Licensing of safety critical software for nuclear reactors

Common position of international nuclear regulators and authorised technical support organisations

REGULATOR TASK FORCE ON SAFETY CRITICAL SOFTWARE (TF SCS)
consisting of:

Bel V, Belgium

BASE, Germany

CNSC, Canada

CSN, Spain

KAERI, South Korea

KINS, South Korea

NSC, China

ONR, United Kingdom

SSM, Sweden

STUK, Finland

TIS, Germany

REVISION 2024

Foreword

This consensus document has been revised and improved by the Regulator Task Force on Safety Critical Software (TF SCS) several times since its original publication in 2000, in order to provide up-to-date practical guidance and consistent standards of quality in the regulatory review of safety critical software. It addresses practical issues encountered in previous licensing activities. TF SCS member organisations routinely use the document and recommend it to nuclear regulators and licensees throughout the world, for their reference and use.

Disclaimer

The document can be obtained from the following organisations or downloaded from their websites:

Bel V, Subsidiary of the Federal Agency for Nuclear Control Rue Walcourt, 148 B-1070 Brussels, Belgium https://www.belv.be	**Korea Institute of Nuclear Safety (KINS)** 34 Gwahek-ro, Yuseong-Ku, Daejeon 305-338 Korea https://www.kins.re.kr/en
Bundesamt für die Sicherheit der nuklearen Entsorgung (BASE) Willy-Brandt-Str. 5 38226 Salzgitter, Germany https://www.base.bund.de	**Nuclear and Radiation Safety Center (NSC)** PO Box 8088, 54 Hongliannancun, Haidian, Beijing, 100082, PR China http://www.chinansc.cn/home
Canadian Nuclear Safety Commission (CNSC) P.O. Box 1046, Station B, 280 Slater Street Ottawa, Ontario, Canada K1P 5S9 https://www.nuclearsafety.gc.ca	**Office for Nuclear Regulation (ONR)** Redgrave Court, Merton Road, Bootle, Merseyside, L20 7HS United Kingdom http://www.onr.org.uk
Consejo de Seguridad Nuclear (CSN) C/ Justo Dorado 11 28040 Madrid Spain https://www.csn.es	**Strålsäkerhetsmyndigheten (SSM)** **Swedish Radiation Safety Authority** SE-17116 Stockholm Sweden https://www.ssm.se
TÜV Rheinland Industrie Service GmbH (TIS) Zeppelinstraße 1 D-85399 Hallbergmoos, Germany https: //www.tuv.com	**STUK** **Radiation and Nuclear Safety Authority** Jokiniemenkuja 1 FIN-01370 Vantaa, Finland https://www.stuk.fi
Korea Atomic Energy Research Institute (KAERI) Daedeok-Daero, Yuseong-Gu Daejeon, Korea https://www.kaeri.re.kr/eng	

Contents

Executive Summary

Objectives

It is widely accepted that the safety assessment of software (and thus, software-dependent systems) cannot be limited to verification and testing of the end product, ie the computer code. Other factors such as the quality of the processes and methods for specifying, designing and coding have an important impact on the implementation. Existing standards provide some guidance on the regulatory and safety assessment of these factors, but the licensing approaches taken by nuclear safety authorities and by technical support organisations are determined with only limited technical co-ordination and information exchange. It is notable that several software implementations of nuclear safety systems have been marred by costly delays caused by deficiencies, which could have been mitigated by more coordinated and optimised development and qualification processes.

This document is written by the Regulator Task Force on Safety Critical Software (TF SCS), consisting of regulatory authorities and technical support organisations. The task force compared respective licensing approaches, to identify where a consensus already exists, and to introduce greater consistency and more mutual acceptance into current practice. Within this comparison, the term software also includes firmware, microcode and programmable logic.

The 2007 version of this document was completed at the invitation of the Western European Nuclear Regulators' Association (WENRA). The document identifies consensus and common technical positions that reflect good practice, on a set of important licensing issues experienced by members, encountered in the design and operation of computer-based systems used in nuclear power plants for the implementation of safety functions. Although the motivating issues come from experience with nuclear power plants, the positions will in general be applicable to other nuclear installations. The purpose is to introduce greater consistency and more mutual acceptance into current practices. To achieve these common positions, the task force carefully considered the licensing approaches followed in the participating countries.

Use of new and more advanced digital and programmable technology for nuclear applications continues to increase. Recent examples include robotics, autonomous systems and artificial intelligence (including machine learning). The common positions and recommended practices in this document remain applicable to these new technologies as they are considered for use in nuclear power plants.

The document is intended to be useful:

- to coordinate regulators' and safety experts' technical positions in licensing practices, or design and revision of guidelines;
- as a reference in safety cases and demonstrations of safety of software-based systems;

– as guidance for manufacturers, I&C suppliers, designers and potential licensees on the international market, for creating new products, issuing bids and developing new applications.

Scope

The task force decided at an early stage to focus attention on computer-based systems used in nuclear power plants for the implementation of safety functions of the highest safety criticality; namely, those systems classified by the International Atomic Energy Agency as safety systems. Therefore, *the common positions and recommended practices of this report – except those of chapters 1.2 and 1.11 – address safety systems*.

The task force has *not* considered whether or not the common positions and recommended practices are applicable to safety related systems, except for those in chapters 1.2 and 1.11 and listed in 1.11.3.6. For these cases, the task force concluded that specific positions and practices could be dispensed with or relaxed for safety related systems. These positions and practices are therefore explictly stated as applying only to safety systems.

The task force first selected a set of specific technical issue areas that it considered to be, from a regulatory viewpoint, some of the most important and practical issue areas raised in the licensing of software important to safety. Secondly, the task force studied and discussed each of these issue areas in detail until it reached a common position.

Common positions are expressed with the auxiliary verb “shall”. The use of this verb “shall” for common positions is intended to convey that the licensees need to satisfy the requirement expressed in the clause; this is the unanimous agreement of the task force members.

These common position clauses are a common set of requirements and practices in member states represented in the task force. The requirements in the common position section of a particular chapter are necessary but may not be sufficient. In certain cases, alternative practices may suffice, but such alternatives could be difficult to justify, in the opinion of task force members.

Recommended practices in each chapter are expressed with the auxiliary verb “should”. Recommended practices are supported by most task force members, but may not be systematically implemented by all.

Historical Background

In 1994, the Nuclear Regulator Working Group (NRWG) and the Reactor Safety Working Group (RSWG) of the European Commission Directorate General XI (Environment, Nuclear safety and Civil Protection) launched a task force of experts from nuclear safety institutes with the mandate of "reaching a consensus among its members on software licensing issues having important practical aspects". This task force selected a set of key issues and produced an EC (European Commission) report [1] publicly available and open to comments. In March 1998, a project called ARMONIA (Action by Regulators to Harmonise Digital Instrumentation Assessment) was launched with the mission to prepare a new version of the document, which would integrate the comments received and would deal with a few software issues not yet covered. In May 2000, the NRWG approved a report classified by the EC under the category "consensus document" (report EUR 19265 EN [2]). After this publication, the task force continued to work on important licensing aspects of safety critical software that had not yet been addressed. At the end of 2005, the EC disbanded the NRWG. In 2007, Western European Nuclear Regulators' Association (WENRA) invited the task force to pursue and complete a revision of the report. The task force included the common positions and recommended practices of EUR 19265 [2] in the 2007 revision. The task force continued to work on missing and emerging licensing aspects of safety critical software, leading to new published revisions between 2010 and 2022. These have been further revised to produce the present edition.

The U.S. Nuclear Regulatory Commission (NRC) regularly participated in meetings of the task force from 2009 onwards. Although the NRC does not endorse the document for regulatory use by the NRC, it published the 2015 and 2022 revisions as a technical report in NRC's NUREG/IA series [3] because it considered the common positions a valuable technical reference for future improvements in its own regulatory guidance.

The task force has liaised closely with the NEA CNRA working group on digital instrumentation and control (WGDIC) from 2019 onwards to ensure technical and regulatory consistency when revising common or consensus positions.

This revision contains updates to the introductory material, chapters 1.1 (safety demonstration), 1.12 (software design diversity), 2.1 (computer-based system requirements) and 2.2 (computer system architecture and design).

*   *   *

I INTRODUCTION

All government,
– indeed every human benefit and enjoyment,
every virtue and every prudent act –
is founded on compromise and barter.

(Edmund Burke, 1729-1797)

Objectives

It is widely accepted that the safety assessment of software (and thus, software-dependent systems) cannot be limited to verification and testing of the end product, ie the computer code. Other factors such as the quality of the processes and methods for specifying, designing and coding have an important impact on the implementation. Existing standards provide some guidance on the regulatory and safety assessment of these factors, but the licensing approaches taken by nuclear safety authorities and by technical support organisations are determined with only limited technical co-ordination and information exchange. It is notable that several software implementations of nuclear safety systems have been marred by costly delays caused by deficiencies, which could have been mitigated by more coordinated and optimised development and qualification processes.

This document is written by the Regulator Task Force on Safety Critical Software (TF SCS), consisting of regulatory authorities and technical support organisations. It is the result of a comparison of respective licensing approaches, to identify where a consensus already exists, and to introduce greater consistency and more mutual acceptance into current practice. Within this comparison, the term software also includes firmware, microcode and programmable logic.

The 2007 version of this document was completed at the invitation of the Western European Nuclear Regulators' Association (WENRA). The document identifies consensus and common technical positions that reflect good practice, on a set of important licensing issues experienced by members, encountered in the design and operation of computer-based systems used in nuclear power plants for the implementation of safety functions. Although the motivating issues come from experience with nuclear power plants, the positions will in general be applicable to other nuclear installations. The purpose is to introduce greater consistency and more mutual acceptance into current practices. To achieve these common positions, the task force carefully considered the licensing approaches followed in the participating countries.

The document is intended to be useful:

– to coordinate regulators' and safety experts' technical positions in licensing practices, or design and revision of guidelines;
– as a reference in safety cases and demonstrations of safety of software-based systems;
– as guidance for manufacturers, I&C suppliers, designers and potential licensees on the international market, for creating new products, issuing bids and developing new applications.

Scope

The task force decided at an early stage to focus attention on computer-based systems used in nuclear power plants for the implementation of safety functions of the highest safety criticality; namely, those systems classified by the International Atomic Energy Agency as "safety systems". Therefore, *the common positions and recommended practices of this report – except those of chapter 1.2 and 1.11 – address safety systems*.

The task force has *not* considered whether or not the common positions and recommended practices are applicable to safety related systems, except for those in chapter 1.2 and 1.11 and listed in 1.11.3.6. For these cases, the task force concluded that specific positions and practices could be dispensed with or relaxed for safety related systems. These positions and practices are therefore explicitly stated as applying only to safety systems. Some relaxations of requirements for safety related systems are also mentioned in chapter 1.2, and all relaxations are restated in chapter 1.11.

The task force worked on the assumption that the use of digital and programmable technology has in many situations become inescapable. A discussion of the appropriateness of the use of this technology has therefore not been considered. As the most difficult aspects of the licensing of digital programmable systems are rooted in the specific properties of the technology, the document provides practical and technical licensing guidance, rather than proposing basic principles or requirements. The design requirements and the basic principles of nuclear safety in force in each member state are assumed to remain applicable.

Use of new and more advanced digital and programmable technology for nuclear applications continues to increase. Recent examples include robotics, autonomous systems and artificial intelligence (including machine learning). The common positions and recommended practices in this document remain applicable to these new technologies as they are considered for use in nuclear power plants.

This document represents the consensus view of the task force. From this common agreement, regulators can draw support and benefit when assessing safety demonstrations, licensee's submissions and issuing regulations or guidance.

This document should neither be considered as a standard, nor as a new set of international regulations, nor as a common subset of national regulations, nor as a replacement for national policies. It is the account, as complete as possible, of a common technical agreement among regulatory and safety experts. National regulations may have additional requirements or different requirements, but hopefully in the end have no essential divergence with the common positions.

Safety Demonstration Approach

Evidence to support the safety demonstration of a computer-based digital system is produced throughout the system lifecycle, and evolves in nature and substance with the project. A number of distinguishable types of evidence exist on which the demonstration can be constructed.

The task force has adopted the view that three basic related types of evidence should be used to construct the demonstration: evidence related to the characteristics of the product; evidence related to the quality of the development process actually executed in all of the system lifecycle phases; and evidence of the competence and qualifications of the staff involved. In addition, convincing operating experience may support the safety demonstration of pre-existing software.

As a consequence, the task force reached early agreement on an important fundamental principle (see 1.1.3.1) that applies at the inception of any project, namely:

> *A safety demonstration plan[1] shall be agreed upon at the beginning of the project between the regulator and the licensee. This plan shall identify how the safety demonstration will be achieved. More precisely, the plan shall identify the types of evidence that will be used, how and when this evidence shall be produced, and how the evidence contributes to the demonstration.*

[1] A safety demonstration plan is not necessarily a specific document.

This document neither specifies nor imposes the specific contents of a safety demonstration plan. All the subsequent recommendations are founded on the premise that a safety demonstration plan exists and has been agreed upon by all parties involved. The intent herein is to give guidance on the evidence set to be provided, its integration to demonstrate safety and the documentation for the safety demonstration and for the contents of the safety demonstration plan. It is therefore implied that all the evidence and documentation recommended by this document, among others that the regulator may request, should be made available to the regulator.

The safety demonstration plan should include a safety demonstration strategy. For instance, this strategy could be based on a plant independent type approval of software and hardware components, followed by the approval of plant specific features, as it is practised in certain countries.

Often this plant independent type approval is concerned with the analysis and testing of the non-plant-specific part of a configurable tool or system. It is a stepwise verification which includes:

– an analysis of each individual software and hardware component with its specified features, and
– integrated tests of the software on a hardware system using a configuration representative of the plant-specific systems and their environments.

Only properties at the component level can be demonstrated by this plant independent type approval. It must be remembered that a program can be correct for one set of data, and be erroneous for another. Hence assessment and testing of the plant specific software, integrated in the plant-specific system and environment, remains essential.

Licensing Issues: Generic and Lifecycle Specific

The task force first selected a set of specific technical issue areas that it considered to be, from a regulatory viewpoint, some of the most important and practical issues raised in the licensing of software important to safety. Secondly, the task force studied and discussed each of these issue areas in detail until it reached a common position.

These issue areas are partitioned into two sets: “generic licensing issues” and “lifecycle phase licensing issues”. Issues in the second set are related to a specific stage of the computer-based system design and development process, while those of the former have more general implications and apply to several stages or to the whole system lifecycle. Each issue area is dealt with in a separate chapter of this document, namely:

The task force believes that this set of issue areas addresses a consistent set of licensing aspects from the inception of the lifecycle up to and including commissioning. A balanced consideration of these different licensing aspects is needed for the safety demonstration – as the balance may be system-specific, it is not inherently reflected in the level of attention given in this document.

Definition of Common Positions and Recommended Practices

Each chapter (except chapter 1.3 which describes the standards in use by members of the task force) is organised into the following four aspects:

– *Rationale:* technical motivations and justifications for the issue from a regulatory point of view;
– *Issues involved:* description of the issue in terms of the problems to be resolved;
– *Common position:* intended to convey the unanimous views of the task force members on necessary practice for achieving safety;
– *Recommended practices:* supported by most task force members, but may not be systematically implemented by all.

Common positions are expressed with the auxiliary verb "shall" and recommended practices with "should".

The use of the verb "shall" for common positions is intended to convey that the licensees need to satisfy the requirement expressed in the clause; this is the unanimouse agreement of the task force members. The use of "shall" and "requirement" does not imply a mandatory condition incorporated in regulation.

These common position clauses are a common set of requirements and practices in member states represented in the task force. The requirements in the common position section of a particular chapter are necessary but may not be sufficient. In certain cases, alternative practices may suffice, but such alternatives could be difficult to justify, in the opinion of task force members.

The task force included already accepted common practices where these provide useful context and foundations for the issues being addressed.

Recommended practices are supported by most task force members, but may not be systematically implemented by all. Some of these recommended practices originated from proposed common position resolutions on which unanimity could not be reached.

