



DRAFT REGULATORY GUIDE

C-138 (E)

SOFTWARE IN PROTECTION AND CONTROL SYSTEMS

Issued for public comments by the
Atomic Energy Control Board
October 1999



Atomic Energy
Control Board

Commission de contrôle
de l'énergie atomique

Canada

DRAFT REGULATORY GUIDE

**Software in Protection and Control Systems
C-138 (E)**

**Issued for Public comments by the
Atomic Energy Control Board
October 1999**

AECB Regulatory Documents

The Atomic Energy Control Board (AECB) operates within a legal framework that includes law and supporting regulatory documents. Law includes such legally enforceable instruments as acts, regulations, licences and directives. Regulatory documents such as policies, standards, guides, notices, procedures and information documents support and provide further information on these legally enforceable instruments. Together, law and regulatory documents form the framework for the regulatory activities of the AECB.

The main classes of regulatory documents developed by the AECB are:

- **Regulatory Policy:** a document that describes the philosophy, principles and fundamental factors used by the AECB in its regulatory program.
- **Regulatory Standard:** a document that is suitable for use in compliance assessment and describes rules, characteristics or practices which the AECB accepts as meeting the regulatory requirements.
- **Regulatory Guide:** a document that provides guidance or describes characteristics or practices that the AECB recommends for meeting regulatory requirements or improving administrative effectiveness.
- **Regulatory Notice:** a document that provides case-specific guidance or information to alert licensees and others about significant health, safety or compliance issues that should be acted upon in a timely manner.
- **Regulatory Procedure:** a document that describes work processes that the AECB follows to administer the regulatory requirements for which it is responsible.

Document types such as regulatory policies, standards, guides, notices and procedures do not create legally enforceable requirements. They support regulatory requirements found in regulations, licences and other legally enforceable instruments. However, where appropriate, a regulatory document may be made into a legally enforceable requirement by incorporation in an AECB regulation, a licence or other legally enforceable instrument made pursuant to the *Atomic Energy Control Act*.

DRAFT REGULATORY GUIDE**Software in Protection and Control Systems****C-138 (E)****October 1999****NOTICE**

On March 20, 1997, Bill C-23, the *Nuclear Safety and Control Act* (NSC Act), received Royal Assent. New regulations that are derived from this Act will become law and replace the existing regulations. Draft Regulatory Guide C-138 references the *NSC Act* and new regulations, which will come into force in 2000 on a date to be fixed by order of the Governor in Council.

About this document**Comments**

In order for interested persons to determine this document's impact and value, public comments are being solicited over two month consultation. At the end of this consultation trial period, comments will be studied to determine how best to improve the document. Comments on this guide will be most helpful if received in writing by December 30, 1999. Reference our file number 1-8-8-138, and direct enquiries and/or comments to the address below.

Document availability

The document can be viewed on the AECB website at www.aecb-ccca.gc.ca. A copy of C-138 can be ordered in English or French from:

Operations Assistant
Corporate Documents Section, Atomic Energy Control Board
P.O. Box 1046, Station B, 280 Slater Street
Ottawa, Ontario K1P 5S9
CANADA
Telephone (613) 996-9505
Facsimile (613) 995-5086
E-mail: reg@atomcon.gc.ca

Contents

About this document	i
Comments	i
Document availability	i
Purpose	1
Scope	1
Categorization	2
1. Technical Requirements Specification	2
1.1 Software requirements specification	2
1.2 Systematic inspection of software design and implementation	4
1.3 Software testing	6
2. Software Development Process, Management and Quality Assurance	7
2.1 Plans, standards and procedures	7
2.2 Configuration management	7
2.3 Software development process	8
2.4 Using pre-developed software	11
3. System Considerations	12
3.1 System specification	12
3.2 System design	13
3.3 Software-hardware interface	13
Glossary	16
References	20

Purpose

This Regulatory Guide describes:

- what constitutes a sufficient description of software used in protection and control systems
- what constitutes a sufficient demonstration that the software design is adequate

This document is not a standard, nor is it intended to be used in place of a standard for software development or quality assurance. This Regulatory Guide is intended to assist AECB staff in their assessment of software, and to guide applicants and licensees in the preparation of their submissions to the AECB.

Scope

The main focus of this Regulatory Guide is the software component of protection and control systems, including software installed as firmware. The complete system and the other components are also important, but comprehensive guidance on how they should be described and how their adequacy should be demonstrated is beyond the scope of this document. However, Section 4 discusses the relationship between the software and the system as a whole, as well as the interfaces between the software and the other components, including the interface to the computer that runs the software.

