the software development process. {RTCA/DO-178B [20] and RTCA/DO-178C [28], Section 4.2.b.}

15. Software tools are defined by the software planning process and identified in the software life cycle environment configuration index. The following guidance should be used to assign a control category to each tool. Based on the control category, the software configuration management process objectives specified in Table 5.1 should be completed for that tool. {RTCA/DO-178C [28], Sections 7.3 and 7.5}

    a. Configuration identification should be established for the executable object code (or equivalent) of the tools used to develop, control, build, verify, and load the software.

    b. The SCM process for controlling qualified tools should comply with the objectives associated with control category 1 (CC1) for software development tools and control category 2 (CC2) for software verification tools.

    c. Unless paragraph (b) above applies, the SCM process for controlling the executable object code (or equivalent) of tools used to build and load the software (compilers, assemblers, linkage editors) should comply with the SCM process objectives associated with CC2, as a minimum.

Table 5.1 – SCM Process Objective Control Categories

SCM Process Objective	CC1	CC2
Configuration Identification	●	●
Baselines	●	
Traceability	●	●
Problem Reporting	●	
Change Control – integrity and identification	●	●
Change Control – tracking	●	
Change Review	●	
Configuration Status Accounting	●	
Retrieval	●	●
Protection against Unauthorized Changes	●	●
Media Selection, Refreshing, Duplication	●	
Release	●	
Data Retention	●	●

16. If a life cycle process is eliminated, reduced, or automated by the use of software development tools and the tool output has not been verified, then the software development tools need to be treated at a level equivalent to the safety-related plant software and that the tool meets its tool-specific operational requirements. A trial period may be needed during which a verification of tool output is performed and tool-related problems are analyzed, recorded, and corrected to demonstrate that the tool meets its operational requirements. {RTCA/DO-178B [20], Section 12.2.1.c}

17. The tool qualification plan should explain the methods that will be used to qualify the tool, including those objectives and guidance that will be met using a service history approach. {RTCA/DO-248B [27], Section 12.3.4}

18. If deficiencies exist in the software life cycle data of COTS software tools, the data should be augmented to satisfy certification requirements. {RTCA/DO-178C [28], Section 2.5.3}

19. If the tool eliminates, reduces, or automates software life cycle processes and its output is not verified, tool qualification is needed and the following procedure should be used to determine the tool qualification level (TQL) and to qualify the tool. {RTCA/DO-178C [28], Sections 2.3.2, 2.3.3, 12.2.2, and 12.2.3}

    a. Each tool should be assigned a criteria as determined below based on the impact of the tool use on the software life cycle processes.

        Criteria 1: A tool whose output is part of the airborne software and thus could insert an error.

        Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:

1. Verification process(es) other than that automated by the tool, or

2. Development process(es) that could have an impact on the airborne software.

Criteria 3: A tool, which within the scope of its intended use, could fail to detect an error.

b. Each tool should be assigned a software level as determined below consistent with the software component associated with each tool based upon the failure condition which may result from anomalous behavior of the software as determined by the system safety assessment.

Level A: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft which would result in multiple fatalities, usually with the loss of the airplane.

Level B: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a hazardous failure condition for the aircraft which would reduce the capability of the airplane or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be:

1. A large reduction in safety margins or functional capabilities

2. Physical distress or excessive workload such that the flight crew cannot be relied upon to perform their tasks accurately or completely or

3. Serious of fatal injury to a relatively small number of the occupants other than the flight crew.

Level C: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to the flight crew, or physical distress to passengers or cabin crew, possibly including injuries.

Level D: Software whose anomalous behavior would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft which would not significantly reduce airplane safety, and which involve crew actions that are well within their capabilities including, for

example, a slight reduction in safety margins or functional capabilities, a slight increase in crew workload, such as routine flight plan changes, or some physical discomfort to passengers or cabin crew.

Level E: Software whose anomalous behavior would cause or contribute to a failure of system function with no effect on aircraft operational capability, safety or crew workload. If a software component is determined to be Level E and this is confirmed by the certification authority, no tool qualification is required.

c. Each tool should be assigned an appropriate TQL as determined from Table 5.2 based on the assigned criteria from step a and the software level from step b. The tool should then be qualified following the guidance of RTCA/DO-330 [31].

Table 5.2 – Tool Qualification Level Determination

Software Level	Criteria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5
E	No tool qualification necessary		

20. Methods, techniques, and tools that produce results that cannot be verified to an acceptable degree or that are not compatible with safety requirements should be prohibited, unless analysis shows that the alternative would be less safe. {NUREG-0800 BTP 7-14 [64], Section B.3.1.2.3}

21. Engineering judgment must be based on quantifiable critical characteristics and adequate evidence. The documentation of the judgment should be sufficient to allow a comparably qualified individual to come to the same conclusion. For software tools these documents should include any and all documentation from the original manufacturer, results of any testing performed by the tool user (software developer), and the results of any audits, surveys, or monitoring that may have been performed at a minimum. {EPRI TR 106439 [106], Page 4-19}

22. In those cases where the quality assurance requirements of an original tool cannot be determined, an engineering evaluation shall be conducted by qualified individuals to establish the requirements and controls. This evaluation shall assure that interfaces, interchangeability, safety, fit, and function are not adversely affected or contrary to applicable regulatory or code requirements. {EPRI TR 102260 [105], Section 1.3.3.2}

23. Reasonable assurance of tool performance must be based on facts, actions, or observations. However, the decision that reasonable assurance has been attained is

inherently subjective and the judgment of reasonability may vary between different observers. {EPRI TR 102260 [105], Section 2.1.2.2.3}

24. The primary difference between a critical characteristic for design and a critical characteristic for acceptance is that the critical characteristics for acceptance need only be based on the complexity, intended safety-related function, and performance of the tool. {EPRI TR 102260 [105], Section 2.3.2.4}

25. The information needed to reach a determination that the software tools are adequate for their intended use should be contained in the documentation of the software tool verification program. The intended use of those tools should be described in the software development plan, the software integration plan, and software test plan, depending on how the tool will be used. Use of the tools may be audited to assess the quality of the tools. {DI&C-ISG-06 [63], Section D.10.4.2.3.2}

5.2.3 Technical Processes

1. System requirements should be developed using a controlled procedure or methodology. Based on the system requirements, a set of critical requirements can then be generated for the commercial-grade dedication process or evaluation of the software tools used. {EPRI TR 107339 [107], Section 5.1}