These common positions and recommended practices have of course not been elaborated in isolation. They take into account not only the positions of the participating regulators, but also the guidance issued by other regulators with experience in the licensing of computer-based nuclear safety systems. They have also been reviewed against the international guidance, the technical expertise and the evolving recommendations issued by the IAEA, the IEC and the IEEE organisations. The results of research activities on the design and the assessment of safety critical software by EC projects such as PDCS (Predictably dependable computer systems), DeVa (Design for Validation), CEMSIS (Cost Effective Modernisation of Systems Important to Safety) and by studies carried out by the EC Joint Research Centre have also provided sources of inspiration and guidance. A bibliography at the end of the document gives the major references that have been used by the task force and the consortium.

Historical Background

In 1994, the Nuclear Regulator Working Group (NRWG) and the Reactor Safety Working Group (RSWG) of the European Commission Directorate General XI (Environment, Nuclear Safety and Civil Protection) launched a task force of experts from nuclear safety institutes with the mandate of "reaching a consensus among its members on software licensing issues having important practical aspects". This task force selected a set of key issues and produced an EC (European Commisson) report [1] publicly available and open to comment. In March 1998, a project called ARMONIA (Action by Regulators to Harmonise Digital Instrumentation Assessment) was launched with the mission to prepare a new version of the document, which would integrate the comments received and would deal with a few software issues not yet covered. In May 2000, the NRWG approved a report classified by the EC under the category "consensus document" (report EUR 19265 EN [2]). After this publication, the task force continued to work on important licensing aspects of safety critical software that had not yet been addressed. At the end of 2005, the EC disbanded the NRWG. In 2007, the Western European Nuclear Regulators' Association (WENRA) invited the task force to pursue and complete a revision of the report. The task force included the common positions and recommended practices of EUR 19265 [2] in the 2007 revision. The task force continued to work on missing and emerging licensing aspects of safety critical software, leading to new published revisions between 2010 and 2022. These have been further revised to produce the present edition.

The experts, members of the task force, who actively contributed to this document, are:

Belgium	P-J Courtois, Bel V (1994-2018) (Chairman 1994-2007)
	A Geens, AVN (2004-2006)
	S van Essche, Bel V (2011-2013)
	E Peeters, Bel V (2014-)
Canada	G Chun, CNSC (2013-2020)
	Y-C Liu, CNSC (2021-)
China	Z Wang NSC (2016-)
	C Mao, NSC (2016-)
	Y Feng, NSC (2017-)
Finland	ML Järvinen, STUK (1997-2003)
	P Suvanto, STUK (2003-2010)
	M Johansson, STUK (2010-2018)
	E Sarkio, STUK (2015-)
Germany	M Kersken, ISTec (1994-2003)
	EW Hoffman, ISTec (2003-2007)
	J Märtz, ISTec (2007-2014)
	M Baleanu, ISTec (2014-2017)
	H Miedl, ISTec (2018-)
	R Heigl, ISTec (2018-)
	F Seidel, BfS (1997-2017)
	M Hellmich, BASE (2017-)
Spain	R Cid Campo, CSN (1997-2003)
	F Gallardo, CSN (2003-2017)
	M Martínez, CSN (2013-2017)
	J Galindo, CSN (2018-)
	D Fernández, CSN (2019-)
South Korea	J-S Lee, KAERI (2015-2018)
	Y-M Kim, KINS (2016-)
Sweden	B Liwång, SSM (1996-2015)
	S Persson, SSM (2015-)
United Kingdom	N Wainwright, NII (1994-1999)
	RL Yates, ONR (1999-2013) (Chairman 2007-2013)
	M Bowell, ONR (2007-) (Chairman 2013-)

In 1994, P Govaerts (AVN) was instrumental in setting up the task force. In March 1998, ISTec, NII, and AVN (chair) created a consortium to give research, technical and editorial support to the task force. Under the project name of ARMONIA (Action by Regulators for harmonising Methods Of Nuclear digital Instrumentation Assessment), the consortium received financial support from the EC programme of initiatives aimed at promoting harmonisation in the field of nuclear safety. P-J Courtois (AVN), M Kersken (ISTec), P Hughes, N Wainwright and RL Yates (NII) were active members of ARMONIA. In the long course of meetings and revisions, technical assistance and support was received from J Pelé, J Gomez, F Ruel, JC Schwartz, H Zatlkajova from the EC, and G Cojazzi and D Fogli from the Joint Research Centre, Ispra.

The task force acknowledges and appreciates the support provided by the EC and WENRA during the production of some of the earlier versions of this work.

The US Nuclear Regulatory Commission (NRC) regularly participated in meetings of the task force from 2009 onwards. The NRC participants in task force meetings have included Dr Steven Arndt, William Kemper, Norbert Carte, John Thorp, Sergiu Basturescu and Dr Sushil Birla. Additionally, Michael Waterman and Russell Sydnor of the NRC staff have provided input. Although the NRC does not endorse the document for regulatory use by the NRC, it published the 2015 and 2022 revisions as a technical report in NRC's NUREG/IA series [3] because it considered the common positions a valuable technical reference for future improvements in its own regulatory guidance.

More recently, members from Canada (CNSC), South Korea (KAERI and KINS) and China (NSC) joined the task force, as its content continues to be recognised internationally as increasingly relevant and applicable.

The task force has liaised closely with the NEA CNRA working group on digital instrumentation and control (WGDIC) from 2019 onwards to ensure technical and regulatory consistency when revising common or consensus positions.

This revision contains updates to the introductory material, chapters 1.1 (safety demonstration), 1.12 (software design diversity), 2.1 (computer-based system requirements) and 2.2 (computer system architecture and design).

* * *

II GLOSSARY

The following definitions provide meanings of the terms, as used in this document.

The first usage in a chapter of each is highlighted as a defined term.

Availability: The ability of an item to be in a state to perform a required function under given conditions at a given instant of time or over a given time interval, assuming that the required external resources are provided. (IEC 60050-191-02-05)
Note: This term is also used, when quantification is implied, to refer to the mean of the instantaneous availability under steady-state conditions over a given time interval. (Where instantaneous availability is the probability that an item is in a state to perform a required function under given conditions at a given instant of time, assuming that the required external resources are provided.). (IEC 60050-191-11-06)

Category (safety-): One of three possible assignments (safety, safety related and not important to safety) of functions in relation to their different importance to *safety*.

Channel: An arrangement of interconnected *components* within a *system* that initiates a single output. A channel loses its identity where single output signals are combined with signals from other channels such as a monitoring channel or a safety actuation channel. (See IEC 61513 and IAEA SSG 39)

Class (safety-): One of three possible assignments (safety, safety related and not important to safety) of *systems*, *components* and software in relation to their different importance to safety.

Commissioning: The onsite process during which plant *components* and *systems*, having been constructed, are made operational and confirmed to be in accordance with the design assumptions and to have met the safety requirements, the performance criteria and the requirements for periodic testing and maintainability.

Common cause failure: *Failure* of two or more structures, *systems* or *components* due to a single specific event or cause. (IAEA Safety Glossary)

Common position: Requirement or practice unanimously considered by the member states represented in the task force as necessary for the licensee to satisfy. The use of the term “requirement” shall not necessarily be construed as a mandatory condition incorporated in regulation.

Completeness: Property of a *formal* system in which every true fact is provable.

Complexity: The degree to which a system or component has a design or implementation that is difficult to understand and verify. (IEC STD 100-2000)
Note: This document does not use the term complexity in the context of metrics.

Component: One of the parts that make up a *system*. A component may be hardware or software (or a *PLD* in the case of chapter 1.16) and may be subdivided into other components. (IEC 61513 [4])

Computer-based system (in short also the *System*)*:* The plant *system important to safety* in which is embedded the computer implementation of the *safety/safety related function(s)*.

Computer system architecture: The hardware *components* (processors, memories, input/output devices) of the *computer-based system*, their interconnections, physical separation and electrical isolation, the communication systems, and the mapping of the software functions on these components.

Consistency: Property of a *formal* system which contains no sentence such that both the sentence and its negation can be proven from the assumptions.

Dangerous failure: Used as a probabilistic notion, *failure* that has the potential to put the *safety system* in a *hazardous* or fail-to-function state. Whether the potential is realised may depend on the *channels* of the *system* architecture; in systems with multiple channels to improve *safety*, a dangerous hardware failure is less likely to lead to the overall dangerous or fail-to-function state. (IEC 61508-4, 3.6.7)

Diversity: Existence of two or more different ways or means of achieving a special objective. (IEC 60880 [5])

Diversity design options/seeking decisions: Choices made by those tasked with delivering *diverse* software *programs* as to what are the most effective methods and techniques to prevent common cause *failure* of the programs.

Equivalence partitioning: Technique for determining which classes of input data receive equivalent treatment by a *system*, a software *module* or *program*. A result of equivalence partitioning is the identification of a finite set of software functions and of their associated input and output domains. Test data can be specified based on the known characteristics of these functions.

Error: Manifestation of a *fault* and/or state liable to lead to a *failure*.

Failure: The termination of the ability of a structure, system or component to perform a required function. (IEEE STD 100)

Failure mode: The physical or functional manifestation of a failure. (ISO/IEC/IEEE 24765)

Fault: Defect or flaw in a hardware or software *component*. (IEEE STD 100 definition 13)

Formal methods, formalism: The use of mathematical models and techniques in the design and analysis of computer hardware and software.

FMECA: failure modes and effects criticality analysis.

Functional diversity: Application of *diversity* at the functional level (for example, to have trip activation on both pressure and temperature limit). (IEC 60880 [5] and IEC 61513 [4])

Functional requirement: Service or function expected to be delivered.

Graded requirement: A possible assignment of graded or relaxed requirements on the qualification of the software development processes, on the qualities of software products and on the amount of *verification* and validation resulting from consideration of what is necessary to reach the appropriate level of confidence that the software is fit for purpose to execute specific functions in a given *safety category*.

Harm: Physical injury or damage to the health of people, either directly or indirectly, as a result of damage to property or the environment. (IEC 61508-4, 3.1.1)

Hazard: Potential source of *harm*. (IEC 61508-4, 3.1.2)

HDL: hardware description language.

I&C: instrumentation and control.

IAEA: International Atomic Energy Agency.

IEC: International Electrotechnical Commission.

IEEE: Institute of Electrical and Electronic Engineers.

IP: intellectual property.

Licensee safety department: A licensee's department, staffed with appropriate computer competencies independent from the project team and operating departments appointed to reduce the *risk* that project or operational pressures jeopardise the *safety systems'* fitness for purpose.

NPP: nuclear power plant.

Non-functional requirement: Requirement that specifies a property of a system or of its element(s) in addition to their functional behaviour.
Note 1: See 1.1.1 for further detail and example properties.
Note 2: ISO 25010:2011 uses the term "quality requirement", defined as a requirement for the corresponding intrinsic quality property to be present.

Pfd: *probability of failure* on demand.

Plant safety analysis: Deterministic and/or probabilistic analysis of the selected postulated initiating events to determine the minimum *safety system* requirements to ensure the safe behaviour of the plant. System requirements are elicited on the basis of the results of this analysis.

Pre-existing software (PSW): Software which is used in a *NPP* *computer-based system important to safety*, but which was not produced by the development process under the control of those responsible for the project (also referred to as "pre-developed" software). "Off-the-shelf" software is a kind of PSW.

Probability of failure: A numerical value of *failure* rate normally expressed as either probability of failure on demand (*pfd*) or probability of *dangerous failure* per year (eg 10^{-4} pfd or 10^{-4} probability of dangerous failure per year).

Programmable logic device (PLD): an integrated circuit with logic that can be configured by the designer.

Programmed electronic component: An electronic *component* with embedded software that has the following restrictions:
- its principal function is dedicated and completely defined by design;
- it is functionally autonomous;
- it is parametrizable but not programmable by the user.

These components can have additional secondary functions such as calibration, self-tests, communication, information displays. Examples are relays, recorders, regulators, smart sensors and actuators.

PSA: probabilistic *safety* assessment.

QRA: quantitative *risk* assessment.

Recommended practice: Requirement or practice considered by most member states represented in the task force as necessary for the licensee to satisfy.

Regulator: The regulatory body and/or authorised technical support organisation acting on behalf of its authority.

Reliability: Continuity of proper service. Reliability may be interpreted as either a qualitative or quantitative property.

Reliability level: A defined numerical *probability of failure* range (eg 10^{-3} > *pfd* >10^{-4}).

Reliability target: *Probability of failure* value typically arising from the *plant safety analysis* (eg *PSA*/*QRA*) for which a *safety demonstration* is required.

Requirement specification: Precise and documented description or representation of a requirement.

Risk: Combined measure of the likelihood of a specified undesired event and of the consequences associated with this event.

(Nuclear) Safety: The achievement of proper operating conditions, prevention of accidents or mitigation of accident consequences, resulting in protection of workers, the public and the environment from undue radiation hazards. (IAEA Safety Glossary)
Note: In this document nuclear safety is abbreviated to safety.

Safety demonstration: The arguments and evidence that support a selected set of claims on the *safety* of the operation of a *system important to safety* used in a given plant environment. This does not necessarily imply a formal claims arguments evidence structure.

Safety integrity level (SIL): Discrete level (one out of a possible four), corresponding to a range of values of the probability of a *system important to safety* satisfactorily performing its specified *safety requirements* under all the stated conditions within a stated period of time.

Safety demonstration plan: A plan, which identifies how the *safety demonstration* is to be achieved; more precisely, a plan which identifies the types of evidence that will be used, and how and when this evidence shall be produced. A safety demonstration plan is not necessarily a specific document.

Safety related systems: An abbreviation for a safety related instrumentation and control system. A *system important to safety* that is not part of a *safety system*. (IAEA Safety Glossary)

Safety system: A *system important to safety* provided to assure the safe shutdown of the reactor or the residual heat removal from the reactor core, or to limit the consequences of anticipated operational occurrences and design basis accidents. (IAEA Safety Glossary)

Security: The prevention of unauthorised disclosure (confidentiality), modification (integrity) and retention of information, software or data (availability).

Shall: Conveys unanimous consensus by the Task Force members for the licensees to satisfy the requirement expressed in the clause.

Should: Conveys that the practice is recommended, ie is supported by most Task Force members but may not be systematically implemented by all.

Smart sensor/actuator: Intelligent measuring, communication and actuation devices employing *programmed electronic components* to enhance the performance provided in comparison to conventional devices.

Software architecture, software modules, programs, subroutines: Software architecture refers to the structure of the modules making up the software. These modules interact with each other and with the environment through interfaces. Each module includes one or more programs, subroutines, abstract data types, communication paths, data structures, display templates, etc. If the *system* includes multiple computers and the software is distributed among them, then the software architecture must be mapped to the hardware architecture by specifying which programs run on which processors, where files and displays are located and so on. The existence of interfaces between the various software modules, and between the software and the external environment (as per the software requirements document), should be identified.

Software maintenance: Software change in operation following the completion of *commissioning* at site.

Software modification: Software change occurring during the development of a system up to and including the end of *commissioning*.

Soundness: Property of a formal system in which every provable fact is true.

SQA: software quality assurance.

SST: software statistical testing

Synchronisation programming primitive: High level programming construct, such as for example a semaphore variable, used to abstract from interrupts and to program mutual exclusion and synchronisation operations between co-operating processes (see eg [2]).

System: When used as a stand-alone term, abbreviation for *computer-based system* or, in the case of chapter 1.16, *PLD*-based system.

Systematic failure: failure, related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors.

Systems important to safety: *Systems* which include *safety systems* and *safety related systems*. Systems important to safety include:
- Those systems whose malfunction or *Failure* could lead to undue radiation exposure of site personnel or members of the public;
- Those systems that prevent anticipated operational occurrences from leading to accident conditions;
- Those systems that are provided to mitigate the consequences of malfunction or *Failure* of structures, systems and components.

(IAEA Safety Glossary)

Transformation tool: A tool such as a code generator or compiler, that transforms a work product text at one level of abstraction into another, usually lower, level of abstraction.