This Regulatory Guide is not intended to be applicable to non-real-time software used for the design and development of other systems.

This Regulatory Guide concerns the software in systems that:

- provide protection against the undesirable release of radioactive materials or radioactivity
- control nuclear reactions, control the movement of radioactive materials, or control the cooling of radioactive materials
- mitigate the effects of failures in other systems so that the risk of exposure to radiation is reduced

This Regulatory Guide is relevant to new software, including:

- software in new systems
- software that implements new requirements in existing systems
- software that replaces existing software

Changes to existing software to correct or adjust existing functionality should be carefully controlled. Such changes create opportunities to improve the existing software documentation to bring it closer to the quality defined in current standards and in this Regulatory Guide.

This Regulatory Guide does not define any regulations or license conditions on which systems, or changes to systems, must be approved by the AECB before installation. It defines only what description and demonstration of adequacy is appropriate for software, if such a description and demonstration is required under current regulations and license conditions.

Categorization

Applicants or licensees may choose to categorize software to various levels according to the safety significance of the system and the role the software plays with respect to the safety of the system. Although the AECB does not currently mandate a categorization method for software-based systems, this Regulatory Guide assumes that such categories exist. The document refers to three suggested categories of software, with “level 1” software having the most significance to safety and “level 3” software having the least significance to safety. In this way the detailed guidance included in this Guide can address different requirements as may be appropriate for different categories of software. Software with no safety significance is not within the scope of this Regulatory Guide.

1. Technical Requirements Specification

During the development of software, three aspects are of particular importance in providing evidence of completeness, correctness and safety:

- specification of software requirements
- systematic inspection of software design and implementation
- software testing

These three aspects are described in detail in this section. A fourth important aspect, the process and management of the process used to develop the software, will be described in Section 2.

1.1 Software requirements specification

The attributes that should be met by the software requirements specification [2] are described in the following paragraphs and apply to levels 1, 2, and 3 software unless stated otherwise. It is not the intent of this Guide to suggest how these attributes are to be fulfilled. This should be covered by the software development standards and procedures proposed by the licensee.

The software requirements specification document should be clear and consistent. The document should be organized in a logical manner such that its precision, accuracy, consistency and completeness can be checked systematically by reviewers and by the AECB. Each piece of information should have a single, identifiable place in the document. Requirements should be specified in such a way that changes can be introduced later without introducing inconsistencies.

Software requirements should be correct with respect to the role of the software in the larger system and its environment. They should also accurately describe the required behaviour of the software implementation. If the actual behaviour of the software implementation differs in any way from the requirements specification, corrections should be made to either the software or the requirements.

Software requirements should form a solid base for the subsequent software development. They should include all information about the role of the software within the larger system and its environment, especially the interfaces with the operator, other systems and components. Any normal situation and any credible abnormal situation in the computer and process system that could arise during operation should be covered. The document should include the following requirements: functional, performance, safety, reliability, maintenance and user interface. It should be sufficiently complete that if all requirements are met, the software will be considered adequate and acceptable.

FUNCTIONAL REQUIREMENTS specify all inputs and outputs, and the required functional relationship between outputs and inputs. Unless completely defined in a system design document, specification of inputs and outputs should define the precise relationship between monitored quantities in the environment and the software inputs, and between software outputs and the controlled quantities in the environment. It is particularly important to define how continuous and unlimited quantities will be represented by discrete and limited data values. The entire domain of input values should be covered. Formal notation should be used for software level 1. For level 2, formal notation, diagrammatic techniques, pseudocode, structured English or a combination of these should be used, but natural language is not acceptable. Natural language is acceptable for level 3 software only.

PERFORMANCE REQUIREMENTS include all necessary timing relationships between inputs and outputs, including permitted deviations in time. Performance requirements should also include resource requirements.

SAFETY REQUIREMENTS specify the software behaviour as well as constraints on software behaviour that is safe with respect to system-level hazard analysis. If avoidance or mitigation of system hazards depends on particular functional or performance requirements, then references to these requirements should be included.

RELIABILITY REQUIREMENTS specify the numerical reliability targets for the software based on system level reliability targets and reliability targets or known reliability of other components. The definition of what constitutes a failure for reliability purposes should be specified. Refer to Section 1.3 for information on random testing to demonstrate that the reliability target for level 1 software is met. Levels 2 and 3 software should have reliability targets that will not need explicit demonstration.