2. The use of software tools should be evaluated in the overall context of the quality control and V&V process, and there should be a method of evaluating the output of the tool. {NUREG-0800 Appendix 7.1-D [65], Section 5.3.2}

3. V&V tasks of witnessing, reviewing, and testing are not required for software tools, provided the software that is produced using these tools is subject to V&V activities that will detect flaws introduced by the tools. {RG 1.168 Revision 1 [38], RG 1.168 Revision 2 [41], Section C.2}

4. The selection, qualification, and use of software tools for the development of software for safety systems should follow the guidelines of Section 14 of IEC 60880. {MDEP DICWG-02 [3], Position 9}

5. The introduction of unnecessary or extraneous requirements by tools that may result in inclusion of unwanted or unnecessary code should be prevented in the requirements phase. {RG 1.152 Revision 3 [34], Section 2.2.2}

6. Measures should be taken to prevent the introduction of unnecessary design features by tools that may result in unwanted or unnecessary code in the design phase. {RG 1.152 Revision 3 [34], Section 2.3.2}

7. COTS systems and tools are likely to be proprietary and generally unavailable for review. It is likely that there is no reliable method to determine security vulnerabilities for operating systems and tools. In such cases, unless such systems or tools are modified by the application developer, the security effort should be limited to ensuring that the

features within the operating system or tool do not compromise the secure operational environment that would degrade the reliability of the safety system. {RG 1.152 Revision 3 [34], Section C.2.4.2}

8. If requirements are generated using specification or representation tools, traceability should be maintained between these representations and the natural language descriptions of the software requirements that are derived from systems requirements and system safety analyses. {RG 1.172 [56] and RG 1.172 Revision 1 [58], Section C.2.a}

9. Requirements for tools should meet the following characteristics: (RTCA/DO-330 [31], Section 5.2.1.2}

    a. Should be developed and documented in a manner that provides a determination if the tool satisfies the tool operational requirements;

    b. Should include all functional and interface-related requirements that will be implemented in the tool;

    c. Should be developed following the processes described in the tool development plan;

    d. Should be developed using the tool requirements standards;

    e. Should be verifiable and consistent;

    f. Should be defined such that abnormal behavior is detected and that invalid output is prevented;

    g. Each requirement should trace to one or more tool operational requirements, with the exception of derived tool requirements;

    h. Derived tool requirements are those not traceable to tool operational requirements. The existence of derived tool requirements should be justified, and they should be evaluated to ensure that they do not negatively impact the expected functionality and outputs defined in the tool operational requirements;

    i. Should provide user instructions, list of error messages, and constraints;

    j. Should identify requirements related to unused tool functions that do not trace to tool operational requirements; and

    k. Should be defined to a level of detail appropriate to ensure proper implementation and to assess correctness of the tool.

10. The tool design process should define the tool architecture. (RTCA/DO-330 [31], Section 5.2.2.2.a}

11. The tool design should address all tool architecture features, such as protection, if applicable. Protection may be used to isolate parts of the tool or the collection of tools to apply a different approach or different level of qualification. {RTCA/DO-330 [31], Section 5.2.2.2.b}

12. The tool requirements should be refined into low-level tool requirements that are traceable to tool requirements or identified as derived low-level requirements. {RTCA/DO-330 [31], Section 5.2.2.2.c}

13. The tool design description, including tool architecture and low-level tool requirements, should conform to the tool design standards. {RTCA/DO-330 [31], Section 5.2.2.2.e}

5.2.4 Software Implementation Processes

1. Integrated development environments (IDE), editors, compilers, and linkers should be easy to learn, well integrated or easy to integrate with other tools, default to enforcing standards, and well documented. {NASA-GB-8719.13-1 [12], Section 11.2}

2. When automatically generated code is safety critical or resides within an unprotected partition with safety-critical code, the automatically generated code should be subject to the same inspection, analysis, and testing as hand-generated code. {NASA-GB-8719.13-1 [12], Section 11.3}

3. If the compiler used to generate executable code uses optimization, the correctness of the optimization need not be verified if the test cases give coverage consistent with the software level. Otherwise, the impact of optimization should be determined. {RTCA/DO-178B [20] and RTCA/DO-178C [28], Section 4.4.2.a}

4. Compilers may also produce object code that is not directly traceable to source code, for example, initialization, built-in error detection, or exception handling. The software planning process should provide a means to detect this object code and to ensure verification coverage. {RTCA/DO-178B [20] and RTCA/DO-178C [28], Section 4.4.2.b}

5. Language and compiler considerations: {RTCA/DO-178C [28], Clause 4.4.2}

    a. Some compilers have features intended to optimize performance of the object code. If the test cases give coverage consistent with the software level, the correctness of the optimization need not be verified. Otherwise, the impact of these features on structural coverage analysis should be determined.

    b. The implementation of certain compiler features may produce object code that is not directly traceable to the source code, for example, initialization, built-in error detection, or exception handling. The software planning process should provide a means to detect this object code and to ensure verification coverage, and should define the means in the appropriate plan.

    c. If a new compiler, linkage editor, or loader version is introduced, or compiler options are changed during the software life cycle, previous tests and coverage

analyses may no longer be valid. The verification planning should provide a means of re-verification.

6. When user-modifiable software is included in the project, the non-modifiable component should be protected from the modifiable component to prevent interference in the safe operation of the non-modifiable component. This protection can be enforced by hardware, by software, by the tools used to make the change, or by a combination of the three. If the protection is provided by a tool, the tool should be qualified. {RTCA/DO-178C [28], Section 5.2.3}

7. Changes to an application or development environment should be identified, analyzed, and re-verified. If a new development environment uses software tools, the tool may need to be qualified. {RTCA/DO-178C [28], Section 12.1.3.a}

8. Using a different autocode generator or a different set of autocode generator options may change the source code or object code generated. Therefore, the impact of these changes should be analyzed. {RTCA/DO-178C [28], Section 12.1.3.c}

9. If a different compiler or options are used, previous verification may not be valid and should not be used for the new application. {RTCA/DO-178C [28], Section 12.1.3.d}

10. If qualified tools are used to develop or verify a Reusable Software Component (RSC), the RSC developer must consider reuse of those tools during RSC development and acceptance. {FAA AC 20-148 [25], Section 11.f}

    a. For development tools, the tool qualification plan and the tool accomplishment summary must document the portions of the tool qualification that the applicant is required to complete.

    b. For verification tools, the plan for software aspects of certification (PSAC) and the software accomplishment summary (SAS) must document the portions of the tool qualification that the applicant is required to complete.

    c. The RSC developer must provide tool plans, tool operational requirements, and the tool accomplishment summary to the applicant for all tools used in getting acceptance of the RSC.