Validation: Confirmation, through the provision of evidence, that a product satisifies its specific intended use or application. (Adapted from ISO/IEC/IEEE 9000:2008(E))
Note 1: The product may be a set of requirements or may be a computer-based system.
Note 2: Validation of a set of requirements will include a demonstration of its correctness, completeness, consistency and unambiguity.

Verification: Confirmation that a product satisfies its specified requirements. (Adapted from ISO/IEC/IEEE 24765:2010(E))
Note: The specified requirements are usually the output of a previous phase or phases.

V&V: verification and validation.

* * *

PART 1: GENERIC LICENSING ISSUES

1.1 Safety Demonstration

"Sapiens nihil affirmat quod non probet."

1.1.1 Rationale

Standards and national rules reflect the knowledge and consensus of experts; the fulfilment of their requirements may not be sufficient to assure safety in all cases. They usefully provide a benchmark for activities such as requirements specification, design, verification, validation, maintenance, operation, and contribute to the improvement of safety demonstration practices.

However, the process of evaluating and approving software for safety and safety related functions is far from trivial, and will continue to evolve. Reviews of licensing approaches showed that, except for procedures that formalise negotiations between licensee and regulator, no systematic method is defined or in use in many member countries for demonstrating the safety of a software-based system.

A systematic and well-planned approach contributes to improving the quality, timeliness and cost-effectiveness of the safety demonstration. The benefit can be at least three-fold, enabling the parties involved to:

– focus attention on the specific safety issues associated with the system, and on the corresponding specific system requirements (see chapter 2.1) that must be satisfied;
– prioritise these system requirements, with a commensurate allocation of resources;
– organise system requirements so that the arguments and evidence are limited to what is necessary and sufficient for the parties involved to reach consistent conclusions.

A safety demonstration addresses the relevant properties of a particular system operating in a specific environment (properties other than safety need consideration only to the extent they affect safety). It is therefore specific and carried out on a case-by-case basis. However, the demonstration still cannot be performed "à la carte" with a free choice of means and objectives.

The safety functions for a system depend on properties in addition to the functional behaviour of the system hardware and software. Hence, claims that safety functions are satisfied need to be supported by claims that the system and its functions have these particular properties. Examples of relevant properties are security, reliability, availability, maintainability, testability, usability, accuracy and performance (eg accuracy, timing constraints, cycle and required response times in relation to rates of change of the plant parameters). The relevance of particular properties to the safety demonstration will differ on a case-by-case basis.

1.1.2 Issues Involved

1.1.2.1 Overview

Safety demonstration in practice is complex.

- It is not a single linear process (see 1.1.2.2).
- It is not limited to an analysis of the system by itself, but needs to consider also the system's interactions with its environment (see 1.1.2.3).
- It is not limited to an analysis of the initially implemented design. The system and its environment change over the life of the system. Therefore, the full lifetime needs to be considered (see 1.1.2.4).
- A unifying framework is needed (see 1.1.2.5).
- Goals and assumptions need to be clear and explicit (see 1.1.2.6).
- Planning is needed to ensure that appropriate evidence is available to support the goals (see 1.1.2.7).

1.1.2.2 Various approaches are possible to demonstrate system safety

A licensee and a regulator have several possible alternatives for the demonstration of the safety of a computer-based system. The demonstration may be conditioned on the provision of evidence of compliance with a set of agreed rules, laws, standards, or design and assessment principles (a rule-based approach). It also may be conditioned on the provision of evidence that certain specific residual risks are acceptable, or that the properties necessary for system safety are achieved (a goal-based approach). Any combination of these approaches is of course possible. For instance, compliance with a set of rules or a standard can be invoked as evidence to support a particular system requirement. A safety demonstration may be multi-legged, supported by many types of evidence.

The selection should be made with awareness of the issues associated with the respective alternatives. The law-, rule-, design principle- or standard- compliance approach often fails to demonstrate convincingly *by itself* that a system is safe enough for a given application, thereby entailing licensing delays and costs. A multi-legged approach may suffer from the same shortcomings. By collecting evidence in different and orthogonal directions, which remain unrelated, one may still fail to convincingly establish a system property. The safety goal approach requires ensuring that the initial set of goals, which is selected, is complete and coherent.

1.1.2.3 System safety depends upon its environment

Most safety requirements are related to the application in which the computer and its software are embedded. Many pertinent arguments to demonstrate safety – for instance the provision of safe states and safe failure modes – are not provided by the computer system design and implementation, but are determined by the environment and the role the system is expected to play in it. Guidance on the safety demonstration of computer-based systems often concentrates on the V&V of the detailed design and implementation and pays little attention to a top-down approach starting with the environment-system interface.

1.1.2.4 System safety is affected by changes over its life

Safety depends not only on the design, but also, and ultimately, on the installation of the integrated system, on operational procedures, on procedures for (re)calibration, (re)configuration, maintenance, even for decommissioning in some cases. A safety case is not closed before the actual behaviour of the system in real conditions of operations has been found acceptable. The safety demonstration of a software-based system therefore involves more than a code correctness proof. It involves a large number of various claims spanning the whole system lifecycle. This is well known to application engineers, but often not sufficiently addressed in the computer system design.

Since evidence to support the safety demonstration of a computer-based system is produced throughout the system lifecycle, the demonstration itself should evolve in nature and substance with the project, ideally starting with the safety demonstration plan which should include the planned safety demonstration approach.

1.1.2.5 Safety demonstration may involve claims, arguments and evidence

The key issue of concern is how to demonstrate that the system requirements (see 2.1) have been met. Basically, a safety demonstration is a set of arguments and evidence elements that support a selected set of safety claims of the operation of a system important to safety used in a given plant environment. (See Figure 1 below and eg [6], [7] and [8].)

Claims are statements that can be shown and agreed to be true (or false). Once complete, a satisfactory safety demonstration contains no false claims, all counterclaims are adequately addressed and each satisfied claim will contribute to the demonstration's overall objective.

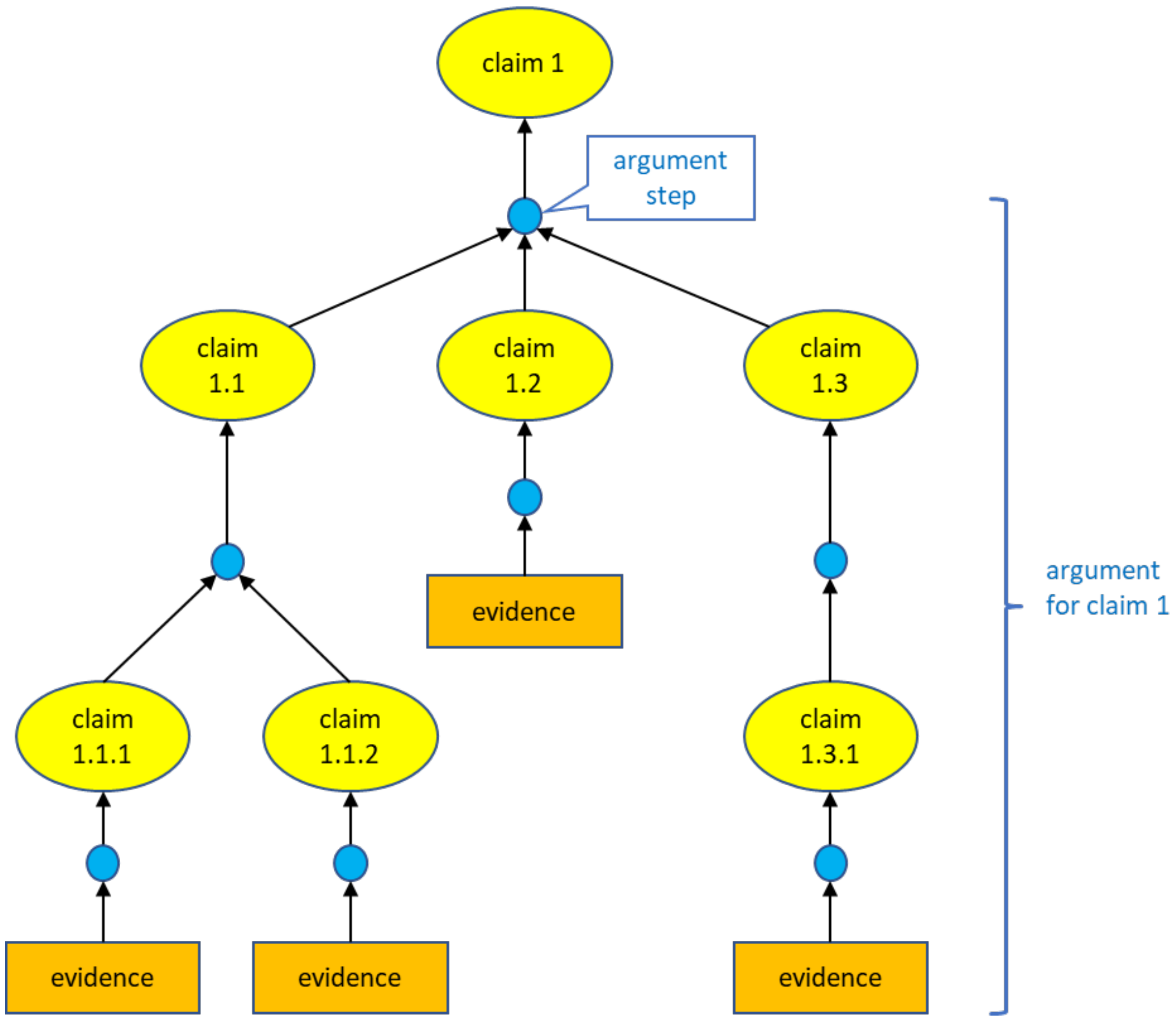
Claims are supported by further claims (ie sub-claims of the original claim), which themselves are supported either by further claims or by evidence.

Claims include functional and/or non-functional requirements that can be shown to be satisfied by the system. A claim may be the existence of a safe state, the correct execution of an action, a specified level of reliability or availability, etc. The aim is for the set of claims to be coherent and as complete as possible, so that this set demonstrates the system is safe. Claims can be decomposed into, and inferred from, sub-claims at various levels of the system architecture, design and operations. Claims may coincide with the computer-based system requirements that are discussed in chapter 2.1. They also may pertain to a property of these requirements (completeness, coherency, soundness) or claim an additional property of the system that was not part of the initial requirements, as in the case of pre-existing software for instance.

Evidence identifies facts or data that is sufficiently convincing to support its claim, without further evaluation, quantification or demonstration. This requires a consensus from all parties involved that the evidence is sufficiently convincing. Evidence on which the demonstration may be constructed includes documents showing the quality of the development process, product specifications, results from the validation of the specification, a demonstration of the correctness of its implementation, and evidence of the competence and qualifications of the staff involved in all of the system lifecycle phases. An evidence item may not completely justify the claim; other evidence items may fill the gaps. Convincing operating experience may be needed to support the safety demonstration of pre-existing software (see chapter 1.14). To realise their value, every item of evidence is logically related to one or more specific claims rather than merely juxtaposed.

An *argument* provides the reasoning to demonstrate how any particular claim is met. Hence an argument may consist of a supporting claim structure, reasoning in the form of a textual narrative, or both (either in combination or as alternative expressions of the same argument), supported by evidence. An argument may use reasoning rules (in an argument step) to describe why the set of sub-claims for a particular claim is an appropriate concretion, substitute or decomposition of that claim. It is particularly important for the argument to show how the evidence is sufficient for the claim it supports to be accepted as true, including assumptions, context and limitations. Arguments also consider counterclaims and conflicting evidence (also termed defeaters) to show that all relevant facts have been taken into account to reach a logically sound conclusion.

Figure 1: Example claim, arguments and evidence structure



1.1.2.6 Clarity in the safety goal and representation of the system and its environment is important

Safety is a property, the demonstration of which – in most practical cases – cannot be strictly experimental and obtained by eg testing or operational experience, especially for complex digital systems. For instance, safety does not only include claims of the type: “the class X of unacceptable events shall occur less than once per Y hours in operation”. It also includes or subsumes the claim that the class of events X is adequately identified, complete and consistent. Thus, safety cannot be discussed and shown to exist without providing clear, coherent, specific goals (eg supported with models of postulated accidents), using *accurate descriptions of the system architecture, of the hardware and software design of the system behaviour and of its interactions with the environment.* These descriptions must include unintended systems behaviour, be unambiguously understood and agreed upon by all those who have safety case responsibilities: users, designers and assessors. This is unfortunately not always the case. Claims, although usually based on a huge amount of engineering and industrial past experience, may be only poorly specified. The simplifying assumptions behind the descriptions and representations of the system and its environment and for the evaluation of safety are not always sufficiently explicit. As a result one should be wary of attempts to “juggle with assumptions” (ie to argue and interpret hypotheses about the system, its environment and accidents in order to make unfounded claims of increased safety and/or reliability) during licensing negotiations.

The code itself is the most accurate available description of the behaviour of the software in the system, hence evidence directly related to the code and its properties is invaluable.

1.1.2.7 Planning the safety demonstration ensures appropriate evidence is available

The explicit identification of claims, arguments and evidence can be useful for planning a safety demonstration. At this stage, claims represent goals – statements that the licensee aims to show to be true. Early identification of claims, the claims they depend on and planned evidence provides an opportunity for the licensee and all other parties, including the regulator, to agree on exactly what needs to be demonstrated and how this can credibly be achieved. Agreed claims provide documented purpose and requirements for the evidence to be produced. Any missing claims can be highlighted early, so the licensee still has the opportunity to add them to the demonstration, propose new evidence and/or modify the design.

Practical experience has found that resources can be saved when evidence that would normally be produced under default processes can be shown to be unnecessary to satisfy a required claim, because for example sufficient alternative evidence already exists or is planned elsewhere, or even the claim the evidence supports is not needed in the planned demonstration.

Claims will need to be updated as evidence becomes available, since the reality the evidence reveals will not always exactly match the original goal. This provides another opportunity for all parties to agree adjusted claims/goals as soon as the evidence becomes available, for example by

- accepting the adjusted claims as broadly equivalent,
- modifying the design,
- adding new compensating claims, and/or
- planning new activities to generate additional evidence.

1.1.3 Common Position

1.1.3.1 A safety demonstration plan shall be agreed upon at the beginning of the project between the regulator and the licensee. A safety demonstration plan is not necessarily a specific document.

1.1.3.2 The licensee shall identify all software used in systems important to safety (including pre-existing software, firmware in programmable field devices etc). All identified software shall be covered by a safety demonstration plan.

1.1.3.3 The licensee shall produce a safety demonstration plan as early as possible in the project, and shall make this safety demonstration plan available to the regulator.

1.1.3.4 The safety demonstration plan shall identify how the safety demonstration will be achieved.

1.1.3.5 The safety demonstration plan shall define

a the activities to be undertaken in order to demonstrate that the system is adequately safe for its intended use and environment,

b the organisational arrangements needed for the demonstration (including independence of those undertaking the safety demonstration activities) and

c the programme (the activities and their inter-relationships, allocated resources and time schedule) for the safety demonstration.

1.1.3.6 The plan shall identify the types of evidence that will be used, its logical organisation, and how and when this evidence shall be produced. The following three different types of evidence shall be considered:

a evidence related to quality of the development process

b evidence related to adequacy of the product c evidence of the competence and qualification of the staff involved in all of the lifecycle phases, including verification and validation and safety analysis.

1.1.3.7 The licensee shall implement the safety demonstration plan, including its application to its supply chain.

1.1.3.8 If the safety demonstration is based on a claim argument evidence structure, then the safety demonstration plan shall identify the claims that are made on the system, the types of evidence that are required, the arguments that are applied, and when this evidence shall be produced.

1.1.3.9 The safety demonstration shall identify a complete and consistent set of functional and non-functional requirements (for each system in scope) that need to be satisfied (see 2.1). These requirements shall address at least:

a The validity of the requirements and the adequacy of the design specifications; those must satisfy the plant/system interface safety requirements and deal with the constraints imposed by the plant environment on the computer-based system;

b The correctness of the design and the implementation of the embedded computer system for ensuring that it performs according to its specifications, including freedom from unsafe behaviour;

c The operation and maintenance of the system to ensure that the safety claims and the environmental constraints will remain satisfied during the whole lifetime of the system. This includes claims that the system does not display behaviours unanticipated by the system specifications, or that potential behaviours outside the scope of these specifications will be detected and their consequences mitigated.