MAINTENANCE REQUIREMENTS specify the functions or interfaces that are expected to change within the operational life of the software, such as the interfaces to other systems, to the operator, or to hardware devices, and should specify any periodic inspection and testing requirements. If they are known, the frequency of changes, number of changes and corresponding time frame of expected changes should be specified.

USER INTERFACE REQUIREMENTS should be clearly specified to ensure that the software adequately supports the necessary user functions and tasks. User requirements for the expected range of plant conditions should be extracted from the results of function and task analysis. Further information on function and task analysis to be performed is included in reference [14].

Statements describing software should be unambiguous so that there is no doubt about their meaning. If a specification is intended to allow several distinct behaviours, specify what behaviour is allowed and what behaviour is not allowed. Each requirement should be precise enough that a test or verification is feasible to distinguish between a correct and an incorrect implementation and to derive acceptance criteria for this implementation.

1.2 Systematic inspection of software design and implementation

Perform a systematic inspection of both the functionality and the safety of the software. The aim is to provide evidence both that the software does what is required and is safe, and that it does not do anything that is not required or is unsafe. Recognizing that methods of software design and implementation are undergoing considerable change, this Regulatory Guide does not mandate any particular inspection method. However, the products of the software development phases should meet the basic criteria of correctness, completeness and safety (see Section 2.3 for further quality attributes). In addition, the verification and validation mentioned in Section 1.2 refers to software verification and validation, not to human factors verification and validation which are beyond the scope of this document.

FUNCTIONAL ANALYSIS: The following applies to levels 1, 2, and 3 software, unless stated otherwise. Functional analysis should include:

- the demonstration that the software performs all required functions and does not perform any unintended functions. For level 1 software, AECB recommends that both the software requirements specification and the design documentation are based on formal notation and techniques so that the functional analysis can use mathematical methods and automated tools.
- the verification that each product is complete, is internally consistent and consistent with preceding products, and conforms to the standards and guides as specified in the project plan.
- the validation that the final system meets all system requirements and specified user interface requirements. Validation is different from verification in that validation checks the final product against the original intent, while verification checks an interim product against the product that immediately preceded it.

SOFTWARE SAFETY ANALYSIS: The system level safety and hazard analysis used to categorize the criticality of the software should be submitted to the AECB at an early stage of the project so that the level of further analysis that is required can be determined.

The software design should incorporate fail-safe and fault-tolerant features where the increase in safety justifies the additional complexity. Analysis of software safety should include both the expected and known possible operations of the user interface.

The software safety analysis should demonstrate that the software does not contribute to any unsafe system states under expected operating and accident conditions. A system-level hazard analysis should have identified system hazards and traced them through the system to determine the contribution of the software to each hazard. This analysis should be revised and refined as the design is being implemented.

If a software failure could result in serious consequences, the hazard analysis should be extended into the software design and code. This analysis might result in changes that reduce the frequency with which hazardous states occur, or changes that mitigate the consequences of certain failures so that they do not result in hazardous states.

Level 1 software must be simple enough that a complete analysis of software safety is both feasible and credible. Safety critical software should be isolated from non-critical software. Physical isolation by means of hardware is preferable to isolation by means of software only.

1.3 Software testing

Software is tested to find and remove all defects and to establish confidence in the safety and reliability of the final product.

Functional testing exercises the software with inputs selected to cover the functionality and logic of the software.

Random testing exercises the system with inputs randomly selected from a realistic operating profile.

TEST PLAN: The test plan should be written before tests are run. The plan should state the coverage criteria to be achieved by testing. Records should be kept of all test procedures and results. Test results should be cross-referenced with test plans, configuration of both test equipment and components under test, and change control records that result from problems detected during the tests.

FUNCTIONAL TESTING: For level 1 and 2 software, functional testing includes integration testing, system testing and unit testing. For level 3 software, functional testing includes integration and system testing (unit testing is not required as some software development methods substitute formal verification for unit testing but may be included as part of the development method).

Functional tests check every function of the software and all failures that the software is designed to tolerate or mitigate. Record a measure of test coverage, including "typical" cases, boundary-value cases, and cases representing all system hazards. Functional testing also includes checking the timing and performance requirements of the software running on the target computer.

RANDOM TESTING: applies to level 1 software only. The AECB recognizes that research is underway in the area of software reliability and that some hypotheses used to apply reliability models to software are questionable. However, statistically valid random testing establishes confidence that a product will function without failure under specific operating conditions and demonstrates that the reliability target is met.