11. When automatic code generation (ACG) tools are used in the software development process to eliminate, reduce, or automate life cycle processes, the following guidelines should be used when applicants are qualifying ACG tools. {Position Paper CAST-13 [24], Section 3}

    a. The software level of the tool needs to be determined. Where there is no justification to reduce the software level of the tool, the tool should be qualified to the same software level and objectives as that of the safety-related plant software.

b. If the tool is designed and developed in such a way that the ACG tool's architecture and logic can produce unintended, non-deterministic, or erroneous code, then a software level reduction is not possible.

c. A software level reduction is possible if it can be demonstrated that potential errors in the ACG tool's output can be detected by additional verification on the code produced or if it can be demonstrated that potential errors and failure conditions in the software as a result of the ACG tool's software level reduction or objective's alleviation cannot contribute to the failures of the software.

d. The applicant can propose a reduction in the ACG software level based on a safety assessment, software architecture, tool partitioning, additional verification, and/or by other means.

e. If justification of a lower software level exists, the verification objectives and activities for the reduced software level should be satisfied. Justification of a lower software level should address the consequences of a tool's software level reduction on the safety-related plant software.

f. Some life cycle process objectives can be alleviated or removed if it can be demonstrated that incompatibility problems with the ACG tool are obvious and readily detected and the applicants supply relevant rationale and justification for each objective's alleviation.

12. Special care should be taken when reviewing software tool usage, especially compilers and linkers, to ensure that the single-failure criterion is not violated. For example, diverse systems that have been compiled with a single compiler may subject the code output to a common-mode failure mechanism. {NUREG-0800 Appendix 7.1-D [65], Section 5.1}

13. Different tools should be used on redundant systems to reduce the possibility of common-mode failure. {NUREG/CR-6263 [67], Section 14.2.3}

14. If a single software tool is used during several portions of the software development life cycle, care must be taken to ensure that a fault in the tool does not introduce a source of a common-cause failure in the software. {IEEE Std. 603-1991 [37] and IEEE Std. 603-2009 [71]}

15. The tool source code should implement the low-level tool requirements and conform to the tool architecture. {RTCA DO-330 [31], Section 5.2.3.2.a}

16. The tool source code should conform to the tool code standards. {RTCA DO-330 [31], Section 5.2.3.2.b}

17. The tool source code should be traceable to the low-level tool requirements. {RTCA DO-330 [31], Section 5.2.3.2.c}

18. Inadequate or incorrect inputs detected during the tool coding process should be provided to the process(es) that produced the incorrect input(s) for clarification or correction. {RTCA DO-330 [31], Section 5.2.3.2.d}

19. The tool executable object code should be generated from the tool source code and the compiler and linking instructions. It should be noted that the tool executable object code sometimes includes other files, such as configuration files. {RTCA DO-330 [31], Section 5.2.4.2.a}

20. Inadequate or incorrect inputs detected during the tool integration process should be provided to the process(es) that produced the inadequate or incorrect input(s). {RTCA DO-330 [31], Section 5.2.4.2.b}

21. If the tool verification environment is different than the tool development environment, or if there are multiple tool verification environments, then the tool executable object code should be installed in the tool verification environment(s). {RTCA DO-330 [31], Section 5.2.4.2.c}

5.2.5 Software Support Processes

1. Reviewers should thoroughly evaluate tool usage. Tools used for software development may reduce or eliminate the ability for the vendor to evaluate the output of those tools, and therefore, rely on the tool, or on subsequent testing to show the software will perform as intended. Testing alone can only show that those items tested will operate as intended, and cannot be relied upon to show that no unintended functions exist, or that the software will function in conditions other that those specifically tested. The use of software tools should be evaluated in the overall context of the quality control and V&V process, and there should be a method of evaluating the output of the tool. {NUREG-0800 Appendix 7.1-D [65], Section 5.3.2}

2. A comprehensive test suite is an effective approach to increase objectivity and uniformity in the application of the qualification criteria to structural coverage analysis tools. {DOT/FAA/AR-06/54 [23], Section 2}

3. Software verification tools can be qualified by just confirming that the tool meets its tool-specific operational requirements under normal operational conditions. {RTCA/DO-178B [20], Section 12.2.2}

4. The functional requirements of a verification tool should accurately reflect the corresponding life cycle process objective that it is replacing or simplifying. When selecting a tool, the first process should be how well the required life cycle process objectives can be satisfied by the tool. The objectives must be readily understood. If they are not, it is not possible to correctly determine that the tool has implemented the objectives correctly. {RTCA/DO-248B [27], Section 4.1.2.1}

5. The interpretation of a tool's output is based upon the user's understanding of the tool's function. If an incorrect assumption has been made about the tool's function, the

interpretation of its output may be equally flawed. When selecting a tool it is essential to understand its limitations so that additional manual efforts may augment the tool to complete the verification process. {RTCA/DO-248B [27], Sections 4.1.2.3.1 and 4.1.2.3.2}

6. Tools used in the development of safety system software should be handled according to IEEE Std. 7-4.3.2-2003 [35] as endorsed by RG 1.152, Revision 3 [34]. In particular, a software configuration management (SCM) program operated by the using organization that complies with IEEE Std. 828-2005 [48] should take control of tools (i.e., tools should be treated as configuration items). SCM plans should identify that software tools used in the production of safety system software be considered as a configuration item. {RG 1.169, Revision 1 [47], Section C.8}

7. Repetition is sometimes needed when automated tools are used for testing. Test information must be made easily accessible to improve the timeliness of a safety conclusion. {RG 1.170 Revision 1 [51], Section C.8}

8. Documentation used to support software testing must include as a minimum environmental conditions and special controls, equipment, tools, and instrumentation needed for the accomplishment of testing. {RG 1.170 [49] and RG 1.170 Revision 1 [51], Section C.1.c}

9. Verification and validation activities common to software development in digital systems are critical characteristics that can be verified as being performed correctly following the completion of the software development by conducting certain dedication activities such as audits, examinations, and tests. {EPRI TR 1025243 [108], Page viii}

10. Tools should be verified and assessed consistent with the tool reliability requirements, the type of tool, and the potential of the tool to introduce faults. {MDEP DICWG-02 [3], Position 5}

11. Verification is the technical assessment of the outputs of both the tool development processes and the tool verification process. The tool verification process is applied as defined by the tool planning process and the tool verification plan. Verification includes reviews, analyses, and tests. {RTCA DO-330 [31], Section 6.1}

12. In order to determine that the higher level of requirements is satisfied by the lower levels of requirements (or source code) and to identify derived requirements, traceability should exist as follows: {RTCA DO-330 [31], Section 6.1.2.a}

    a. Tool requirements should be traceable to the tool operational requirements.

    b. The low-level tool requirements should be traceable to the tool requirements.

    c. The tool source code should be traceable to the low-level tool requirements.