1.1.3.10 The licensee shall make available all evidence identified in the safety demonstration.

1.1.3.11 When upgrading an old system, with a new digital system, it shall be demonstrated that the new system preserves the existing plant safety properties, eg considering timing constraints, delays and response times etc.

1.1.3.12 If a claim argument evidence structure is followed, the safety demonstration shall accurately identify the evidence that supports all claims, as well as the arguments that relate the claims to the evidence.

1.1.3.13 The plan shall precisely identify the regulations, standards and guidance that are used for the safety demonstration. The applicability of the standards to be used shall be justified, with respect to consistency adequacy and completeness of the identified requirements, with potential deviations and omissions being evaluated and justified.

1.1.3.14 The contribution of compliance with the identified set of requirements of the standards, regulations and guidance to any specific claims or evidence components shall be stated.

1.1.3.15 If a claim argument evidence structure is followed, the basic assumptions and the necessary descriptions and interpretations, which support the claims, evidence components, arguments and relevant safety requirements (eg related to incident or accident scenarios, performance constraints etc), shall be precisely and fully included in the safety demonstration.

1.1.3.16 The system descriptions used to support the safety demonstration shall include accurate descriptions of the system design, the architecture of the system, hardware and software, system/environment interface, interactions, constraints and assumptions, and the system operation and maintenance.

1.1.3.17 The safety demonstration plan and safety demonstration (including all supporting evidence) shall be subject to configuration management, change control and impact analysis, and available to the regulator. The safety demonstration plan and safety demonstration shall be updated and maintained in a valid consistent state throughout the system lifecyle.

1.1.4 Recommended Practices

1.1.4.1 A key safety claim for a system (for example at the plant-system interface level) and its supporting evidence can be usefully organised in a multi-level structure. Such a structure is based on the fact that a claim for prevention or for mitigation of a hazard or of a threat at the plant-system interface level necessarily implies sub-claims of some or all of three different types:

a claims that the functional and/or non-functional requirement specifications of how the system has to deal with the hazard/threat are valid,

b claims that the system architecture and design correctly implement these specifications, and

c claims that the specifications remain valid and correctly implemented in operation and through maintenance interventions.

The supporting evidence for a safety claim can therefore be organised along the same structure. It can be decomposed into the evidence components necessary to support the various claims from which the safety claim is inferred.

* * *

1.2 System Classes, Function Categories and Graded Requirements for Software

1.2.1 Rationale

Software is a pervasive technology increasingly used in many different nuclear applications. Not all of this software has the same criticality level with respect to safety. Therefore not all of the software needs to be developed and assessed to the same degree of rigour.

Attention in design and assessment must be weighted to those parts of the system and to those technical issues that have the highest importance to safety.

This chapter discusses the assignment of categories to functions and of classes to systems, components and software in relation to their importance to safety. We also consider the assignment of "graded requirements" for the qualification of the software development process, of the software products of these processes, and on the amount of verification and validation necessary to reach the appropriate level of confidence that software is fit for purpose.

To ensure that proper attention is paid to the design, assessment, operation and maintenance of the systems important to safety, all systems, components and software at a nuclear facility should be assigned to different safety classes. Graded requirements may be advantageously used in order to balance the software qualification effort.

Thus levels of software relaxations must always conform and be compatible with levels of safety. The distinction between the safety categories applied to functions, classes applied to systems etc. and graded requirements for software qualification does not mean that they can be defined independently from one another. The distinction is intended to add some flexibility. Usually, there will be a one-to-one mapping between the safety categories applied to functions, classes applied to systems etc. and the graded requirements applied to software design, implementation and V&V.

1.2.1.1 System classification

This document focuses attention on computer-based systems used to implement safety functions (ie the functions of the highest safety criticality level); namely, those systems classified by the International Atomic Energy Agency as "safety systems". The task force found it convenient to work with the following three system classes (cf. IAEA NS-R-1 and SSG 39):

– safety systems
– safety related systems
– systems not important to safety.

The three system classes have been chosen for their simplicity, and for their adaptability to the different system class and functional category definitions in use.

The three-level system classification scheme serves the purpose of this document, which is principally focusing on software of the highest safety criticality (ie the software used in safety systems to implement safety functions). Software of lower criticality is addressed in chapter 1.11, and elsewhere when relaxations on software requirements appear clearly practical and recommendable.

The rather simple correspondence between the three system classes above, the IAEA system classes, and the IEC 61226 function categories can be explained as follows. Correspondence between the IEC function categories and the IAEA system classes can be approximately established by identifying the IAEA safety systems class with IEC 61226 category A, and coalescing categories B and C of the IEC and mapping them into IAEA safety related systems class. This is summarized in Table 1 below.

Table 1: Correspondence between categories, classes and graded requirements

Categories of functions		**Classes of systems**	**Examples of graded requirements for software**
IEC61226	This document	IAEA and this document	IEC and this document

A	Safety functions	Safety systems	– IEC 60880;
			– Table 2 and Table 3 (chapter 1.11) in this document;

B	Safety related functions	Safety related systems	– IEC 62138 chapter 6;
			– Table 2 and Table 3 (chapter 1.11) in this document;

C	Safety related functions	Safety related systems	– IEC 62138 chapter 5; – Table 2 and Table 3 (chapter 1.11) in this document.
Functions not important to safety		Systems not important to safety	----

1.2.1.2 Graded requirements

The justification for grading software qualification requirements is given by 1.2.3.10, namely the necessity to ensure a proper balance of the design and V&V resources in relation to their impact on safety. Different levels of software requirements are thus justified by the existence of different levels of safety importance, and there is indeed no sense in having the former without the latter. In this document, we use the term "graded requirements" to refer to these relaxations. Graded requirements should of course not be confused with the system and design requirements discussed in chapter 2.1.

This document does not attempt to define complete sets of software requirement relaxations. It does, however, identify relaxations in requirements that are found admissible or even recommendable, and under which conditions. These relaxations are obviously on software qualification requirements, never on safety functions.

It is reasonable to consider relaxations on software qualification requirements for the implementation of functions of lower safety category, although establishing such relaxations is not straightforward. The criteria used should be transparent and the relaxations should be justifiable.

Graded licensing requirements and relaxations for safety related software are discussed in chapter 1.11 where an example of classes and graded requirements is given (see recommended practice 1.11.4.2).

1.2.2 Issues Involved

1.2.2.1 Identification and assignment of system classes, function categories and graded requirements

Adequate criteria are needed to define relevant classes and graded requirements for software in relation to importance to safety. At plant level the plant is designed into systems to which safety and safety related functions are assigned. These systems are subdivided, in sufficient detail, into structures and components. An item that forms a clearly definable entity with respect to manufacturing, installation, operation and quality control may be regarded as one structure or component. Every structure and component is assigned to a system class or to the class "not important to safety". Identification of the different types of software and their roles in the system is needed to assign the system class and corresponding graded requirements.

In addition to the system itself, the support and maintenance software also need to be assigned to system classes with adequate graded requirements.

Software tools also have an impact on safety (see chapter 1.5 and the common positions therein). The safety importance of the tools depends on their usage: in particular, on whether the tool is used to directly generate online executed software or is used indirectly in a supporting role, or is used for V&V.

1.2.2.2 Criteria

Adequate criteria are needed to assign functions to relevant categories in relation to their importance to safety and, correspondingly, to identify adequate software graded requirements.

When developing the criteria for graded requirements for the qualification of software components it is not sufficient to assess the function in the accomplishment of which the component takes part or which it ensures. The impact of a safety or safety related function failure during the normal operation of the plant or during a transient or an accident must also be considered. In particular, attention must be paid to the possible emergence of an initiating event that could endanger nuclear safety and the prevention of the initiating event's consequences.

In addition, the following factors may have an impact on the graded requirements imposed on a software component and should be taken into account:

– availability of compensatory systems,
– possibilities for fault detection,
– time available for repair,
– repair possibilities,
– necessary actions before repair work,
– the reliability level that can possibly be granted to the component by the graded requirements of the corresponding class.

1.2.2.3 Regulatory approval

Safety categorisation, classification and a scheme of graded requirements for the qualification of software can be used as a basis to define the need for regulatory review and regulatory requirements.

1.2.3 Common Position

System classes

1.2.3.1 The importance to safety of a computer-based system and of its software is determined by the importance to safety of the functions it is required to perform. Categorisation of functions shall be based on an evaluation of the importance to safety of these functions. This functional importance to safety is evaluated by a plant safety analysis with respect to the safety objectives and the design safety principles applicable to the plant.

1.2.3.2 The consequences of the potential failure modes of the system shall also be evaluated. This evaluation is more difficult because software failures are hard to predict. Assessment of risk associated with software shall therefore be primarily determined by the potential consequences of the system malfunctioning, such as its possible failures of operation, misbehaviours in presence of faults, or spurious operations (system failure analysis).

1.2.3.3 The classes associated with the different types of software or components of the system must be defined in the safety demonstration plan at the beginning of the project.

1.2.3.4 For a smart sensor/actuator, the licensee shall define the safety class, which shall be the same as the class of the system in which the smart sensor/actuator is embedded unless justified otherwise.

1.2.3.5 Any software performing a safety function, or responsible within a safety system for detecting the failure of that system, is in the safety systems class.

1.2.3.6 As a general principle, a computer-based system important to safety and its software is a safety related system if its correct behaviour is not essential for the execution of safety functions (eg corresponding to IEC 61226 category A functions – see Table 1 above)

1.2.3.7 Common position 1.12.3.12 is applicable.

1.2.3.8 Any software generating and/or communicating information (for example, to load calibration data, parameter or threshold modified values) shall be of at least the same class as the systems that automatically process this information, unless

a the data produced is independently checked by a diverse method, or

b it is demonstrated (for example using software hazard analysis) that the processing software and the data cannot be corrupted by the generating/communicating software. (Common position 2.8.3.9 restates this position for the loading of calibration data.)

1.2.3.9 Any software processing information for classified displays or recorders shall be of at least the same class as the display or the recorder.

Graded requirements

1.2.3.10 The licensee shall have criteria for grading requirements for different types of software or components to ensure that proper attention is paid in the design, assessment, operation and maintenance of the systems important to safety.

1.2.3.11 Notwithstanding the existence of graded requirements, the licensee shall determine which development process attributes are required and the amount of verification and validation necessary to reach the appropriate level of confidence that the system is fit for purpose.

1.2.3.12 If requirement relaxations (graded requirements) are allowed for the qualification of certain specific types of software, these relaxations shall be justified and precisely defined in the safety demonstration plan, at the start of the project.

1.2.3.13 The graded requirements shall take into account developments in software engineering.

1.2.3.14 The definition of graded requirements can result from deterministic analysis, probabilistic analysis or from both of these methods. The adequacy of the graded requirements shall be justified, well documented and described in the safety demonstration plan.

1.2.3.15 The graded requirements shall coherently cover all software lifecycle phases from computer-based system requirement specifications to decommissioning of the software.

1.2.3.16 The graded requirements (defined in 1.2.1.2) that are applicable to safety systems include all "shall" requirements of IEC 60880 [5].

1.2.3.17 The risk of common cause failure shall remain a primary concern for multitrain/channel systems utilising software. Requirements on defence against common cause failure, on the quality of software specifications, and on failure modes, effects and consequence analysis (FMECA) of hardware and software shall not be relaxed without careful consideration of the consequences.

1.2.4 Recommended Practices

System classes

1.2.4.1 The functional criteria and the category definition procedures of IEC 61226 are recommended for the identification of the functions, and hence systems, that belong to the most critical class safety systems and that – with the exception of chapter 1.11 – are under the scope of this document. The examples of categories given in the annex A of this IEC standard are particularly informative and useful.

1.2.4.2 It may be reasonable to consider the existence of a back-up by manual intervention or by another item of equipment in the determination of the requirements applied to the design of a safety system.

Graded requirements

1.2.4.3 Relaxations and exceptions:

a For reasons already explained above, the definition of distinct requirements per class and of exceptions and relaxations for classes of lower importance to safety is difficult when software is involved.

b However, if reliability targets are used, exceptions to 1.2.3.16 above can be acceptable. For example, if reliability targets derived from a probability assessment are used, certain IEC 60880 requirements can be relaxed for software of class safety system for which the required reliability is proven to be low, ie for example in the range where the probability of failure on demand is between 10^{-1} and 10^{-2}. An example of such relaxations can be applicable to air-borne radiation detectors used as an ultimate LOCA (Loss of Coolant Accident) detection mechanism for containment isolation in the event of a failure of the protection system to detect this LOCA.

c Relaxations to the standard IEC 60880 and to its requirements for computer-based systems of the safety related systems class are dealt with in chapter 1.11.

1.2.4.4 In all cases, a system failure analysis should be performed to assess the consequences of the potential failures of the system. It is recommended that this failure analysis be performed on the system requirement specifications, rather than at code level, and as early as possible so as to come to an agreement between the regulator and the licensee on graded requirements.

1.2.4.5 Only design and V&V factors which affect indirectly the behaviour of programmed functions should be subject to possible relaxations: eg validation of development tools, SQA and V&V specific plans, independence of V&V team, documentation, and traceability of safety functions in specifications.

1.2.4.6 In agreement with 1.11.2.3 (safety related system class equipment), a software-based system intended to cover less frequent initiating event demands can be assigned adequate less stringent graded requirements if and only if it is justified by accident analysis.

* * *

1.3 Reference Standards

"Cuando creíamos que teníamos todas las respuestas,
de pronto, cambiaron todas las preguntas"

Mario Benedetti

1.3.1 Introduction

This document does not endorse industrial standards. Standards in use vary from one country and one project to another.

Many regulators do not "endorse" industrial standards in their whole. As a matter of fact, many national regulations ask for the standards used in a project design and implementation to be specified and their applicability justified, with potential deviations being evaluated and justified (see 1.1.3.13)

The applicable standards mentioned below in 1.3.2 are standards which are or will be used by members of the Task Force as part of their assessment of the adequacy of safety systems.

A system is regarded as compliant with an applicable standard when it meets all the requirements of that standard on a clause-by-clause basis, sufficient evidence being provided to support the claim of compliance with each clause.

However, in software development, compliance with an applicable standard may be taken to mean that the software is compliant with a major part of the standard. Any non-compliance will then need to be justified as providing an acceptable alternative to the related clause in the standard.

1.3.2 Reference standards and guidelines for safety systems in the member states represented in the task force

Belgium

- ANSI/IEEE-ANS-7-4.3.2, 2003 revision: “IEEE criteria for Digital Computers in Safety Systems of Nuclear power generating Stations” as endorsed by revision 2 of US NRC Regulatory Guide 1.152: “Criteria for programmable digital computer system software in safety related systems of NPPs”.
- IEC 60880, Edition 2 (2006) “Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions”.
- IEC 61226, Edition 2 (2005) “Nuclear power plants – Instrumentation and control systems important to safety – Classification”.
- IEC 61513, Edition 2 (2011) “Nuclear power plants – Instrumentation and control important to safety – General requirements for systems”.
- IAEA NS-G-1.1 “Software for Computer-Based Systems Important to Safety in Nuclear Power Plants”.
- IAEA SSG-39 “Design of Instrumentation and Control Systems for Nuclear Power Plants”.

These are the standards applicable in most cases, but are not mandated by the regulator. Standards are selected on a case-by-case basis and the selection has to be justified in the safety demonstration.

Canada

- REGDOC-2.5.2 “Design of Reactor Facilities: Nuclear Power Plants”, CNSC.
- REGDOC-1.1.2 “Licence Application Guide, Licence to Construct a Reactor Facility”, CNSC.
- IAEA SSG-39, “Design of Instrumentation and Control Systems for Nuclear Power Plants”.
- IEC 60880 “Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions”.
- IEC 61513 “Nuclear power plants – Instrumentation and control important to safety – General requirements for systems”.
- IEC 61226 “Nuclear power plants – Instrumentation, control and electrical power systems important to safety – Categorization of functions and classification of systems”.
- IEEE Std. 7-4.3.2 “IEEE Standard Criteria for Programmable Digital Devices in Safety Systems of Nuclear Power Generating Stations”.