Statistical validity, for level 1 software, should be demonstrated by showing that a large number of independent, randomly selected test cases have been run without failure. The number of test cases to be run is determined by the reliability model used. The selection of input data for the random tests should be shown to represent accurately either real operating conditions or operating conditions in the area of greatest concern. Record a measure of test coverage and relate the number of tests to the reliability requirements. Select a reliable method of detecting failures and run tests on the actual system to be installed or a close equivalent.

2. Software Development Process, Management and Quality Assurance

2.1 Plans, standards and procedures

At the beginning of a project to develop new software within the scope of this Regulatory Guide, the licensee should submit to the AECB an analysis of the safety significance of the system and any associated categorization of the software. The results of this analysis will determine what further submissions will be needed.

All new software development should follow a software development plan and a quality assurance plan. For level 1 and level 2 software, these plans should be submitted to the AECB at the beginning of the project. For level 3 software, these plans need only be available for inspection.

When changes are required to existing software, software maintenance and quality assurance plans that were part of the acceptance of the original software should be followed. Software maintenance plans should be available for AECB inspection.

Software development plans and software maintenance plans should reference the governing standard(s) and the procedures to be followed. This section provides guidance on the content of the standards and procedures. The plan itself defines or references other documents which define:

- the processes to be followed
- the documents to be produced
- the staff roles, responsibilities and qualifications needed
- the organizational structure
- the facilities and tools to be used

For level 1 and 2 software, a schedule showing the documents to be produced is important to help with planning AECB staff assessments.

2.2 Configuration management

Software configuration management should be part of the overall plant configuration management system, but software and its associated documentation require special care because of the changeability and invisibility of electronic data. Software configuration management based on a recognized standard, such as IEEE Std 828 [5], is an integral part of quality assurance program manuals and procedures and takes into account the actual organization of the developers, reviewers and operators. Automated tools should be used to facilitate and enforce configuration management procedures.

Software should have a unique identifier and a list of uniquely identified components and associated documents which relate to it. Configuration management determines:

- how and when these unique identifiers are assigned
- how changes to identified items are controlled
- how relationships between identified items are maintained as changes are made
- who is responsible for each aspect

The documents submitted to the AECB should be the same as those used by the licensee—documents should not be produced for regulatory purpose only.

2.3 Software development process

The licensee should follow accepted standards and procedures defining how the software will be developed. The AECB has accepted some corporate standards and procedures for use on specific projects. These standards and procedures are still under development, so guidance is still needed on what constitutes acceptability. In the long term, software development standards and procedures will stabilize; and, once accepted by the AECB, they will not have to be reassessed each time they are used.

Standards and procedures should be based on well-known and well-understood software engineering principles as well as on experience with similar software and system development projects. Include a mechanism for evaluating the experience gained during each project and incorporating improvements into the standards and procedures. Software development should be subject to at least the same level of quality assurance as other systems having similar roles in the plant. The licensee prepares a project plan for the development of software and updates that plan throughout the development as the situation changes. Standards, procedures and plans should cover the topics outlined in Section 2.3.

SOFTWARE LIFE CYCLE: The period of time that starts when a software product is conceived and ends when the product is no longer available for use [9]. This should be divided into a planned and controlled set of phases in a logical order, such as: requirements, design, implementation, integration, installation, operation, maintenance, succession. All phases should allow for iterations, verification, testing and rework.

METHODOLOGY: The method used to design and document the software should emphasize ease of review. Formal specification reduces ambiguity and allows formal verification for simple systems and is, therefore, recommended for level 1 software. This may also be beneficial in the specification of level 2 software but formal

verification may be impractical. Level 3 software can be specified and verified informally but with emphasis on understandability and clarity.

DESIGN QUALITY ATTRIBUTES: The standards being used should establish the design quality attributes to be achieved; the plans and procedures should be directed at how those attributes will be achieved. The technical details of this Regulatory Guide suggest that the design should be complete, correct, simple, predictable, robust, consistent, structured, verifiable, modifiable, traceable, modular and understandable (see Glossary in page 16).

DOCUMENTATION: Each life cycle activity should end with a product (usually a document) that provides tangible evidence that the activity is complete. The same set of documents should be used by all the persons who develop, review and maintain the software. Submit to the AECB the same documents as those in use on the project—they should be precise, accurate, consistent, complete, verifiable, changeable and traceable (see Glossary page 16).

TOOLS: Tools used in the development of the software should be selected or built to meet explicit criteria derived from project needs that were defined in advance. An assessment report on how well each tool meets these criteria should be identified. Policies and procedures must be in effect and enforced to ensure proper use of these tools.