13. In order to identify the requirements not covered by test cases, the test cases should be traceable to the requirements (tool requirements and low-level tool requirements). {RTCA DO-330 [31], Section 6.1.2.b}

14. When it is not possible to verify specific tool requirements, other means should be provided and their justification for satisfying the tool verification process objectives defined in the tool verification plan and/or tool verification results. {RTCA DO-330 [31], Section 6.1.2.d}

15. Deficiencies and errors discovered during the tool verification process should be reported to the appropriate tool development processes for clarification and correction. {RTCA DO-330 [31], Section 6.1.2.e}

16. Tool configuration management process objectives are: {RTCA DO-330 [31], Section 7.1}

    a. Each configuration item and its successive versions are labeled unambiguously so that a basis is established for the control and reference of configuration items;

    b. Baselines are defined for further tool life cycle process activity and allow reference to, control of, and traceability between configuration items;

    c. The problem reporting process records non-compliance with tool plans and standards, records deficiencies of outputs of tool life cycle processes, records anomalous behavior of tool products, and ensures resolution of these problems;

    d. Change control provides for recording, evaluation, resolution, and approval of changes throughout the life cycle;

    e. Change review ensures problems and changes are assessed, problems and changes are approved or disapproved, approved changes are implemented, and feedback is provided to affected processes through problem reporting and change control methods defined during the tool planning process;

    f. Status accounting provides data for the configuration management of tool life cycle processes with respect to configuration identification, baselines, problem reports, and change control;

    g. Archival and retrieval ensures that the tool life cycle data associated with the tool product can be retrieved in case of a need to duplicate, regenerate, retest, or modify the tool product. The objective of the release activity is to ensure that only the authorized tool is used in the software life cycle processes, in addition to being archived and retrievable; and

    h. Tool life cycle environment control ensures that other tools used to produce the tool itself are identified, controlled, and retrievable.

5.2.6 Software Reuse Processes

1. When a project includes reusable verification or development tools, the following general guidelines apply for reusable tool qualification. {Position Paper CAST-22 [26], Section 3.0}

    a. When reusable software verification and development tools are initially planned, the boundaries and limitations of what is reusable and what will be project specific should be established during the first acceptance of the reusable tool qualification package.

    b. The tools should be packaged and qualified in such a way that some of the tool qualification data may be reused on other projects.

    c. The initial reusable tool qualification must be done in the context of an actual certification project and must have the documented agreement of all the stakeholders.

    d. For each objective that the tool automates, replaces, or supplements, the amount of credit being sought (full, partial, or no credit), assumptions, means of compliance, tool limitations, and remaining activities to be completed by the tool user, system developer, and/or applicant must be described in the reusable tool documents.

2. Any assumptions and limitations that affect proper reusable tool operation should also be identified in the project plan (as well as in the tool user's guide). {Position Paper CAST-22 [26], Section 4.5}

3. COTS and pre-developed tools can be qualified under the commercial dedication processes of EPRI-TR-106439 [106]. {DI&C-ISG-06 [63], Section D.10.4.2.3.2}

6 Summary and Conclusions

6.1 Contract Summary

NRC-HQ-13-C-04-0004 is a research project phased in three tasks. In Task 1, Information Systems Laboratories, Inc. (ISL), the contractor, has surveyed standards, regulatory guidance, review/approval practices, and other current industry practices concerning the use of software tools on which the safety assurability of a digital, safety-related system is dependent. The product of Task 1 is this report summarizing the results of the survey including a broad baseline of requirements covered by all the industry standards and a broad baseline of guidance and review and approval practices covered by all the regulatory guidance and industry review/approval practices.

6.2 Scope of Document Survey

ISL has surveyed and compared 92 documents from the space industry, the civil aviation industry, the Nuclear Regulatory Commission (NRC) and commercial nuclear power industry, the Institute of Electrical and Electronics Engineers (IEEE), the International Electrotechnical Commission (IEC), the International Atomic Energy Agency (IAEA), the Electric Power Research Institute (EPRI), the National Institute of Standards and Technology (NIST), and Atomic Energy of Canada, Ltd. The surveys focused on standards, guidance, and review/approval practices related to the use of software tools, including commercial-off-the-shelf (COTS) software tools, government-off-the-shelf (GOTS) software tools, and modified-off-the-shelf (MOTS) software tools.

Section 1 of this report presents a series of tables which provide the name, title, version, and a brief abstract of all 92 documents considered for survey. Section 1 does not address the relevance of the 92 documents to software tools. The document abstracts simply provide the reader with the general content of the documents surveyed. The relevance of the document is discussed in Section 4. The tables in Section 1 are subdivided into standards, which will form the basis of the baseline requirements, and documents related to guidance and review/approval practices, which will form the basis of the baseline guidance and review/approval practices.

6.3 Software Life Cycle Summary

ISL has also surveyed and compared numerous documents related to system and/or software life cycles to compare processes, activities, and tasks associated with the development of engineering systems. This survey was performed as a preliminary step in the Task 2 assessment of the vulnerabilities associated with the use of software tools in different stages of the life cycle. The modern approach to software engineering including application of software tools revolves around the implementation of suitable software life cycle processes. As such, any regulatory guidance must consider software life cycle processes. This report does not advocate any particular life cycle model or set of activities or processes. Rather, the report addresses each phase of life cycle processes that are typically used in software engineering to determine whether software tool regulatory guidance is warranted or not warranted in certain software life cycle phases. Regulatory guidance is only warranted when the tools have an impact on the final plant safety software.