- IEEE 603, "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations".
- CSA Standard N290.14 "Qualification of digital hardware and software for use in instrumentation and control applications in nuclear power plants".

China

- HAF 003 "Code on Safety of Nuclear Power Plant Quality Assurance".
- HAF 102 "Code on Safety of Nuclear Power Plant Design".
- HAD 102/16 "Software for Computer Based Systems Important to Safety for Nuclear Power Plants".
- IAEA SSG-39 "Design of Instrumentation and Control System for Nuclear Power Plants".
- GB/T4083 "Nuclear Reactor Protection System Safety Guidelines".
- GB/T5204 "Periodic Tests of Nuclear Power Plant Safety Systems and Monitoring".
- GB/T13284 "Nuclear Power Plant Safety Systems: Design".
- GB/T13626 "Single Failure Criterion Used for Nuclear Power Plant Safety Class of Electrical Systems".
- GB/T13629 "Applicable Criteria for Digital Computers Used in Nuclear Power Plants".
- GB/T13286 "Independence of Nuclear Power Plants Safety Equipment and Circuits".

Finland

- Guide YVL A.3 "Leadership and management for safety", 15.3.2019.
- Guide YVL A.5 "Construction and commissioning of a nuclear facility", 15.3.2019.
- Guide YVL A.12 "Information security management of a nuclear facility", 12.2.2021.
- Guide YVL B.1 "Safety design of nuclear power plants", 15.6.2019.
- Guide YVL B.2 "Classification of systems, structures and components of a nuclear facility", 15.6.2019.
- Guide YVL E.7 "Electrical and I&C equipment of a nuclear facility", 15.3.2019.
- IEC 60880 "Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions".

Germany

- Safety Requirements for Nuclear Power Plants, amended 25 February 2022.
- Interpretations of the "Safety Requirements for Nuclear Power Plants", amended 3 March 2015.
- RSK Guidelines, Chapter 7.6, amended August 1996.
- DIN EN 61513 (2013).
- DIN EN IEC 61226 (2022).
- DIN EN IEC 62138 (2020).
- DIN IEC 60880 (2010).
- KTA. 3503, November 2015, Type testing of electrical modules for the safety related instrumentation and control system.
- VDI/VDE 3527 Design criteria serving to ensure independence of I & C safety functions, July 2002.

Spain

- UNESA CEN-6 "Guideline for the Implementation of Digital Systems at Nuclear Power Plants" (In Spanish), 2002.
- UNE 73-404-91. "Quality Assurance in Computer Systems Applied to Nuclear Installations " (In Spanish), 1991.
- CSN 10.9 Safety Guide. "Quality Asssurance of Computer Applications Related to Nuclear Installations" (In Spanish), 1998.
- ANSI/IEEE-ANS 7-4.3.2-2004. "IEEE Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations", as endorsed by USNRC Regulatory Guide 1.152 "Criteria for Programmable Digital Computer System Software in Safety Systems of NPPs".
- RSK Guidelines, Chapter 7.6, amended August 1996.
- CSN Safety Instruction IS-21 on the requirements applicable to modifications at nuclear power plants, 2009.
- CSN Safety Instruction IS-26 on basic nuclear safety requirements applicable to nuclear installations, 2010.
- CSN Safety Instruction IS-27 on general nuclear power plant criteria, Revision 1, 2017.

In addition to the above mentioned standards, applicable standards in the facility vendor's country of origin are mandatory, once they are included in the facility license basis.

South Korea

- IEEE Std. 603 “IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations”.
- IEEE Std. 7-4.3.2 “IEEE Standard Criteria for Programmable Digital Devices in Safety Systems of Nuclear Power Generating Stations”.
- IEEE Std. 1012-2003 “IEEE Standard for Software Verification and Validation”.

Sweden

- IAEA SSG-39 “Design of Instrumentation and Control Systems for Nuclear Power Plants”.
- IEC 60880 “Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions”.
- IEC 61226 “Nuclear power plants – Instrumentation, control and electrical power systems important to safety – Categorization of functions and classification of systems”.
- IEC 61513 “Nuclear power plants – Instrumentation and control important to safety – General requirements for systems”.

The above standards are applicable in most cases, but are not mandated by the regulator. Standards are selected on a case-by-case basis and this selection shall be fully justified in the safety demonstration.

UK

- NS-TAST-GD-046 ONR Technical Assessment Guide “Computer Based Safety Systems”.
- IAEA SSG-39 “Design of Instrumentation and Control Systems for Nuclear Power Plants”.
- IEC 60880 “Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions”.
- IEC 61226 “Nuclear power plants – Instrumentation, control and electrical power systems important to safety – Categorization of functions and classification of systems”.
- IEC 61513 “Nuclear Power Plants – Instrumentation and control important to safety – General requirements for systems”.

1.3.3 Related Standards, Guidelines and Technical Documents

Guides and standards are continuously in evolution. At the time of writing this document, the International Atomic Energy Agency (IAEA) and the IEC have issued a number of relevant reference documents.

In 2002 and 2000, the International Atomic Energy Agency (IAEA) published the safety guides entitled "Instrumentation and control systems important to safety in nuclear power plants" (NS-G-1.3) and "Software for Computer-Based Systems Important to Safety in Nuclear Power Plants" (NS-G-1.1), both giving guidance on the evidence to be provided at each phase of the software lifecycle to demonstrate the safety of the computer-based system. The intention is to assist the licensing process by the provision of evidence. In 2016, these two safety guides were revised and merged in a new IAEA specific safety guide "Design of Instrumentation and Control Systems for Nuclear Power Plants" (SSG-39).

In addition, the IAEA produced and published technical documents concerning distinct safety and licensing aspects of digital systems for nuclear installations, such as "Safety Assessment of Computerized Protection Systems" (IAEA-TECDOC-780, 1994), "ESRS Guidelines for Software Safety Reviews: Reference document for the organization and conduct of Engineering Safety Review Services (ESRS) on software important to safety in nuclear power plants" (IAEA Services Series No. 6, 2000), "Harmonization of the licensing process for digital instrumentation and control systems in nuclear power plants" (IAEA-TECDOC-1327, 2002), "Solutions for Cost Effective Assessment of Software Based Instrumentation and Control Systems in Nuclear Power Plants" (IAEA TECDOC Series No. 1328, 2002), and the Technical Report 397 "Quality Assurance for Software important to safety which addresses graded requirements for software."

The IEC 61513 standard "Instrumentation and control for systems important to safety – General requirements for systems" (2001) covers the system aspects of computer-based I&C. It acts as a link document for a number of standards, including IEC 60880 "Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer based systems performing category A functions" (second edition 2006), IEC 60987 "Nuclear power plants – Instrumentation and control important to safety – Hardware design requirements for computer-based systems" (second edition 2007), and IEC 61226 "Classification of instrumentation and control functions" (second edition 2005). The IEC 61513 standard contains the top-level requirements on system functions, architecture and I&C system design which are of particular importance to the software specification. The 3rd edition was published in 2011 in order to take into account new developments in software engineering.

The second edition of IEC 61226 contains criteria to rank I&C functions in accordance with their significance to safety. It comprises main requirements on I&C functions, attached to three safety categories (A, B, and C). The third edition (2009) defines the non-hazardous stable state and specifies the criteria to assign safety functions to categories in more detail.

The second edition of IEC 60880 (2006) comprises both the first edition issued 1986 and its supplement issued 2000 with updated requirements covering the whole software lifecycle (described in IEC 61513). The new edition also contains informal annexes on different special software qualification aspects such as defence against common cause failures, tools for software development and qualification, as well as requirements on pre-existing software.

The IEC 62138 standard "Software aspects for computer-based systems performing category B or C functions" (2004) contains graded requirements for software implementing category B and C functions (in the terminology of IEC 61226).

The IEC 62340 standard "Nuclear power plants – Instrumentation and control systems important to safety – Requirements for coping with common cause failure" (2007) contains, among others, distinct requirements on software, eg linked to software robustness, tolerance against postulated latent failure software errors, as well as software maintenance and security.

The second edition of the IEC 60987 standard contains reduced system and project level requirements. It also contains additional requirements on topics such as pre-developed hardware equipment from a generic platform, new function-based statements on the single failure criteria, as well as guidance on the use of advanced hardware designs with embedded micro-codes.

In the mid 1980's, IEC set up two working groups to produce a standard on programmable electronic systems. This international standard is now known as IEC 61508 and is entitled "Functional safety of electrical/electronic/programmable electronic safety-related systems". The standard covers the determination of safety integrity levels (SILs) and the means of achieving these levels in terms of requirements on I&C systems, hardware and software. The standard contains seven parts covering, for example, management, hardware and software requirements. Based on the SILs, part 3 contains graded requirements on the software qualification.

IEC regards IEC 61508 as a generic standard with the intention that, using IEC 61508 as a basis, the various industry sectors will produce their own specific standards. This has been recognised by the IEC working group producing the nuclear standards (see preceding paragraphs in this section) and is being accommodated. The second editions of IEC 61508 Part 1 "General Requirements" and Part 3 "Software Requirements" were published in 2010.

More information on the current work and publications of the IAEA and IEC can be found on their websites (www.iaea.org/resources/safety-standards and www.iec.ch/publications).

*   *   *

1.4 Pre-existing Software (PSW)

1.4.1 Rationale

This document sets out common positions for developing new software throughout its lifecycle, including for example specifying, designing and coding. If the software has already been developed, it may not be possible to demonstrate that the requirements of every common position have been met because not all the requirements were explicit objectives during development. The purpose of this chapter is to set out an alternative set of common positions applicable to pre-existing software (PSW). The common positions of this chapter do not apply to software under development. Furthermore, in many cases, creating a safety demonstration that meets the common positions of this chapter will be more difficult than for new software that meets the other common positions.

The inclusion of appropriately justified PSW components into a computer-based system can be beneficial for productivity and, if applied correctly, may still give adequate confidence that the system is safe. Adequate confidence will usually depend on evidence that the PSW component has been used in many applications, and its operating experience can be taken into account because it is sufficiently assessable and representative. Reusable software components that are adequately specified, verified and proven for safety critical applications in other industries may be available and reusable in the nuclear industry. Licensees may wish to make use of such software. Its previous assessment and contributing evidence such as independent verification records and documented operating experience are helpful but are not sufficient.

This guidance on pre-existing software components is intended to enable their use in a safety-critical system without complete re-engineering of the PSW components or compromising system safety. Examples of PSW components are: real-time operating system services; device drivers or interfaces to hardware devices; mathematical libraries; timing monitors; and application logic subroutines.

1.4.2 Issues involved

1.4.2.1 The functional behaviour and non-functional qualities of the PSW is often not clearly specified and documented.

1.4.2.2 It is doubtful that evidence will be available to demonstrate that the PSW has been developed and produced in accordance with a defined safety lifecycle such as outlined in IEC 60880.

1.4.2.3 The documentation and data on operational experience of the PSW are often not adequate enough to provide the evidence which would be required to compensate for the lack of knowledge on the PSW product and on its development process.

1.4.2.4 As a result of issues 1.4.2.1, 1.4.2.2 and 1.4.2.3, acceptance criteria and procedures of investigation for demonstrating fitness for purpose of PSW for a specific application may be difficult to put in place.

1.4.2.5 The operational experience related to the PSW may not be in exact correspondence with that of the intended application. Therefore, software paths of unknown quality may be invoked by the application.

1.4.3 Common Position

1.4.3.1 The functions that have to be performed by the PSW components, and the other properties allocated to them, shall be clearly and unambiguously specified.

1.4.3.2 Evidence shall be provided that the requirement specifications allocated to the PSW comply with the safety requirements of the system, and may include the results of a system hazard and failure analysis. Special attention shall be paid to possible side-effects and to failures that may occur at the interfaces between the PSW and the user and/or other software components.

1.4.3.3 The PSW components to be used shall be clearly identified including their code version(s).

1.4.3.4 The interfaces through which the user or other software invokes PSW items shall be clearly identified and thoroughly validated. Evidence shall be given that there is no other means of accessing the PSW module, even inadvertently.

1.4.3.5 The PSW shall have been developed and shall be maintained according to good software engineering practice and SQA standards appropriate to its intended use.

1.4.3.6 For safety systems, the PSW shall be subjected to the same assessment (analysis and review) of the final product (not of the production process) as new software developed for the application. If necessary, reverse engineering shall be performed to enable the full specification of the PSW to be evaluated.

1.4.3.7 If modifications of PSW components are necessary, the design documentation and the source code of the PSW shall be available.

1.4.3.8 The information required to evaluate the quality of the PSW product and of its assessment and development processes shall be available; this information shall be sufficient to assess the PSW to the required level of quality.

1.4.3.9 For acceptance the following actions shall be taken:

a Verify that the functions performed by the PSW meet all of the requirements expressed in the safety system requirement specifications and in other applicable software specifications.

b Verify that the PSW functions that are not required by the safety system requirement specifications cannot be invoked and adversely affect the required functions, for example through erroneous inputs, interruptions, and misuses.

c Verify that, if PSW functions are omitted, this omission has no adverse impact on safety.

d Perform a compliance analysis of the PSW design against the applicable standards requirements (eg IEC 60880).

e The PSW functions intended for use shall be validated by testing. The tests may include tests performed by the vendor.

1.4.3.10 If credit is given to feedback experience in the licensing process, sufficient information on operational history and failure rates shall be available. Feedback experience shall be properly evaluated on the basis of an analysis of the operating time, defect reports and release history of systems in operation. This feedback experience shall also be based on use of the PSW under evaluation in identical operational profiles. This operating experience shall be of the version to be used except if an adequate impact analysis shows that previous experience of unchanged parts of the PSW is still valid because these parts have been unaffected by later releases.

1.4.3.11 Defects that are found during the validation of the PSW shall be analysed and taken into account in the acceptance procedure.

1.4.3.12 Depending on the type of equipment containing the PSW, chapter 1.15 on smart sensors and actuators and chapter 1.16 on programmable logic devices will also apply.

1.4.4 Recommended Practices

1.4.4.1 Operational experience may be regarded as statistically based evidence complementary to validation, or to the verification of system software (operating systems, communication protocols, standard functions).

1.4.4.2 Data for the evaluation of the credit that can be given to feedback experience should be collected, in terms of site information and operational profiles, demand rate and operating time, error reports and release history.

Site information and operational profile data should include:

Configuration of the PSW;

- ◦ Functions used;
- ◦ Types and characteristics of input signals, including the ranges and, if needed, rates of change;
- ◦ User interfaces;
- ◦ Number of systems.

Demand rate and operating time data should include:

- ◦ Elapsed time since first start-up;
- ◦ Elapsed time since last release of the PSW;
- ◦ Elapsed time since last severe error (if any);
- ◦ Elapsed time since last error report (if any);
- ◦ Types and number of demands exercised on the PSW.

Error reports should include:

- ◦ Descriptions and dates of errors, severity;
- ◦ Descriptions of fixes.

Release history should include:

- ◦ Dates and identifications of releases;
- ◦ Descriptions of faults fixed, functional modifications or extensions;
- ◦ Pending problems.

The data referenced above should be recorded along with the identification of the release of the PSW and its associated configuration.

1.4.4.3 The PSW functions should be prevented from being used in ways that are different from those which have been specified and tested, through the implementation of pre-conditions, locking mechanisms or other protections.

*   *   *

1.5 Tools

1.5.1 Rationale

The use of appropriate software tools can increase software product quality by improving the reliability of the software development process and the feasibility of establishing the product's fitness for purpose. The use of automated tools reduces the amount of clerical effort required, and hence the risk of introducing errors in the development process. Tools can automatically check for adherence to rules of construction and standards, generate proper records and consistent documentation in standard formats, and support change control. This assures process reliability which is important for achieving product quality.