MAINTENANCE: Software maintenance refers to the process of modifying software because requirements have changed, or because design or implementation errors have been discovered during operation. When problems, symptoms of failure or changes to requirements are identified, an analysis on how best to address them and their impact on safety should be performed. Before making changes to software, a maintenance plan should be prepared. This plan should include development, inspection and testing processes at least as thorough as for the original product.

PROJECT MANAGEMENT: Management responsibilities [6], [8] should include:

- establishing safety policy and culture
- planning, establishing, and implementing a quality assurance program and an effective safety program
- planning life-cycle activity, including milestones, schedules, and submissions to the regulator
- establishing appropriate communication channels among all participants, including decision makers, system experts, operational staff and software experts
- integrating software engineering with system engineering

- establishing required personnel qualifications for each task, and ensuring that only qualified personnel perform these tasks (see Section 2.3)
- establishing responsibility, accountability and authority for the software and documents
- providing tool support and the proper work environment
- subcontractor control (see Section 2.3)
- process improvement

INDEPENDENCE: Each product should be reviewed by at least one person independent of the development team (not including AECB assessment). Systematic inspection and analysis for functionality and safety (see Section 1.2) should be performed and managed by persons independent of those responsible for the software design and implementation. With the possible exception of unit testing, test specification and testing should be performed by persons independent of the developers. Provision should be made for independent audits of the process to check that quality assurance plans and procedures are being followed, and to identify possible improvements to the process. Independent means different persons using different methods, reporting to a different manager and with a separate budget. Level 1 software should have the highest degree of independent inspection. For level 2 and level 3 software, the managerial and budget separation may be less strict, but there should always be different persons assigned to development and verification.

PERSONNEL QUALIFICATIONS: The licensee should be prepared to demonstrate that the person or team responsible for creating and reviewing any given product is properly qualified to do so [6]. The software development plan should define the qualifications needed for each role on the project. An established training program ensures that personnel acquire and maintain the proper qualifications for the role they are assigned.

TEAM SIZE: The teams performing development, inspection and testing should be appropriate to the size and scope of the project. The roles and responsibilities of each team member should be clearly defined in the project plans. Because level 1 software is expected to be simple in design, a large team may indicate inappropriate complexity, giving rise to errors due to poor communication. Level 2 and 3 software are expected to be more complicated and may require larger teams. Regardless of team size, good communication and separation of responsibility are important.

SUBCONTRACTOR CONTROL: If some parts of the software development, systematic inspection or testing are done by subcontractors, the licensee should ensure that the

subcontractors work to the same terms and conditions as are applied to all other aspects of the project. The licensee remains responsible for all submissions to the AECB.

2.4 Using pre-developed software

When a licensee procures software from another organization, the licensee is responsible for ensuring that this software has been developed, inspected and tested in accordance with the accepted governing standard(s). In the case of software that was developed for broader use than the application planned by the licensee, some adjustments are acceptable, as set out in this section.

REQUIREMENT SPECIFICATIONS: Product specifications can be substituted providing they completely and unambiguously specify the software from an external point of view. Constraints on the operating environment and response to illegal inputs should be covered by the specifications.

SYSTEMATIC INSPECTION: There should be evidence that the design and implementation are correct and complete with respect to the product specifications. The licensee should demonstrate that the product specifications describe a component that meets the needs defined by the system design. The licensee should perform a safety analysis to show that the software product cannot contribute to any known system-level hazards.

TESTING: The licensee should plan, conduct and report on acceptance tests based on the product specifications. These tests should cover all required functions and boundary conditions, especially where the intended application may challenge the range of validity of the product. Tests that simulate expected operating conditions should also be run. If proper usage history data are available, this may be acceptable for establishing confidence in the reliability of the product and can substitute for random testing. In such cases, there must be evidence of careful and thorough usage data collection and reporting, and the other uses of the product must be in environments and applications similar to that being proposed.

DEVELOPMENT PROCESS: If development is complete, the requirement to submit plans, standards and procedures in advance, is irrelevant. However, evidence should exist of organizational standards, software development standards, and good management practices that were in place during the product development.

Pre-developed software that will form part of a new system or component should be evaluated to the same level as new software. If the pre-developed software is used only as a tool to assist in the development of software, it can be evaluated to a lower level provided that suitable manual checks are in place on the outputs of the tool.