Section 2 of this report discusses three principal IEEE standards that apply to software life cycles. The first, endorsed by proposed RG 1.173, Revision 1 [6], is IEEE Std. 1074-2006 [4]. IEEE Std. 1074-2006 [4] is a standard that defines the process by which a software project life cycle process (SPLCP) is developed. This methodology begins with a selection of a software project life cycle model (SPLCM) based on the organization's mission, vision, goals, and resources. It continues through the definition of the software project life cycle (SPLC) using the selected model, application of the activities in Annex A that are mapped within the selected SPLCM, and selection of the portion of the software life cycle that is relevant to the project. The methodology concludes with the augmentation of the software life cycle with organizational process assets (OPAs) to create the SPLCP.

The second document that applies to software life cycles is IEEE Std. 15288-2008 [5]. This international standard establishes a common process framework for describing the life cycle of man-made systems. It defines a set of processes and associated terminology for the full life cycle, including conception, development, production, utilization, support and retirement. This standard also supports the definition, control, assessment, and improvement of these processes. These processes can be applied concurrently, iteratively, and recursively to a system and its elements throughout the life cycle of a system

The third document that applies to software life cycles is IEEE Std. 12207-2008 [2]. This document was written concurrently with IEEE Std. 15288-2008 [5]. IEEE Std. 12207-2008 [2] presents a set of system life cycle processes that provides a framework for describing the life cycle of software engineering. The main purpose of this standard is to provide a defined set of processes to facilitate communication among acquirers, suppliers and other stakeholders in the life cycle of the system. This standard has a strong relationship with IEEE Std. 15288-2008 [5] and may be used in conjunction with it. The software life cycle steps presented in IEEE Std. 12207-2008 [2] are subdivided into system-context and software-specific process categories as summarized in Table 6.1. The system life cycle processes of IEEE Std. 12207-2008 [2] mirror the system life cycle processes of IEEE Std. 15288-2008 [5]. The software-specific processes of IEEE Std. 12207-2008 [2] are the result of a software-specific implementation of the system-context life cycle processes.

This report does not endorse any particular life cycle model or any particular set of life cycle processes. However, for purposes of this report, which is to determine which type of software tools are used in different stages of the life cycle, the system and software life cycle processes of IEEE Std. 12207-2008 [2], as summarized in Table 6.1, have been used for describing the life cycle of software engineering.

Table 6.1 – Life Cycle Process Regulatory Guidance

Context	Process	Subprocess	Regulatory Guidance
System-Context Processes	Agreement Processes	Acquisition Process	
		Supply Process	
	Organizational Project-Enabling Processes	Life Cycle Model Management Process	
		Infrastructure Management Process	
		Project Portfolio Management Process	
		Human Resource Management Processes	
		Quality Management Process	●
	Project Management Processes	Project Planning Process	
		Project Assessment and Control Process	
	Project Support Processes	Decision Management Process	
		Risk Management Process	
		Configuration Management Process	●
		Information Management Process	
		Measurement Process	●
	System Technical Processes	**Stakeholder Requirements Definition Process**	●
		System Requirements Analysis Process	●
		System Architectural Design Process	●
		Implementation Process	●
		System Integration Process	●
		System Qualification Testing Process	●
	Software Technical Processes	**Software Installation Process**	●
		Software Acceptance Support Process	●
		Software Operation Process	●
		Software Maintenance Process	●
		Software Disposal Process	
Software-specific Processes	Software Implementation Processes	**Software Implementation Process**	●
		Software Requirements Analysis Process	●
		Software Architectural Design Process	●
		Software Detailed Design Process	●
		Software Construction Process	●
		Software Integration Process	●
		Software Qualification Testing Process	●
	Software Support Processes	Software Documentation Management Process	
		Software Configuration Management Process	●
		Software Quality Assurance Process	●
		Software Verification Process	●
		Software Validation Process	●
		Software Review Process	●
		Software Audit Process	●
		Software Problem Resolution Process	●
	Software Reuse Processes	**Domain Engineering Process**	●
		Reuse Asset Management Process	●
		Reuse Program Management Process	●

Based on the activities, tasks, and products associated with the system-context and software-specific processes of IEEE Std. 12207-2008 [2], and whether a tool used during a specific process impacts the final nuclear plant software, ISL believes that some of the system-context processes and most of the software-specific process should be subject to regulatory guidance. The processes in which ISL considers that there should be regulatory guidance are highlighted and marked with a bullet in Table 6.1.

6.4 Software Tool Summary

ISL has researched industry standards, regulatory guidance, and review/approval practices to determine the types of software tools used in the nuclear and other industries and the general philosophy for software tool use during software development. The industry consensus as determined by ISL from this survey and as expressed by the Multinational Design Evaluation Program (MDEP) Digital I&C Working Group (DICWG) is that the use of appropriate software tools can increase the integrity of the instrumentation and control (I&C) development process and the software reliability by reducing the risk of introducing faults during the process. The use of tools can also increase productivity and reduce costs associated with software development by reducing the time and human effort required to produce systems, components, and software. Tools are primarily used to capture system and software requirements; transform requirements into final system code and data; perform verification, validation, and testing; prepare and control application data; and manage and control processes and products involved in the software development. Section 3 of this report provides a generic list of software tools based on computer-aided software engineering (CASE) tool characteristics discussed in IEEE Std. 14102-2010 [83] and an ISL generated list of software tool categories based upon tools that are widely used within the software development industry. As discussed in Sections 3.1 through 3.9, based on similarities between certain software tools and software tool usage and availability in certain life cycle processes, ISL believes that all software tools can be grouped into nine software tool categories as summarized in Table 6.2. As discussed in Sections 3.1 through 3.9, based on the impact that the tool has on the final safety-related software, ISL believes that some tools, as indicated by a bullet in Table 6.2, should have regulatory guidance.