Different types of tools can be used that affect the correctness of the software:

- Requirements engineering tools used for: eliciting, capturing, specifying and modelling (unambiguously representing) requirements - including design constraints and assumptions; generating test cases; and assisting traceability.
- Design tools for transforming requirement specifications and constraints into architectural and detailed design models.
- Transformation tools such as code generators and compilers, that transform a work product at one level of abstraction into another, usually lower, level of abstraction.
- Verification and validation tools such as test environments, computer equipment for periodic testing, static code analysers, test coverage monitors, theorem proving assistants and simulators.
- Tools to check for adherence to rules of construction and standards.
- Tools to generate proper records and consistent documentation in standard formats.
- Tools to support change control, version control and configuration management.
- Service tools used to produce, modify, display and output software objects in design and assessment. They include graphic editors, text editors, text formatters, communication drivers, print spoolers, window arrangement software, etc.
- Infrastructure tools such as development operating systems, public tool interfaces and version control tools. These tools are potential sources of common errors, as they may interfere with other tools or corrupt intermediate software products.

1.5.2 Issues Involved

1.5.2.1 The production of software may be adversely affected by the use of tools in several ways. For example, transformation tools may introduce faults by producing corrupted outputs; and verification tools may fail to reveal faults that are already present, or may be unable to reveal certain types of faults. In addition, faults introduced by tools may lead to common cause failures.

1.5.2.2 Those using the tool require competencies that may be new and cognitive capabilities that may not be well-understood. The assumptions built into the tool or limitations on its application, eg its valid input range or intended mode of use, may not be explicit or even determined. For example, a verification tool may mask errors it is capable of finding by providing only a representative example to show a particular test has failed, and not reporting similar errors until the example error has been resolved and the verification repeated.

1.5.2.3 Often, work products from different lifecyle phases are in different languages and tool environments, and their semantics are not completely defined. Tools transforming these work products will then rely on assumptions about the source and target languages. With multiple transformation stages, the uncertainties combine in unknown ways.

1.5.2.4 Where tools are used for single programs or software components, the integration of these components sometimes requires manual changes to the tool outputs. Verification of these changes may be outside of the normal tool output verification process. Furthermore, manual changes and their verification may require knowledge about the tool's operation or other outputs that is hard to acquire.

1.5.2.5 It is not easy to determine the required level of tool qualification. Essentially, it will depend on the safety class of the system for which the tool is being used, on the consequence of an error in the tool, on the probability that the fault introduced by the tool will cause a safety significant error in the software being developed, and on the possibilities of the error remaining undetected by other tools or verification steps. Qualification standards and procedures are not always mature. Manual verification of the tool output is difficult and introduces additional uncertainties.

1.5.2.6 In the current state of the art in the certification and qualification of tools, complementary evidence, such as independent checks of the tool outputs, is also necessary.

1.5.3 Common Position

1.5.3.1 The qualification requirements for all tools used for the development, operation and maintenance of the software shall be determined and the following requirements of this section applied accordingly (see 1.5.1 for example tools). The available information shall be sufficient to assess the appropriateness of the tools.

1.5.3.2 The qualification requirements for an individual tool will be a function of its impact on the target software, and of how other tools or processes may or may not detect faults introduced by that tool. The combined set of tools shall be qualified to a level that preserves the required properties of the target software.

1.5.3.3 The user shall have sufficient competence to understand the built-in assumptions and limitations of the tool and the conditions and operational process required for its effective use (see 1.5.2.2). This understanding is necessary to assess and justify the appropriateness of the tool for a particular application and mode of use (particularly if the tool is based on sophisticated principles and methods, eg model checking).

1.5.3.4 The manufacturer's quality assurance documentation of tools used in the development of safety system software shall be available to the regulator.

1.5.3.5 If a different version of the tool on which the safety demonstration depends is considered for use, then the changes to the tool shall be analysed for their impact on the safety demonstration.

1.5.3.6 An analysis shall be applied to the transformation tools used in the software development process to identify the nature of the faults they can introduce in the target software, and their consequences.

1.5.3.7 For safety system software, only tools that are validated against the requirements defined in 1.5.3.2 here above shall be used. Safety system software produced by a transformation tool shall be subjected to verification, and to validation on the target system.

1.5.3.8 The use of a transformation tool for safety system software without further review of its output shall not be accepted unless special justification is provided. In particular, this justification shall include evidence

a that only input data that is syntactically valid can be accepted by the tool for transformation, and

b that the tool preserves the semantics of valid input to the last output stage of the transformation.

1.5.3.9 The vendor of a tool shall maintain and update the tool experience feedback and inform users of all anomalies discovered (including those discovered by users).

1.5.3.10 The strategy for tool use and validation shall be defined in the software quality plan (see 1.7).

1.5.3.11 The programming techniques which are used in combination with transformation tools shall comply with the relevant requirements of chapter 2.4.

1.5.4 Recommended Practices

1.5.4.1 To validate a tool, the following possibilities are recommended:

a the output of a tool can be verified by inspection;

b the behaviour of the output of the tool can be verified by running the output on the target system or on a simulator of this system;

c the tool itself can be certified as processing correctly all valid statements and combinations thereof contained in the definition of a programming language;

d the tool can be validated by appropriate profiles and a sufficient amount of previous operational experience.

1.5.4.2 When formal methods are applied manually, they require the involvement of well-trained humans. In addition, application of these techniques is likely to be affected by mistakes. Therefore these formal methods should be supported by tools.

1.5.4.3 The tools should be incorporated into an integrated project support environment to ensure proper control and consistency. This environment should include suitable data dictionaries which check for inconsistencies in the data base, ensure the correctness of values of items entered and enable data to be interrogated. In addition, it should support the exporting and importing of data to and from other sources.

1.5.4.4 A software tool should be independent from the operating system so as to minimise the consequences of having to use new versions of this system.

1.5.4.5 Tools of reasonable maturity should be used to support the process of specification and design, namely:

a tools for editing and typesetting;

b tools for syntax-checking, data type-checking and consistency-checking;

c tools to prove properties of specifications and discharge proof obligations in refining specifications;

d tools to animate formal specifications;

e tools to derive usable software components automatically from formal specifications;

f configuration control tools to relate specifications to development and to keep track of the level of verification of each step;

g static and dynamic analysis tools to complement code inspections and walkthrough techniques.

1.5.4.6 Reverse engineering tools should be used to validate the output of transformation tools or the output of manual programming processes.

*   *   *

1.6 Organisational Requirements

1.6.1 Rationale

Well structured and effective organisations, ie licensees, their subcontractors and suppliers, are regarded as necessary in the management of all aspects of the safety system lifecycle to reduce the faults introduced into a system, and to ensure that those faults which are found are handled properly. Throughout the lifecycle, clear delineation of responsibilities and effective lines of communication plus proper recording of decisions will minimise the risk of incorrect safety system behaviour. An additional important aspect of an effective management system is the development of a safety culture which, at all levels within the organisation, emphasises safety within the organisation. By the use of managerial supervisory and individual practices and constraints, the safety culture sustains attention to safety through an awareness of the risk posed by the plant and on the potential consequences of incorrect actions.

1.6.2 Issues Involved

Safety culture

1.6.2.1 If staff is not made aware of the risks posed by the plant and the potential consequences of those risks then project pressures, inattention to detail and a lack of rigour may result in the safety system not being fit for purpose. More significantly, this unfitness may not be revealed. However, it has to be recognised that although a safety culture is necessary, it is not regarded, on its own, as sufficient for developing a safety system that is fit for purpose.

Delineation of responsibility

1.6.2.2 If responsibilities are not clearly defined, then potentially unsafe activities or system aspects may not be given their due attention resulting in an increased plant risk.

Staffing levels

1.6.2.3 Inadequate staffing levels may result in a drastic deterioration in the safety culture due to staff being overworked and demoralised. Thus, safety issues may not be explored properly, and inadequate designs may result.

Staff competencies

1.6.2.4 If staff do not have the appropriate level of training and experience for the tasks they are undertaking within the system lifecycle, then it is likely that errors will be introduced. In addition, errors may not be detected, and the design may not have the fault tolerance required to prevent failure of the computer-based safety system.

Project pressures

1.6.2.5 Project pressures, both financial and temporal, might mean that the safety provisions are inadequate, or that there is insufficient attention to detail, resulting in a safety system that is not fit for purpose.

1.6.3 Common Position

1.6.3.1 Only companies with demonstrable capabilities and competence in the development of safety systems shall be used in all stages of the development lifecycle.

1.6.3.2 The licensee, its suppliers and subcontractors shall each provide evidence of the following:

a Safety culture:
A written safety policy shall be available demonstrating a commitment to a safety culture which enhances and supports the safety actions and interactions of all managers, personnel and organisations involved in the safety activities relating to the production of the safety system.

b Delineation of responsibility:
The responsibilities of, and relationships between, all staff and organisations involved in all aspects of the safety system lifecycle shall be documented and clearly understood by all concerned.

c Staffing levels:
It shall be demonstrated, for example by means of appropriate work planning techniques, that levels of staffing are adequate to ensure that the safety system's fitness for purpose is not put in jeopardy through lack of staff effort.

d Staff competencies:
Staff shall have the appropriate level of training and experience for the tasks they are undertaking within the safety system lifecycle. Suitable evidence shall be available for inspection that demonstrates that safety proficiency is achieved and maintained.

1.6.3.3 Before a system is taken into operation, the licensee should document the methods to be employed (change control, configuration management, maintenance, data entry etc.) to ensure that the required level of integrity of the safety system will be maintained throughout its operational life.

1.6.3.4 Project programme recovery strategies shall only be undertaken provided the requirements of the safety system remain achievable.

1.6.4 Recommended Practices

1.6.4.1 Adequate procedures should be in place for controlling contract documentation between customers and suppliers to ensure that all specification and safety-critical contractual requirements are properly controlled.

1.6.4.2 The licensee should provide evidence that he is monitoring adequately the safety aspects of any contract involving a safety system.

1.6.4.3 Project/Operational Pressures:
A licensee safety department staffed by personnel with the appropriate computer competencies and independent from the project team (for plant under construction or modification) or from the operating department, should be appointed to reduce the probability (through regular monitoring of the activities associated with the safety system) that project or operational pressures will jeopardise the safety systems fitness for purpose. This department should ensure that the appropriate standards, procedures and personnel are used at all stages of the software lifecycle to meet the safety requirements of the computer-based system. This department should be independent from all project and operational staff such as designers, implementers, verifiers, validaters, and maintainers.

1.6.4.4 Regular progress reports from the licensee safety department or from an appropriate body within the licensee's organisation should be available for inspection throughout the project lifecycle.

*   *   *

1.7 Software Quality Assurance Programme and Plan

1.7.1 Rationale

It is widely accepted that the quality of software cannot be guaranteed by inspection or by testing of the product alone. In general, the application of a good quality assurance system provides essential evidence of the quality of the end product. Software is no different: confidence in the safety of the final software-based system increases with the use of quality assurance techniques. This is particularly true for any non-trivial software product since it cannot be exhaustively tested and interpolation between test inputs is not possible due to the digital nature of the system (a situation that does not pertain to analogue systems).

Additionally, software quality assurance (SQA) aids the development of a convincing safety demonstration because it is a structured and disciplined method of ensuring that there is sufficient auditable evidence.

1.7.2 Issues Involved

1.7.2.1 It is recognised that the reliability of a computer-based safety system cannot be demonstrated by testing. Therefore, the demonstration of safety has to depend to some degree on the quality of the processes involved. The major strength of a SQA plan and programme is that it ensures the procedures and controls are in place to govern and monitor the system lifecycle thus providing an auditable record of the production process and of the operational life of the system.

1.7.2.2 The SQA programme and plan do not ensure that the design solution is the most appropriate unless there are specific requirements in the specification for design reviews by appropriate personnel.

1.7.2.3 SQA audits, because of their sampling nature do not always detect non-compliance with procedures and standards.

1.7.2.4 In the majority of cases, the more complex the system and the larger the project, the more important SQA becomes, because of the ensuing increase in the functional and organisational complexity.

1.7.3 Common Position

1.7.3.1 The Software quality assurance shall cover all aspects of the system lifecycle.

1.7.3.2 A reviewable quality assurance plan and programme for the software aspects of the safety system, covering all stages of specification, design, manufacture, V&V, installation, commissioning, operation and maintenance shall be produced at the beginning of the project. This shall form part of the evidence that the project is being properly controlled. SQA will ensure that the functional behaviour is traceable throughout the system lifecycle and that the behaviour has been adequately demonstrated through testing to the required standards.

1.7.3.3 The properties addressed in the specification of the non-functional requirements (see 1.1.1 and 2.1.3.2 b) shall be considered in the SQA programme and plan through a specific statement on how conformance with the requirements is to be established.

1.7.3.4 The SQA programme and plan shall include a description of the organisational structure, staff competencies and adequate procedures covering activities such as defect reporting and corrective action, computer media control (production and storage), testing, supplier control. Record keeping and documentation shall be available to demonstrate adequate control of lifecycle activities.

1.7.3.5 The SQA programme and plan shall make reference to standards and written procedures to control the activities described in 1.6.3.3. These standards and procedures shall also act as the focus for periodic, independent audits to confirm compliance (this is considered to be an important aspect, since well documented audits with corrective action notices fully resolved provide a valuable source of evidence on the adequacy of the lifecycle processes).

1.7.3.6 The SQA programme and plan shall be accessible to the regulator for review and acceptance, if required.

1.7.3.7 There shall be SQA audits of compliance with the agreed SQA plan. A justification shall be provided for the frequency of SQA audits to be performed by the specialist software auditors.

1.7.3.8 The performance of appropriate reviews shall be required by the SQA process to ensure that due consideration is given to the completeness and correctness of documents.

1.7.3.9 SQA shall ensure that the tests are systematically defined and documented to demonstrate that adequate test coverage has been achieved and that the tests are traceable to the system requirement specifications. When faults are found, SQA shall ensure that procedures are in place for addressing those faults, and also for ensuring that corrections are properly implemented.

1.7.4 Recommended Practices

1.7.4.1 It is highly recommended that an agreement be reached at the beginning of the project between the licensee and the regulator on the SQA programme and plan.

1.7.4.2 Whenever possible, audits should be performed using aids and tools which assist in identifying compliance with standards and procedures, and record anomalies and corrective action notices.

1.7.4.3 The standard IEEE Std 730.1-2002, “Standard for Software Quality Assurance Plans” is recommended.

* * *

1.8 Security

1.8.1 Rationale

Security of any system important to safety is necessary to ensure that its performance will not be degraded from unwanted intrusion. This chapter focusses on the practices necessary to avoid degradation of the computer-based system safety functions from unwanted intrusion through software. Degradation can result from insecure design, implementation, application or maintenance of the safety system. The many possible outcomes of degradation include the output of a function not provided when needed, provided when not needed, or giving an incorrect value. Or the output could be provided at the wrong time, or out of sequence, or could interfere with another safety function, or exhibit Byzantine behaviour.

Security depends on the prevention of unauthorised access or influence. Traditionally, access has been restricted by physical means, such as locked doors and key interlock systems. But this is insufficient for computer-based systems because the associated software (including application specific system software, operational and runtime software, programmed logic and IP cores) has unique vulnerabilities. Associated interconnections, such as networks, may facilitate unintended or malicious interactions, and increase the potential for malicious intrusion and interference.

Degradation of the safety function can also result from a lack of security in systems, devices, logic, data, or any other resource, on which the computer-based system depends. Such resources are also within the scope of this chapter. This includes heating, ventilation and air conditioning systems, electrical supply control and protection systems, design and build documentation (including configuration settings and as-built drawings), maintenance instructions, maintenance scheduling systems, fuel burn-up calculations and other calculations (on or off-line) used to support operational safety, and computer-based systems for emergency arrangements and dosimetry recording.

Design decisions to avoid or reduce dependence on software, such as use of hardwired logic, or relatively simple monitoring systems that ensures a safe state is reached, can substantially increase security but are outside the scope of this document.

Security is an extensive and evolving topic. This chapter sets out general positions only, on policy, programs and procedures to establish and maintain security. Further guidance is available in relevant IAEA, IEC and national guidance and standards [25-28].