3. System Considerations

Software must inevitably be integrated with other components including hardware and users to form a system. The scope of this guide does not include the development of either the hardware or the complete system and does not include human factors engineering. However, some system and hardware issues are closely linked with software. This section gives guidance on the assessment of the interfaces between the software and the rest of the system.

3.1 System specification

A system is a bounded physical entity that achieves in its environment a defined purpose through the interaction of its parts. Software development assumes that the bounds and the purpose of the system have been defined in a system specification. If the system's functionality is largely implemented in software, and if the system specification fulfils the attributes described in Section 1.1, a separate software requirements specification may not be needed. In this case, the system specification can be assessed as the software requirements. However, it is also acceptable for the system specification to describe the system requirements in a less detailed manner when a separate software requirement specification provides the complete, formal and detailed description of the software. In this case, it is important to be able to trace and verify each requirement in the separate software requirements specification to the system specification. In either case, final validation of the correctness of the system should be done against the system specification.

3.2 System design

System design should identify the roles of hardware and software in the system. Those roles and the interfaces between those components should be specified completely and unambiguously. To understand properly how the software will operate, the specification of the hardware should fully define the external behaviour for all legal software instructions and should describe the error handling mechanisms. It is expected that the selection of the computer hardware will involve an understanding of the role of the software, and that iterations of system design may be needed as these roles are clarified.

To achieve the system requirements for safety and reliability, the system may need to be designed to use multiple, diverse components performing the same or similar functions. For example, AECB Regulatory Documents R-8 and R-10 require two

independent and diverse protective shutdown systems in Canadian nuclear power reactors [10], [11]. It should be recognized that when multiple components use software to provide the same or similar functionality, there is a danger that design diversity will be compromised. The design should address this danger by enforcing other types of diversity such as functional diversity, independent and diverse sensors, and timing diversity. The existing practical limitations on quantifying the reliability of software suggest that no single software component should be expected to have an unavailability of better than approximately 10^{-4} . If higher system reliability is needed, then the use of multiple, diverse software components would be expected.

3.3 Software-hardware interface

There are several interfaces between the hardware and the software that will affect the software development and analysis. Functional analysis, software design and coding depend on a complete and unambiguous definition of the computer instruction set. This interface may be encapsulated in a compiler for a higher level programming language, as well as in the operating system and device drivers for associated hardware devices. Timing analysis and testing depend on the timing, sequencing and interrupts of the hardware. Because timing analysis is generally difficult and error-prone, the analysis should be followed with performance testing. Performance testing should be done on the target hardware configuration and should be non-intrusive (i.e., the test environment should not add hardware or software that could change the performance).

Analysis of software safety should be part of an overall analysis of system safety for level 1 systems. Software safety analysis depends on knowledge of the failure modes, failure effects and failure frequencies of the hardware. A failure analysis of the hardware could result in requirements for the software to monitor the hardware and respond to detected failures.

ASSEMBLERS, COMPILERS AND OTHER CODE GENERATING TOOLS: Software development always includes the use of tools for translating a software design expressed in a programming language into code that can be executed by the computer hardware. Programming languages range from high-level design languages to low-level detailed assembly languages. Tools that translate high-level languages contain pieces of software and incorporate them into the product, and thus should be treated as pre-developed software. In another sense, these are design tools, and it is their product (the executable code) that needs to be examined in detail.

Design tools should be selected and evaluated against established criteria (see Section 2.3). The supplier of pre-developed software should be able to demonstrate good quality process and product control as well as evidence of thorough verification and

testing (see Section 2.4). Certification or approval of a tool by an international or national validation agency is an advantage but cannot yet be accepted without question. Developers of level 1 and level 2 applications should examine the programming language and translator and select a subset of the language and tool features that contains only parts that are well defined, predictable and well understood. This should be described in the software design and coding procedures.

Assemblers, compilers and other code-generating tools generally include some automated checking of the software design during translation. Such checking is desirable and can be included as part of the reported verification. These tools may also generate run-time checks. If these are used, include them explicitly included as part of the self-checking.

Software testing is generally the only check on the products of these tools. Each test that is run on the executable code also tests the correctness of the tools. If test results show a discrepancy from predictions, tool error is a possibility and should be addressed. The most direct check of the tool that creates the executable code is to independently reverse the process and reproduce the detailed design from the executable code. Although this is not generally feasible, it may be necessary under special circumstances to provide additional confidence for the most critical part of the software or for software that cannot be directly tested.