Table 6.2 – Software Tools Category Regulatory Guidance

Tool Category	Regulatory Guidance
Management Support Tools	
Implementation Tools	•
Modeling Tools	•
Construction Tools	•
Maintenance Tools	•
Documentation Tools	
Configuration Management Tools	•
Quality Assurance Tools	•
Verification and Validation Tools	•

6.5 Document Survey Summary

The survey of industry standards, regulatory guidance, and review/approval practices of software tools has found that there is little information pertaining to software tools in the nuclear field internationally. While some guidance does exist in other industries, software tools are still not widely addressed or regulated except for the civil aviation industry, which contains several documents providing guidance and review/approval practice for software tool use, and IEEE standards that discuss the wide abundance of CASE tools. Based on the extensive survey of major industries, ISL believes that the IEEE standards related to the selection, use, evaluation and qualification of CASE tools along with the Federal Aviation Administration (FAA) standards, FAA orders, FAA advisory circulars, Certification Authorities Software Team (CAST) papers, and Radio Technical Commission for Aeronautics (RTCA) documents may provide a basis upon which to build software tool (including COTS, GOTS, and MOTS software tools) regulation, guidance, and review/approval practices for the commercial nuclear industry pending further analysis.

6.6 Baseline Summary

During the survey of various international standards, regulatory guidance documents, and documents describing “best practices,” several basic guidelines were stated in similar forms. These guidelines cover a majority of the software tools that are of concern for this study. Additional guidelines, that were limited to one or two documents, were not included in our baseline either because they did not apply to software tools used in the development of software important to safety in a nuclear environment or because the technical basis did not meet the standards of the survey team.

The purpose of the tool qualification process is to obtain confidence in the tool functionality. The tool qualification effort varies based upon the potential impact that a tool error could have on the system safety and upon the overall use of the tool in the software life cycle process. The higher the risk of a tool error adversely affecting system safety, the higher the rigor required for tool qualification {RTCA DO-330 [31], Section 2.0}.

The determination of which tools should be subject to regulatory guidance is based on several factors. These include:

1. whether the tools have a direct effect on the final software output and result in the creation of software errors (e.g., autocode generators);
2. whether the tools have an indirect effect on the final software output and result in the creation of software errors (e.g., failure to resolve quality assurance (QA) audit and review anomalies);
3. whether they are used in the V&V process and may fail to detect errors;
4. or
    a. if the use of the tool can be used to justify bypassing other verification and validation processes which could result in failure to detect errors or

b. if they can be used to justify elimination of software constructs that would have been used in fault handling which could result in failure to react properly to errors.

Based on the above guidelines for determining which tools should be subject to regulatory guidance, ISL has produced a broad baseline of requirements covering all the industry standards for the use of software tools (including COTS, GOTS, and MOTS software tools) and a broad baseline of guidance and review/approval practice covering all the regulatory guidance. The baseline requirements for software tools to support regulatory assurance engender the need for proper evaluation and qualification, adequate configuration management, thorough V&V, proper use as described by the tool's intended purpose, correct software tool classification level, thorough up-front planning, continuous QA, and adequate training on tool use. The baseline guidance and review/approval practices for software tools to support regulatory assurance include evaluation and qualification guidance, configuration management practices, V&V practices, guidance and limitations on tool use, tool classification level guidance, planning practices, QA guidance, and training practices.

7 References

[1] *IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations,* IEEE Std. 7-4.3.2-2010, June 2010

[2] *Systems and Software Engineering - Software Life Cycle Processes,* IEEE Std. 12207-2008, February 2008

[3] *MDEP Common Position on Software Tools for the Development of Software for Safety Systems*, MDEP Generic Common Position DICWG No 2, December 2010

[4] *IEEE Standard for Developing a Software Life Cycle Process,* IEEE Std. 1074-2006, July 2006

[5] *Systems and Software Engineering - System Life Cycle Processes,* IEEE Std. 15288-2008, February 2008

[6] *Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.173, Revision 1, DRAFT

[7] *Software Assurance Standard,* NASA-STD-2201-93, November 1992

[8] *Software Assurance Standard,* NASA-STD-8739.8, July 2004

[9] *Software Safety Standard,* NASA-STD-8719.13B, July 2004

[10] *Software Assurance Guidebook,* NASA-GB-A201, September 1989

[11] *Software Quality Assurance Audits Guidebook,* NASA-GB-A301, November 1990

[12] *NASA Software Safety Guidebook,* NASA-GB-8719.13-1, March 2004

[13] *Software Measurement Guidebook,* NASA-GB-001-94, August 1995

[14] *Software Process Improvement Guidebook,* NASA-GB-001-95, January 1996

[15] *Software Management Guidebook,* NASA-GB-001-96, November 1996

[16] *Software Development for the National Airspace System (NAS),* FAA-STD-026A, June 2001

[17] *Software Life Cycle Processes*, IEEE/EIA 12207.0, 1996

[18] *Software Life Cycle Processes – Life Cycle Data*, IEEE/EIA 12207.1, 1997

[19] *Software Life Cycle Processes – Implementation Considerations*, IEEE/EIA 12207.2, 1997

[20] *Software Considerations in Airborne Systems and Equipment Certification,* RTCA/DO-178B, December 1992

[21] *Software Approval Guidelines,* FAA Order 8110.49, June 2003

[22] *Software Approval Guidelines,* FAA Order 8110.49 Change 1, September 2011

[23] *Software Verification Tools Assessment Study,* DOT/FAA/AR-06/54, June 2007

[24] *Automatic Code Generation Tools Development Assurance,* Position Paper CAST-13, June 2002

[25] *Reusable Software Components,* FAA AC 20-148, December 2004

[26] *Reuse of Software Tool Qualification Data Across Company Boundaries (Applying the Reusable Software Component Concept to Tools), Revision 1,* Position Paper CAST-22, March 2005

[27] *Final Report for Considerations of DO-178B,* RTCA/DO-248B, October 2001

[28] *Software Considerations in Airborne Systems and Equipment Certification,* RTCA/DO-178C, December 2011

[29] *Supporting Information for DO-178C and DO-278A,* RTCA/DO-248C, December 2011

[30] *Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems*, RTCA/DO-278A, December 2011

[31] *Software Tool Qualification Considerations,* RTCA/DO-330, December 2011

[32] *Software Assurance Policy,* FAA Order 1370.109, October 2009

[33] *Guidance for the Validation of Software Tools Used in the Development of Instrument Flight Procedures by Third Party Service Providers,* FAA AC 90-111, June 2011

[34] *Criteria for Use of Computers in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.152, Revision 3, July 2011

[35] *IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations,* IEEE Std. 7-4.3.2-2003, September 2003

[36] *Criteria for Safety Systems,* NRC Regulatory Guide 1.153, Revision 1, June 1996