1.8.2 Issues Involved

1.8.2.1 Traditional physical access restriction is not sufficient for prevention of unauthorised access, because digital technology for computing, storage and communications allows new ways to access information and to intervene in its production. Potential unauthorised actions include:

- disclosure or access of information in order to degrade a safety function (confidentiality),
- modification of information in order to degrade a safety function (integrity),
- with-holding resource (eg information or capacity) in order to prevent the performance of a safety function when required (availability).

1.8.2.2 Digital computation and communication in secondary or supporting functions increases the potential for the degradation of safety functions. Examples include loading parameter values, loading calibration data, testing equipment, maintenance scheduling, on or off-line calculations to support operational safety (such as fuel burn-up), removable data transfer, and computer-based systems for emergency arrangements and dosimetry recording.

1.8.2.3 Means, mechanisms and agents for malicious activities are proliferating (including viruses, software logic bombs, software time bombs, Trojan horses, and online hacking tools).

1.8.2.4 Intrusions can occur through supply chains, which are getting longer, more complex and more difficult to control. Information is transferred between multiple environments with many interfaces, covering for example how the information is presented and stored, and using different tools, procedures (documented and undocumented) and policies. Each step and interface provides a potential for intrusion.

1.8.2.5 Approaches for safety and security are not always well integrated, with the potential for conflict. This includes goals, scales for ranking significance, objectives, policies and requirements. A change made for security might have an unintended effect on safety. For example, information assumed to be accessible that is relevant to decisions necessary to maintain safety may become restricted after it is identified as posing a security risk. Traditionally, software safety specialists have a background in instrumentation and control software, and software security specialists have a background in information technology. Different knowledge, background, approaches and methods can hinder effective communication and co-operation, during development and onsite.

1.8.2.6 Previously established security requirements may be invalidated (or become incomplete) by changes in technology, capability and intent. These changes can be rapid and have a cumulative impact.

1.8.2.7 Changes in the operational environment introduce the potential for previously unanalysed threats, invalidating the basis of previous analyses. These changes might not be the subject of appropriate impact analysis procedures.

1.8.2.8 Security vulnerabilities introduced through software are not as easily detected as those only involving traditional hardwired systems. They may remain unrevealed during operation for a long time. Examples include: hidden, unanalysed or unused code; defective or unanalysable code or design; or defective specification.

1.8.2.9 Information stored in a digital system may be more vulnerable than that stored by other means. Furthermore utilising the inherent advantages of digital information storage, namely greater capacity and ease of communication, increases the potential for degradation or misuse.

1.8.2.10 Modes other than steady state operation (such as configuration, calibration, functional testing, replacement, startup, shutdown), and transitions between them, may introduce unanticipated intrusion pathways with inadequate access controls.

1.8.2.11 Interdependence (eg data interdependence) between items of different levels of safety significance, whether intended or unintended, could degrade the higher level safety function.

1.8.2.12 Changes in the development environment might not be the subject of appropriate impact analysis procedures and create unknown intrusion pathways.

1.8.2.13 Traditional methods of hazard analysis described in established industry standards (such as failure modes and effect analysis, fault tree analysis, event tree analysis, and traditional security and vulnerability analysis techniques) may not discover new hazards from unwanted intrusion or interaction.

1.8.2.14 Attacks on computer-based systems can be difficult to detect. Attacks may consist of multiple steps spread over time, at multiple integration levels, combining different vulnerabilities.

1.8.2.15 The software development process can provide opportunities for an attack if it is not properly secured.

1.8.2.16 Use of removable media for data transfer (from or to the system) is a significant mechanism for attack. Removable media can contain a large amount of data, are readily available and are difficult to control.

1.8.3 Common Position

1.8.3.1 All items that include software and can affect nuclear safety functions shall be identified and shall be subject to the common positions in this chapter. In this context, software includes firmware and logic in any form, including supporting data. Relevant items include:

a application specific software,

b operational and runtime software, including operating systems and their components,

c pre-existing software, including programmable logic and IP cores,

d software in supporting equipment (see 1.8.1)

e fixed stored programmed and configuration logic,

f IP cores,

g software tools,

h data that affects the safety function, including control, alarm and protection setpoints,

i external data storage devices,

j network equipment and devices (switches, routers, bus, network etc),

k items necessary to determine whether a system is running in a real or test environment,

l items necessary to maintain required independence,

m items (temporarily or permanently) connected to another relevant item,

n items intended for non-safety purposes that have the potential to interfere with safety systems, such as IT systems, personal computers and personal mobile phones.

1.8.3.2 All items (see 1.8.3.1) used in the supply chain that can affect the safety system, including items on which the supply chain products and services depend, shall be subject to the common positions in this chapter (for example through contractual obligation). This includes items used by contractors, vendors, technical support organisations and any other service provider. Relevant supply chain activities include software upgrades, patching, analysis using external tools, testing, system modification and transportation.

1.8.3.3 Safety and security shall be integrated conceptually and logically across the system lifecycle, avoiding conflicts and inconsistencies. This will avoid security features unduly impacting the safety functions, and vice versa. The integration shall include:

a goals and losses of concern,

b safety classification,

c safety and security demonstration,

d objectives,

e roles and responsibilities,

f policies, processes and procedures

g requirements,

h plans,

i analysis,

j design,

k operability,

l verification and validation,

m assurance,

n the management and content of operational and maintenance procedures.

1.8.3.4 For example, any cyber security features necessary for safety shall be developed and/or qualified to the same safety classification as the system these features secure. If not, evidence shall be presented that the cyber security features cannot adversely affect the safety function.

1.8.3.5 Another example concerns the need to identify security constraints and appropriate tests as early as possible and to assure that the system is safe and secure once an already qualified part (such as a platform) is integrated into the system. Any additional verification and validation (eg manual review, fuzz testing of communication protocols) that is not possible for individual parts shall be done at the earliest possible phase of integration. The aim is to reduce the risk of compromising safety once the system is installed by avoiding security testing from installation onwards.

1.8.3.6 The integration between safety and security shall also include the architecture, and how it supports safety and security objectives. For example, some aspects designed for safety also benefit security, such as:

a prevention of data transmission from a less qualified system to a more qualified system (preferably ensured by use of hardware that restricts by solely physical means data transmission to one direction, eg data diodes),

b online surveillance of communication between channels of a safety system and between safety systems,

c exclusion of network connections between a safety system and any system external to the plant,

d prevention of a safety function being affected by data not qualified to the same safety significance.

1.8.3.7 Requirements, including constraints, shall be specified for measures across the lifecycle to protect against any threat with the potential to degrade a safety function, based on the consequences of a successful attack. Threats may involve items on which the safety function depends (see 1.8.3.1 and 1.8.3.2) and supporting activities (see 1.8.1). Example threats include:

a design basis threats identified by the licensee and/or the regulatory authority,

b unauthorised access through items on which the safety function depends,

c unauthorised influence by a less qualified item (such as provision of false information),

d unauthorised access or modification through supporting activities.

1.8.3.8 Potential vulnerabilities shall be considered for all items. Vulnerabilites may result, for example, from a poor development process, the inadequate application of pre-developed software, or infection with malware. Specific examples include unused code, unused or non-specified functionality, non-specified variables and undetected malware. (The determination of potential means for degradation, accounting for vulnerabilities, threats and consequences for the plant, leading to requirements for mitigation and adequate protection, may be referred to as a security risk assessment.)

1.8.3.9 All known vulnerabilities shall be mitigated, either through eliminating the vulnerability or by requiring adequate protection.

1.8.3.10 Requirements shall also be specified for effective incident management and system recovery.

1.8.3.11 Appropriate measures shall be required for detecting and recording intrusions. Forensic capture arrangements shall be determined to facilitate effective investigation and learning following any significant intrusions.

1.8.3.12 Requirements, including constraints, shall be formulated to protect against intrusion or unauthorised modification during supporting activities such as testing, calibration (including update of calibration data), configuration, modification, loading software, other maintenance, and documenting and associated management activities. Example requirements include:

a connecting particular devices only when necessary (ie restricting modes and/or times of use),

b restricting access (eg to specific staff with appropriate roles and security clearance),

c limiting access to specific time intervals and/or situations,

d protecting communication content (for example by verifying after transmission that data has not been modified by a third party),

e integrity checking (eg comparing software or data against its original source, stored independently),

f limiting interfaces and interactions,

g preventing modification by inherent design (for example storing software on read only memory or separate storage partitions with read only access),

h protecting boundaries between different layers of defence in depth, ideally by physical means,

i appropriately assessed procedures and equipment for transportation.

1.8.3.13 Access to systems shall be restricted to only those parts that are necessary and only to individuals that are suitably qualified and explicitly authorised, and only to items that are explicitly authorised. Example means for restricting access include physical mechanisms such as key lock systems and software mechanisms such as passwords.

1.8.3.14 For safety systems, physical access to the system shall require the presence of two persons who are capable of detecting potentially degrading activity.

1.8.3.15 For safety systems, security requirements for the transport of the system, and/or its software, shall be specified by the licensee for each aspect of the process (eg addressing, dispatch, transit and receipt). These requirements shall be met every time transport of such items occurs.

1.8.3.16 Operating experience of threats and vulnerabilities (eg onsite security incidents or detected cyber attacks elsewhere that could potentially degrade system safety) shall be accounted for in the specification, implementation and validation of security requirements. See also 1.14.3.1 to 1.14.3.3 and 1.14.3.6 to 1.14.3.9.

1.8.3.17 Any change to the system or its environment (including changes to the threat assessment) shall be analysed and evaluated for its potential to degrade the safety function. All security modifications shall comply with 2.7.3 (on change control and configuration management). The effect on safety of any new security requirement shall be evaluated. This analysis shall be triggered by awareness of changes and also by scheduled periodic review.

1.8.3.18 The training and experience required in 1.6.3.2 d shall include that necessary for security, including preventing, detecting and mitigating the consequences of intrusions and attacks.

1.8.3.19 The security requirements shall be implemented.

1.8.3.20 Licensees shall agree with the regulator, and then implement, appropriate graded reporting arrangements.

1.8.3.21 The following common positions are also particularly applicable in the context of security:

Chapter 1.4 on pre-existing software:

All common positions in 1.4.3, with security aspects integrated to the extent these affect safety.

Chapter 1.7 on software quality assurance programme and plan:

All common positions in 1.7.3, paying particular attention to potential conflicts to safety from security driven software modificatons (eg patch management).

Chapter 2.2 on computer system architecture and design:

Common positions 2.2.3.4 and 2.2.3.5, resolving potential conflicts to safety from time and resource consumed by security measures (eg intrusion detection or online malware scanning), and 2.2.3.12.

Chapter 2.3 on software requirements, architecture and design:

Common positions 2.3.3.3, 2.3.3.4, 2.3.3.5, 2.3.3.7, 2.3.3.10, 2.3.3.14 and 2.3.4.5 to 2.3.4.7.

Chapter 2.4 on software implementation:

Common positions 2.4.3.2, 2.4.3.3, 2.4.3.4, 2.4.3.7 to 2.4.3.12, 2.4.3.14, 2.4.3.15, 2.4.3.18, 2.4.3.25, 2.4.3.26, 2.4.3.29 to 2.4.3.31, 2.4.3.32, 2.4.3.35 and 2.4.3.36.

Chapter 2.8 on operational requirements:

Common position 2.8.3.19.

1.8.4 Recommended Practices

1.8.4.1 The following requirements derived from operating experience should be considered.

a Identify software that is running in a test environment instead of a real system, and ensure that all software is running in its intended operating mode (eg for testing or real-time operation).

b During normal plant operation, physically isolate unnecessary systems (eg for maintenance, training and other supporting functions). Security arguments often rely on this physical isolation. Evaluate and justify the effectiveness of this isolation if it is required (for example consider manual activities before and after support functions that might compromise the isolation).

1.8.4.2 The licensee should define and strictly enforce a policy for the onsite use of all electronic devices, by its own staff, contractors, vendors and technical support organisations. This should include personal computers, personal mobile phones and removable media, and include consideration of intrusion mechanisms, eg disabling wireless capability.

1.8.4.3 Physical and logical separation of systems of different security levels should be applied using security zones, which group together systems of the same security level.

1.8.4.4 All authorised access to systems, and changes to system parameters, should be recorded, to facilitate security audits.

1.8.4.5 All changes to system parameters should be consistently logged, to facilitate security incident detection.

1.8.4.6 The licensee should implement a policy for reporting significant security attacks to alert other facilities.

1.8.4.7 The following recommended practices are also particularly applicable in the context of security:

Chapter 2.4 on software implementation:

Recommended practice 2.4.4.11.

Chapter 2.8 on operational requirements:

Recommended practice 2.8.4.4.

*   *   *

1.9 Formal Methods

"Saber y saberlo demonstrar es valer dos veces"

Baltasar Gracián

1.9.1 Rationale

"Formal methods" is a general and rather confusing term which is used to refer to various methods for modelling systems, expressing specifications, programming, analysis, proof, notations, V&V, and even design and code generation. In this document, the term will be used to refer to the use of mathematics and logic to describe systems and designs in a way that makes their mathematical or logical analysis possible. The terms correctness, consistency, soundness, and completeness are used in this chapter with their classical meaning in mathematical logic.

In practice such methods offer the possibility of establishing precise and unambiguous specifications, thus helping in a – sometimes not easy – understanding of specifications. The availability of such precise descriptions enables analysis, verification, and sometimes proof. Another effect of these methods is to increase the possibility of more systematically or even automatically refining the specifications (levels of design), towards executable or high-level language code. The latter possibility, however, is often confused with the automatic generation of code (compilation) from high level application-oriented programming languages.

Precise models of system specifications also allow animation. Animations may be used for checking certain aspects of the requirements and may provide a reference to test the final software. Mathematical methods may also be used for the validation of system requirements, as they can be used to show that certain properties are consequences of the specification, ie that programs which satisfy the specification will necessarily have these properties.

As a general remark, it is useful to remember that system design, verification and validation are human activities which necessarily imply the use of abstract representations and descriptions. We cannot apprehend reality without the use of mental models. In particular, safety can only be discussed and anticipated as an (abstract) property of models of system structure and behaviour, of its interactions with the environment and of accident scenarios.

1.9.2 Issues Involved

1.9.2.1 Potential benefits of formal methods may include:

- more systematic verification of system specifications for consistency and completeness;
- provision of more rigorous strategies for decomposition and design;
- more systematic refinement of the system specifications towards descriptions from which source or object code can be generated;
- demonstration of syntactic correctness of software specifications;
- verification of a piece of code using mathematical proof or rigorous analysis techniques to show that it is a complete and correct translation of its specification;
- validation by using mathematical proof techniques to derive properties from the specification (e. g. proving that an unsafe state is not reachable);
- evidence that a module with a formally specified and verified behaviour can be incorporated into a larger software system in such a way that certain properties, eg safety, remain satisfied.

1.9.2.2 A useful introduction to formal methods and their use for the certification of critical systems can be found in [10].

1.9.2.3 Unfortunately, mathematical analysis has significant limitations, which make its use problematic in practice. Each type of analysis can cover only specific aspects as no theory currently can combine all the features and properties found in a real system. In addition, mathematical analysis cannot establish the validity of system requirements.

1.9.2.4 Moreover, when used inappropriately, formal methods may be dangerous as:

- their lack of legibility may lead to difficulties in understanding and verification, especially for plant specific application software which must be understood by different branches of engineering,
- no method is universal, and the impossibility of expressing some types of requirements important to safety (e. g. non functional requirements or aspects of real time behaviour) may lead to incompleteness or inconsistencies,
- they might be used by insufficiently trained personnel or without support of adequate tools.

1.9.2.5 In addition, the training effort required by the use of a formal method can be disproportionate to the benefits that can be expected.

1.9.3 Common Position

1.9.3.1 No credit can be taken in a safety demonstration for the use "per se" of a formal method without due consideration being given to the specific evidence brought in by this use, and to its contribution to the safety demonstration of the system.

1.9.3.2 Whatever (combination of) method(s) and notation(s) is used to describe the system requirements, this description shall be based on a definition of the system boundaries and on a systematic capturing of the functional and non-functional requirements of the system. These boundaries, requirements and associated properties shall be explicitly, unambiguously and accurately documented (see 2.1.3.2).