OPERATING SYSTEMS: Operating systems should be treated as pre-developed software. Complex operating systems should not be used for level 1 software. A small operating system “kernel” can be used if it can be assured to the same level as the application software, but all unnecessary routines should be removed to avoid any unexpected and undesired side-effects. Operating systems pose particular problems with respect to software maintenance. The operating system vendor’s maintenance might not meet the requirements of a licensee’s particular application, yet it may be difficult to obtain the necessary rights, documentation and expertise to take on custom maintenance. These issues should be examined and addressed in the software development plan.

TIMING AND PERFORMANCE: Timing and performance requirements are known to be difficult to specify and verify in general. For level 1 systems, the computer hardware and software should be designed so that timing is rigidly controlled and predictable. For level 2 and level 3 systems, more flexible designs can be used and may be necessary. In order to predict timing and performance in a computing environment that includes multiple interruptible processes, each process should be specified, designed and implemented in a modular fashion, so that the process's timing needs and effects are explicitly defined, and interactions between processes can be understood.

SELF-CHECKING: This is the capability of a system to detect and respond to component failures. Systems that include software can be designed to detect some failures of sensors, devices or computer hardware, or even to detect failures in other parts of the software. After detecting such failures, the software initiates fail-safe, fault-tolerant or diagnostic outputs. While self-checking does offer some benefits, these have to be balanced against the danger of adding excessive complexity.

A system hazards analysis or a Failure Modes and Effects Analysis (FMEA) should be used to identify hardware component failures to be detected. The resulting list of component failures that can and should be detected is incorporated into the software requirement specification. If the analysis has not been done (as may be the case for level 2 and level 3 systems), self-checking requirements can be based on experience with similar systems and components.

In the case of software component failures, common errors such as stack overflow or division by zero can be detected easily. Detection mechanisms should be defined in a project-specific coding procedure that specifies the appropriate responses. More subtle and complicated software failures are harder for the software itself to detect during operation, and such self-checking is generally not recommended. Higher level system design should deal with the possibility of software component failure.

Glossary

Note: Definitions followed by a reference number are quoted or adapted from the corresponding document in the References section of this Guide.

ACCURATE:	Describing the actual behaviour of the system or software [2].
CHANGEABLE:	Having a rigid organization that guarantees that information will be in one clearly defined place in the document and only in that place, so that the document can be changed without becoming inconsistent [2].
COMPILER:	A tool for translating a software design expressed in a programming language into code that can be executed by the computer hardware.
COMPLETE:	Describing or covering all required behaviour of the system or software under any situation that could conceivably arise during operation. Situations that are not expected to arise in operation should be explicitly identified. [2]
CONSISTENT:	Containing no contradictions. Containing uniform notation, terminology, comments, symbology and implementation techniques. [2], [7]
CONTROL SOFTWARE:	Control software takes sensor and operator inputs, implements a control function on those and possibly other inputs, and produces outputs that directly drive actuators that influence nuclear reactions, movement of nuclear material, cooling of nuclear material or containment of radioactivity.
CORRECT:	Able to produce the specified outputs when given the specified inputs and the extent to which the implementation satisfies its specification. [7]
FAIL-SAFE:	The ability of a system to move to a predetermined safe state when a failure occurs.[1], [11]
FAULT-TOLERANT:	The built-in capability of a system to provide continued correct execution in the presence of a limited number of hardware or software faults. [12]

FORMAL:	Having strict rules about what can be written and how it must be interpreted. A syntax and semantics based on precise mathematical definitions. [2]
SYSTEM ANALYSIS:	The identification of functions that must be performed by humans and automation to satisfy plant or system objectives and performance requirements. Function analysis provides the basis for task analysis.
FUNCTIONAL ANALYSIS:	A demonstration that the software performs all required functions and does not perform any unintended functions. It consists of both verification and validation.
FUNCTIONAL TESTING:	Testing which attempts to exercise all parts of a program as specified or designed, with a goal of finding errors.
HARDWARE:	In the context of software-based systems, hardware refers to the physical electronic components of the system. [12]
HAZARD ANALYSIS:	System hazards are conditions or system states that can lead to accidents. Hazard analysis is the examination of a system from the point of view of safety; to identify and assess potential hazards with the objective of eliminating or reducing them, or mitigating their effects. [8]
INDEPENDENT:	Different persons using different tools and methods, reporting to a different manager and with a separate budget. [13]
MAINTENANCE:	Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. [12]
MODIFIABLE:	Capable of incorporating changes. [7]
MODULAR:	The extent to which software is composed of discrete components (modules) such that a change to one component has minimal impact on other components. [12]
OPERATING SYSTEM:	A program or set of programs that provide an interface between application software and a specific set of hardware, especially for