[37] *IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations,* IEEE Std. 603-1991, June 1991

[38] *Verification, Validation, Reviews, and Audits for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.168, Revision 1, February 2004

[39] *IEEE Standard for Software Verification and Validation,* IEEE Std. 1012-1998, July 1998

[40] *IEEE Standard for Software Reviews,* IEEE Std. 1028-1997, December 1997

[41] *Verification, Validation, Reviews, and Audits for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.168, Revision 2, DRAFT

[42] *IEEE Standard for Software Verification and Validation,* IEEE Std. 1012-2004, April 2005

[43] *IEEE Standard for Software Reviews,* IEEE Std. 1028-2008, August 2008

[44] *Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.169, September 1997

[45] *IEEE Standard for Software Configuration Management Plans,* IEEE Std. 828-1990, September 1990

[46] *IEEE Guide to Software Configuration Management,* IEEE Std. 1042-1987, September 1987

[47] *Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.169, Revision 1, DRAFT

[48] *IEEE Standard for Software Configuration Management Plans,* IEEE Std. 828-2005, August 2005

[49] *Software Test Documentation for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.170, September 1997

[50] *IEEE Standard for Software Test Documentation,* IEEE Std. 829-1983, August 1983

[51] *Test Documentation for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.170, Revision 1, DRAFT

[52] *IEEE Standard for Software and System Test Documentation,* IEEE Std. 829-2008, July 2008

[53] *Software Unit Testing for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.171, September 1997

[54] *IEEE Standard for Software Unit Testing,* IEEE Std. 1008-1987, December 1986

[55] *Software Unit Testing for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.171, Revision 1, DRAFT

[56] *Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.172, September 1997

[57] *IEEE Recommended Practice for Software Requirements Specifications,* IEEE Std. 830-1993, December 1993

[58] *Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.172, Revision 1, DRAFT

[59] *IEEE Recommended Practice for Software Requirements Specifications,* IEEE Std. 830-1998, June 1998

[60] *IEEE Systems and Software Engineering - Life Cycle Processes -- Requirements Engineering,* IEEE Std. 29148-2011, December 2011

[61] *Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* NRC Regulatory Guide 1.173, September 1997

[62] *IEEE Standard for Developing Software Life Cycle Processes,* IEEE Std. 1074-1995, September 1995

[63] *Digital Instrumentation and Controls Licensing Process,* DI&C-ISG-06, Revision 1, January 2011

[64] *Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems,* NUREG-0800 Branch Technical Position (BTP) 7-14, Revision 7, March 2007

[65] *Guidance for Evaluation of the Application of IEEE Std. 7-4.3.2,* NUREG-0800 Appendix 7.1-D, March 2007

[66] *Software Reliability and Safety in Nuclear Reactor Protection Systems,* NUREG/CR-6101, June 1993

[67] *High Integrity Software for Nuclear Power Plants: Candidate Guidelines, Technical Basis, and Research Needs,* NUREG/CR-6263, June 1995

[68] *A Proposed Acceptance Process for Commercial Off-the-Shelf (COTS) Software in Reactor Applications,* NUREG/CR-6421, March 1996

[69] *IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*, IEEE Std. 7-4.3.2-1993, 1993

[70] *IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations,* IEEE Std. 603-1998, July 1998

[71] *IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations,* IEEE Std. 603-2009, November 2009

[72] *IEEE Standard: Criteria for Protection Systems for Nuclear Power Generating Stations,* IEEE Std. 279-1971, June 1971

[73] *IEEE Standard for Qualifying Class 1E Equipment for Nuclear Power Generating Stations,* IEEE Std. 323-2003, January 2004

[74] *IEEE Standard for Software Configuration Management Plans,* IEEE Std. 828-1998, June 1998

[75] *IEEE Standard for Software Configuration Management Plans,* IEEE Std. 828-2012, March 2012

[76] *Systems and Software Engineering – Vocabulary*, ISO/IEC/IEEE 24765, December 2010

[77] *IEEE Standard Adoption of ISO/IEC 15939:2007 – Systems and Software Engineering – Measurement Process*, IEEE Std. 15939-2008, January 2009

[78] *IEEE Standard for Software Test Documentation,* IEEE Std. 829-1998, September 1998

[79] *IEEE Standard for System and Software Verification and Validation,* IEEE Std. 1012-2012, May 2012

[80] *IEEE Standard for Developing Software Life Cycle Processes,* IEEE Std. 1074-1997, December 1997

[81] *Standard for Information Technology Software Life Cycle Processes Software Development Acquirer-Supplier Agreement*, J-STD-016-1995, 1995

[82] *IEEE Recommended Practice for the Evaluation and Selection of CASE Tools,* IEEE Std. 1209-1992, December 1992

[83] *Information Technology - Guidelines for the Evaluation and Selection of CASE Tools,* IEEE Std. 14102-2010, June 2010

[84] *IEEE Guide for Adoption of ISO/IEC TR 14471:2007 Information Technology – Software Engineering – Guidelines for the Adoption of CASE Tools*, IEEE Std. 14471-2010, September 2010

[85] *IEEE Standard for CASE Tool Interconnections – Reference Model for Specifying System Behavior*, IEEE Std. 1175.4-2008, April 2009

[86] *Information Technology - Guideline for the Evaluation and Selection of CASE Tools,* IEEE Std. 1462-1998, March 1998

[87] *Information Technology – Guideline for the Evaluation and Selection of CASE Tools*, ISO/IEC 14102:1995, 1995

[88] *IEEE Standard for Software Life Cycle Processes - Risk Management,* IEEE Std. 1540-2001, March 2001

[89] *Standard for Software Engineering - Software Life Cycle Processes - Risk Management,* IEEE Std. 16085-2006, December 2006

[90] *Information Technology – Software Life Cycle Processes*, ISO/IEC 12207:1995, 1995

[91] *Adoption of ISO/IEC 15288:2002 Systems Engineering - System Life Cycle Processes,* IEEE Std. 15288-2004, December 2004

[92] *Systems Engineering – System Life Cycle Processes*, ISO/IEC 15288:2002, 2002

[93] *IEEE Guide for Developing System Requirements Specifications*, IEEE Std. 1233-1998, December 1998

[94] *IEEE Guide for Information Technology – System Definition – Concept of Operations (ConOps) Document,* IEEE Std. 1362-1998, December 1998