	initialization, resource allocation, process scheduling, file management and communications.
PRECISE:	Unambiguous; permitting no doubt about the meaning.
PREDICTABLE:	Produces expected outputs from known inputs; not subject to random or unknowable effects.
PROTECTION SOFTWARE:	Protection software monitors the behaviour of systems that control, cool or contain nuclear material, and causes mitigating action if possible hazardous conditions are observed.
QUALITY ASSURANCE:	A planned and systematic pattern of all actions necessary to provide adequate confidence that the system and all physical products conform to established technical requirements.
REAL TIME:	Pertaining to a system or mode of operation in which computation is performed during the actual time that an external process occurs, in order that the computation results can be used to control, monitor, or respond in a timely manner to the external process. [12]
ROBUST:	Able to continue to perform some required functions despite violations of the assumptions of the specification. [7]
SAFETY ANALYSIS:	A demonstration that the software does not contribute to any unsafe actions under expected operating conditions and accident conditions.
SAFETY CRITICAL:	Software that has been categorized as the most critical to safety.
SELF-CHECKING:	Software that is able to detect and respond to component failures.
SIMPLE:	Software design that implements functions in the most understandable manner. [7]
SOFTWARE:	Programs, procedures, rules and any associated documentation pertaining to the operation of a computer system [12].
STRUCTURED:	Any technique for organizing and coding programs that reduces complexity, improves clarity, and facilitates modification. [12]
SYSTEM:	A bounded physical entity that achieves in its environment a defined purpose through the interaction of its parts.

SYSTEMATIC INSPECTION:	Functional analysis and safety analysis of the software as performed by the licensee.
TASK ANALYSIS:	Is the systematic evaluation of the functions allocated to plant personnel to identify user requirements.
TESTING:	The process of exercising (running or operating) a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results. [12]
TRACEABLE:	The source information can be found in the products of preceding life cycle phases.
UNDERSTANDABLE:	Can be understood by qualified persons other than the originator—this also implies a design that is simple, consistent and structured.
VALIDATION:	Checking (or testing) that a system or description of a system is complete and consistent with system requirements and user needs.
VERIFICATION:	The process of determining whether or not the products of a given phase of the software life cycle fulfil the requirements established during the previous phase [12] and the requirements of the governing standards and procedures.

References

- [1] Consultative Document C-22, "Quality Assurance Programmes for Nuclear Facilities," Revision 1, Atomic Energy Control Board, 1991.
- [2] INFO-0505, "Documentation of Computerised Safety Systems of Nuclear Power Stations," by D. L. Parnas, AECB Project 2.234.1.
- [3] Consultative Document C-98, "Guidance for Meeting Reliability Requirements for Safety Related Systems of Nuclear Reactor Facilities," Revision 1 (Draft 19, 95/03/20).
- [4] INFO-0247, "Computer Software Configuration Management," by G. Pelletier, PRIOR Data Science Ltd., AECB Project 2.109.1, 1987.
- [5] IEEE Std 828-1990, "IEEE Standard for Software Configuration Management Plans".
- [6] CAN3-N286.2-86, "Design Quality Assurance for Nuclear Power Plants," Canadian Standards Association, 1986.
- [7] IAEA Technical Report Series No. 282, "Manual on Quality Assurance for Computer Software Related to the Safety of Nuclear Power Plants," International Atomic Energy Agency, 1988.
- [8] IAEA Technical Report Series No. 367, "Software Important to Safety in Nuclear Power Plants," International Atomic Energy Agency, 1994.
- [9] IEC 880, "Software for Computers in the Safety Systems of Nuclear Power Stations," International Electrotechnical Commission Publication 880, First Edition, 1986.
- [10] Regulatory Document R-8, "Requirements for Shutdown Systems for CANDU Nuclear Power Plants," Atomic Energy Control Board, 1991.
- [11] Regulatory Document R-10, "The Use of Two Shutdown Systems in Reactors," Atomic Energy Control Board, 1977.
- [12] IEEE Std 610.12 - 1990, "IEEE Standard Glossary of Software Engineering Terminology," Institute of Electrical and Electronics Engineers, and recognized as an American National Standard, 1990.

- [13] Committee for Review of Oversight Mechanisms for Space Shuttle Flight Software Processes, "An Assessment of Space Shuttle Flight Software Development Processes," National Academy Press, 1993.
- [14] INFO-0605 Human Factors Guides, PHF Services Inc., 1995, AECB Project No 2.280.2.