[95] *Nuclear Power Plants - Instrumentation and Control Systems Important to Safety - Software Aspects for Computer-Based Systems Performing Category A Functions,* IEC 60880 Ed. 2.0, May 2006

[96] *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, IEC 61508 Ed. 1.0, January 2005

[97] *Nuclear Power Plants - Instrumentation and Control Important for Safety - Software Aspects for Computer-Based Systems Performing Category B or C Functions,* IEC 62138 Ed. 1.0B, January 2004

[98] *Nuclear Power Plants – Instrumentation and Control Important to Safety – Classification of Instrumentation and Control Functions*, IEC 61226 Ed. 3.0b, July 2009

[99] *Software for Computers in the Safety Systems of Nuclear Power Stations*, IEC 60880 Ed. 1.0, September 1986

[100] *Software for Computers Important to Safety for Nuclear Power Plants – Part 2: Software Aspects of Defense Against Common Cause Failures, Use of Software Tools and of Pre-Developed Software,* IEC 60880-2 Ed 1.0, December 2000

[101] *Nuclear Power Plants – Instrumentation and Control for Systems Important to Safety – General Requirements for Systems*, IEC 61513 Ed. 1.0, March 2001

[102] *Nuclear Power Plants - Instrumentation and Control Important for Safety - Requirements for Coping with Common Cause Failures (CCF),* IEC 62340 Ed. 1.0B, December 2007

[103] *Nuclear Power Plants – Instrumentation and Control Systems Important to Safety – Separation*, IEC 60709 Ed 2.0, November 2004

[104] *Guideline for the Utilization of Commercial Grade Items in Nuclear Safety Related Applications (NCIG-07),* EPRI NP 5652, June 1988

[105] *Supplemental Guidance for the Application of EPRI Report NP-5652 on the Utilization of Commercial Grade Items,* EPRI TR-102260, March 1994

[106] *Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications,* EPRI TR-106439, October 1996

[107] *Evaluating Commercial Digital Equipment for High Integrity Applications,* EPRI TR-107339, December 1997

[108] *Plant Engineering: Guideline for the Acceptance of Commercial-Grade Design and Analysis Computer Programs Used in Nuclear Safety-Related Applications,* EPRI Technical Report 1025243, June 2012

[109] *Generic Requirements Specification for Qualifying a Commercially Available PLC for Safety-Related Applications in Nuclear Power Plants,* EPRI TR-107330, December 1996

[110] *Reference Information for the Software Verification and Validation Process,* NIST SP 500-234, March 1996

[111] *Software Quality Assurance: Documentation and Reviews,* NIST IR 4909, September 1992

[112] *NASA Software Engineering Requirements*, NPR 7150.2, November 2009

[113] *Software Approval Guidelines,* Notice 8110.110, January 2010

[114] *Security Considerations in the System Development Life Cycle,* NIST SP 800-64 Revision 2, October 2008

[115] *IEEE Standard for Information Technology – Systems Design – Software Design Descriptions*, IEEE Std. 1016-2009, July 2009

[116] *IEEE Standard for Software Quality Assurance Plans*, IEEE Std. 730-2002, 2002

[117] *Advanced Light Water Reactor Utility Requirements Document – Passive Plant*, NP-6780, Volume III, Revision 4, 1993

[118] *Advanced Light-Water Reactor Utility Requirements Document – ALWR Passive Plant*, TR-016780, Volume 3, Revision 8, March 1999

[119] *IEEE Standard for Developing Software Life Cycle Processes,* IEEE Std. 1074-1991, 1992

[120] *Software Engineering – Product Quality*, ISO/IEC 9126:1991, 1991

[121] *Use of Numarc/epri Report TR-102348, "Guideline on Licensing Digital Upgrades," in Determining the Acceptability of Performing Analog-to-Digital Replacements under 10 CFR 50.59*, Generic Letter 91-05, April 1995

[122] *Actions to Improve the Detection of Counterfeit and Fraudulently Marketed Products*, Generic Letter 89-02, March 1989

[123] *Requirements Engineering for Digital Upgrades: Specification, Analysis, and Tracking*, EPRI TR-108831, December 1997

[124] *American National Standard Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations*, IEEE Std. 7-4.3.2-1982, 1982

[125] *IEEE Standard Criteria for Accident Monitoring Instrumentation for Nuclear Power Generating Stations*, IEEE Std. 497-1981, 1981

[126] *Instrumentation for Light-Water-Cooled Nuclear Power Plants to Assess Plant and Environs Conditions During and Following an Accident*, NRC Regulatory Guide 1.97, Revision 3, May 1983

[127] *Criteria for Accident Monitoring Instrumentation for Nuclear Power Plants,* NRC Regulatory Guide 1.97, Revision 4, June 2006

[128] *IEEE Standard for Software Safety Plans*, IEEE Std. 1228-1994, 1994

[129] *Systems and Software Engineering – Life Cycle Processes – Risk Management*, ISO/IEC 16085:2006, 2006

[130] *Quality Assurance Requirements of Computer Software for Nuclear Facility Operations,* ASME NQA-2a-1990, 1990

[131] *IEEE Standard for Software Quality Assurance Plans,* IEEE Std. 730.1-1989, 1989

[132] *IEEE Standard Criteria for Accident Monitoring Instrumentation for Nuclear Power Generating Stations,* IEEE Std. 479-2002, September 2002

[133] *IEEE Standard Criteria for Accident Monitoring Instrumentation for Nuclear Power Generating Stations,* IEEE Std. 479-2010, November 2010

[134] *Safety of Nuclear Power Plants: Design Safety Requirements*, IAEA NS-R-1, September 2000

[135] *Instrumentation and Control Systems Important to Safety in Nuclear Power Plants - Safety Guide*, IAEA NS-G-1.3, March 2002

[136] *The Authoritative Dictionary of IEEE Standards Terms – Seventh Edition,* IEEE Std. 100-2000, February 2007

[137] *Systems and Software Engineering – Vocabulary*, ISO/IEC/IEEE 24765:2010, December 2010

[138] *Software for Computer Based Systems Important to Safety in Nuclear Power Plants – Safety Guide*, IAEA NS-G-1.1, November 2000

[139] *Standard for Software Engineering of Safety Critical Software*, CE-1001-STD, Revision 2, December 1999

[140] *Standard for Computer System Engineering*, CE-0100-STD, Revision 0, December 1999