1.9.3.3 Selection of methods and tools with respect to their intended application for formal descriptions and mathematical analysis shall be justified. The justification shall be in accordance with the safety demonstration (see 0).

1.9.3.4 There shall be objective evidence of a successful use of the formalisms and methods used in an application with comparable properties.

1.9.3.5 The procedures and constraints for using the formal methods and tools shall be documented.

1.9.3.6 Any limitations of the formal descriptions, methods and tools used, and resulting descriptions, shall be explicitly documented. For example, any limitation in describing and reasoning about non-functional requirements, use of resources, or time critical events shall be stated.

1.9.3.7 The formalisms and the methods used for specifying the system requirements shall be unambiguous and understandable by all technical staff involved.

1.9.3.8 The formal description of the system requirements shall be validated against the results of a prior plant safety analysis, and of other relevant analyses at the plant level (see 2.1.3.1).

1.9.4 Recommended Practices

Criteria for the choice of formal methods

1.9.4.1 Training courses and textbooks on the formalisms and methods in use should be available.

1.9.4.2 Tools should enable formal descriptions to be checked for consistency. Tools should enable the use of animation, so as to aid the checking of completeness.

1.9.4.3 The tools associated with the formal method should provide means for producing the necessary documentation for the project, including the ability to add natural language comments to the notation to aid understanding.

Verification when applying formal methods

1.9.4.4 Logical reasoning to justify properties of correctness, consistency and completeness should support the formal descriptions, and where appropriate mathematical analysis should be used to verify properties of input, output and dynamic behaviour of the items described. Checks on formal descriptions comparable to the checks performed by a good compiler on a program should also be performed when possible.

1.9.4.5 Development methods based on application oriented graphical languages generally offer a means for connecting and configuring a set of predefined components in logic diagrams. For these methods, the relations between these predefined components in the logic diagrams should be verified to show that they satisfy a set of syntactical and semantical rules. For this purpose, simulation and/or animation tools should be incorporated.

1.9.4.6 Some development methods supported by a graphical notation, for example those which incorporate a structured design approach, offer top-down consistency checks between levels, type checking and automatic code generation. For these methods, additional verification (eg testing, inspections, static analysis) should still be performed in order to detect semantic faults.

* * *

1.10 Independent Assessment

1.10.1 Rationale

For a software-based safety system an independent assessment of the system is essential to provide the degree of confidence in the design process, in the product and in the personnel involved. The independent assessment ordered by the supplier of the system or by the licensee can be regarded as an important part of the evidence in the safety demonstration. Independent assessment can also be an essential activity during modification of the system.

Its purpose is to provide an objective view on the adequacy of the system and its software which is – as much as possible – independent of both the supplier and the user (licensee).

1.10.2 Issues Involved

1.10.2.1 Ensuring that the independent assessors are independent of project pressures, which – depending on circumstances – may come from the development team, the supplier or from the user, in order that assessors are able to comment freely and frankly without being adversely influenced by these project pressures.

1.10.2.2 Ensuring that independence does not deprive the assessors from access to all relevant information and from the level of competence and familiarity with the system which is necessary to make an efficient analysis.

1.10.2.3 Making sure that the findings of the independent assessment are properly addressed and recorded, and, where changes are proposed, these are handled through the change management process.

1.10.2.4 Independent assessment may be generic or targeted to a given application. In the first case it may allow components to be pre-qualified and re-used in applications, the independent assessment of which may become easier.

1.10.3 Common Position

1.10.3.1 The system and its safety demonstration shall be subjected to a documented review by persons who are:

a competent (commensurate to the nature and complexity of the system);

b organisationally independent of the supplier(s) of the system (and of its safety demonstration), and c not responsible for – or not involved in – the development, procurement and production chain of the system.

1.10.3.2 The area and depth of assessment shall be carefully considered and proposed in each case by the independent assessors, in order to be commensurate with the objective of confirming (to a high level of confidence) that the delivered/modified system will satisfy its safety requirements.

1.10.3.3 Independent assessment of the product shall be undertaken on the final version of the software. The independent assessment should also involve examination of process activities and the resources on which these activities depend, eg people, tools, documentation.

1.10.3.4 Where the independent assessment reveals that certain functional or non-functional requirements or features of the system have not been adequately implemented, ensured or understood, the independent assessor shall identify and document them for response by the development team.

1.10.3.5 Safety questions raised by the independent assessors shall be resolved; the assessor's comments on the responses provided shall be a documented component of their final reporting.

1.10.3.6 The scope, the criteria and mechanisms of the independent assessment activity shall be the subject of an independent assessment plan. This plan shall be formulated as soon as possible in the project evolution, so that the necessary servicing of the associated activities can be properly accommodated and resourced as part of the management of the overall project programme.

1.10.3.7 The independent assessment shall be agreed upon by all parties involved (regulator/licensee/supplier), so that the appropriate resources can be made available at the agreed time.

1.10.3.8 The tools and techniques used by the independent assessor shall, where practicable, be different from those used in the development process and the system's safety demonstration. Where it is necessary to use the same or similar tools, the independent assessor shall recognise this in the independent assessment plan and propose a different approach to guard against a common failure of both the independent assessment and the development team to reveal faults within the software.

*   *   *

1.11 Graded Requirements for Safety Related Systems (New and Pre-existing Software)

1.11.1 Rationale

Software is increasingly used in many different nuclear applications. Not all of this software has the same criticality level with respect to safety. Three classes of systems have been defined and discussed in chapter 1.2: safety systems, safety related systems and systems not important to safety.

It is recognised that for safety related system software the requirements for demonstrating an adequate level of safety can be reduced.

Simplicity is required for safety systems. Safety related systems can be more complex. For these latter systems less information may be available on the development process and on the product. In certain cases, it might be possible to compensate for this lack of information – typical for pre-existing software of safety related systems – by using evidence provided by functional testing and adequate operational feedback.

For the operational feedback, it is important to determine which data are to be collected, and how they must be collected and evaluated.

The requirements on safety related system software can sometimes be relaxed, based on the results of a failure mode analysis of the system, or on arguments or conditions of use which show that the reliability required from the software is low.

1.11.2 Issues Involved

1.11.2.1 The safety classification of a computer-based system and of its software is determined by the importance to safety of the functions it is implementing.

1.11.2.2 The quality of a piece of software can neither be quantified, nor tailored to demand. Graded requirements are therefore not intended to define concessions and relaxations allowing lower quality standards of design and development for certain classes of software. As mentioned in 1.2.3.11, they are intended to determine which development process attributes are adequately required, and the amount of verification and validation necessary to reach the appropriate level of confidence that the system is fit for purpose. It is therefore essential to identify and assess the importance to safety of the software as well as any graded requirement scheme intended to be used, as accurately and as early as possible in a new project.

1.11.2.3 As a general principle, and as already stated in 1.2.3.6, a computer-based system important to safety and its software is a safety related system if its correct behaviour is not essential for the execution of safety functions (see also 1.2.4.6).

1.11.3 Common Position

1.11.3.1 The safety class of a particular computer-based system shall be determined according to the principles of 1.2.3.1. Graded requirements as defined in issue 1.11.2.2 cannot be invoked in the safety demonstration of a computer-based system of any class without a justification for the use and adequacy of these graded requirements.

1.11.3.2 Exceptions to requirements for safety systems can be made for safety related systems, but must be justified.

1.11.3.3 In order to evaluate the possibility of relaxing certain requirements of the safety demonstration, as a minimum, the consequences of the potential modes of failures of the computer-based system shall be evaluated. For instance, a failure mode analysis may show that certain relaxations are possible, when failures of the system can be anticipated and their effects can be detected and corrected in time by other means.

1.11.3.4 More generally, for safety related systems, the specific requirements of IEC 61226, 2nd ed. section 7 can be taken into consideration. The relaxations to the prescriptions of IEC 60880 and to quality assurance shall be justified (for example through reference to a standard applicable to the lower safety class – see 1.11.4.4).

1.11.3.5 The common position 1.2.3.17 is applicable.

1.11.3.6 All common positions and recommended practices of this document which are explicitly stated for safety systems are either not enforced, or are enforced and can be relaxed on safety related systems. These positions and practices are those mentioned in:

Chapter 1.2 on system classes, function categories and graded requirements for software:

Common position 1.2.3.16

Recommended practice 1.2.4.3

Chapter 1.4 on pre-existing software:

Common position 1.4.3.6

Chapter 1.5 on tools:

Common positions 1.5.3.4, 1.5.3.7, 1.5.3.8

Chapter 1.6 on organisational requirements:
Common positions 1.6.3.2 a to c
Recommended practice 1.6.4.3

Chapter 1.8 on security:
Common positions 1.8.3.14, 1.8.3.15

Chapter 2.2 on computer system design:
Common positions 2.2.3.6, 2.2.3.13

Chapter 2.4 on coding and programming directives:
Common positions 2.4.3.1, 2.4.3.13, 2.4.3.14 b
Recommended practices 2.4.4.10

Chapter 2.5 on verification:
Common position 2.5.3.26

Chapter 2.6 on validation:
Recommended practice 2.6.4.1

Chapter 2.7 on change control and configuration management:
Common positions 2.7.3.2, 2.7.3.11, 2.7.3.12, 2.7.3.15, 2.7.3.20, 2.7.3.22
Recommended practices 2.7.4.1, 2.7.4.2

Chapter 2.8 on operational requirements:
Common positions 2.8.3.9, 2.8.3.12, 2.8.3.14
Recommended practices 2.8.4.2, 2.8.4.4

1.11.4 Recommended Practices

1.11.4.1 As mentioned in 1.2.4.4, it is recommended that the failure analysis required in 1.11.3.3 is performed on the system specifications, rather than at code level, and as early as possible in the project so as to reach an agreement between the licensee and the regulator on graded requirements and/or the requirements for the demonstration of safety.

Graded requirements for safety related systems

1.11.4.2 Design and V&V requirements addressing factors which affect indirectly the behaviour of programmed functions may be subject to possible relaxations: eg validation of development tools, SQA and V&V specific plans, independence of V&V team, documentation, and traceability of safety related functions in specifications.

1.11.4.3 Table 2 and Table 3 give an example of classes and graded requirements which could be used in the demonstration of the safety of computer-based safety related systems. Graded requirements are given for pre-existing software (Table 2) and for newly developed software (Table 3). In each case safety related system graded requirements are compared to safety system graded requirements.
It is important to note that these example tables are not meant to provide a complete listing of safety related system and safety system graded requirements.
Some of the principles on which these tables are founded are expressed in the rationale (1.11.1 3rd paragraph). Safety related systems can be more complex than safety systems, and less information may be available on their development process and on their software. In certain cases, it might be possible to compensate for this lack of information – typical of pre-existing software of safety related systems – by provision of evidence resulting from restrictions as to their conditions of use, an analysis of failures and consequences, functional testing and adequate operational feedback.

1.11.4.4 The following standards include sets of graded requirements:

a IEC 62138 includes graded requirements for the qualification of software executing functions of the categories B and C defined in IEC 61226;

b IEC 61508 includes graded requirements for different safety integrity levels.

Table 2: Example of comparative classes of graded requirements for pre-existing software
(continued on next page)

	Systems not important to safety	Safety related systems	Safety systems
Standards		– Recognised codes and guides	– IEC 60880-1986 and IEEE 7-4-3-2 principles, and in particular the requirements of this column
Quality assurance	– Professional level	– SQA plan (generic) covering the whole lifecycle – Evidence of structured design approach – Configuration and software maintenance management after installation	– SQA plan (specific to the project) covering the whole lifecycle – Structured design approach (phases) – Reviews of ends of phases – Evidence of experience of staff – Configuration and software maintenance management after installation
Safety assessment		– Analysis of impact on safety of the software component potential failures – Analysis of risk and consequences of software common cause failure; defence and mitigation countermeasures	– Analysis of impact on safety of the software component potential failures – Analysis of risk and consequences of software common cause failure; defence and mitigation countermeasures
Development	– Complete, clear, non-ambiguous and auditable specifications	– Complete, clear, non-ambiguous and auditable specifications – Identifiability of safety related functions in the specifications – Autocontrol functions (hardware and software) included in specifications – Design/coding limiting faults – Validated tools	– Complete, clear, non-ambiguous and auditable specifications – Identifiability of safety and safety related functions in the specifications – Traceability of specifications through the development – Autocontrol functions (hardware and software) included in specifications – Demonstration of complete coverage by autocontrol and periodic tests – Design/coding limiting faults – Validated tools

Table 2: Example of comparative classes of graded requirements for pre-existing software
(continued from previous page)

	Systems not important to safety	Safety related systems	Safety systems
Verification/ validation	– Supplier's V&V – Site commissioning	– V&V plan – Verification (chapter 2.5) at the end of development – Independent validation (chapter 2.6) – Site commissioning	– V&V plans – Verification (chapter 2.5) at the end of each design phase – Independent V&V (chapter 2.5 and 2.6) – Independent assessment (chapter 1.10) – Site commissioning
Documentation		– Auditable documentation demonstrating satisfaction of requirements	– Integrated set of auditable documents
Operational feedback		– Relevant and documented operational feedback	– Relevant and documented operational feedback

Table 3: Example of comparative classes of requirements for the development of new software
(continued on next page)

	Systems not important to safety	**Safety related systems**	**Safety systems**
Standards		– Recognised codes and guides	– IEC 880-1986 and IEEE 7-4-3-2 principles, and in particular the requirements of this column
Quality assurance	– Professional level	– SQA plan (generic) covering the whole lifecycle – Evidence of structured design approach – Evidence of experience of staff – Configuration and software maintenance management after installation	– SQA plan (specific) covering the whole lifecycle – Structured design approach (phases) – Reviews of ends of phases – Evidence of experience of staff – Configuration and software maintenance management after installation
Development	– Complete, clear, non-ambiguous and auditable specifications – Standard tools	– Complete, clear, non-ambiguous and auditable specifications – Identifiability of safety related functions in the specifications – FMECA of hardware and software – Analysis of risk and consequences of software common cause failure; defence and mitigation countermeasures – Autocontrol functions (hardware and software) included in specifications – Design/coding methods limiting faults – Validated tools	– Complete, clear, non-ambiguous and auditable specifications – Identifiability of safety and safety related functions in the specifications – Traceability of specifications through the development – FMECA of hardware and software – Analysis of risk and consequences of software common cause failure; defence and mitigation countermeasures – Autocontrol functions (hardware and software) included in specifications – Demonstration of complete coverage by autocontrol and periodic tests – Design/coding methods limiting faults (evidence of programming directives) – Validated tools

Table 3: Example of comparative classes of graded requirements for the development of new software
(continued from previous page)

	Systems not important to safety	Safety related aystems	Safety systems
Verification/ validation	– Supplier's V&V – Site commissioning	– V&V plan – Verification (chapter 2.5) at the end of each development phase – Independent validation (chapter 2.6) – Site commissioning	– Specific V&V plan – Verification (chapter 2.5) at the end of each design phase – Independent V&V (chapter 2.5 and 2.6) – Independent Assessment (chapter 1.10) – Site Commissioning
Documentation		– Auditable documentation demonstrating satisfaction of requirements	– Integrated set of auditable documents
Operational feedback		– Relevant and documented operational feedback for support software, libraries and other re-used software	– Relevant and documented operational feedback for support software, libraries and other re-used software

1.12 Software Design Diversity

"Parempi virsta väärää kuin vaaksa vaaraa."

Old Finnish proverb

1.12.1 Rationale

Identical redundancy is effective only in guarding against random hardware failures; it does not provide protection against systematic failures since for these the redundancy can fail as well, resulting in a common cause failure. A systematic failure can only be eliminated by changes in specification, design, implementation and configuration. Diversity can be introduced to provide protection against common cause failures, as a means of achieving independence for systematic failues that would otherwise be common cause failures. For diversity to be effective, it needs to be applied methodically and deliberately according to well-chosen strategies. Generally, the objective is to increase the reliability that can be claimed for the overall system architecture by reducing common cause failures. This chapter focuses on software design diversity issues arising from the use of computer-based systems.

1.12.2 Issues Involved

1.12.2.1 System architecture software considerations

The system architecture provides the context for software design diversity. The objective is to achieve properties for the overall architecture, which will depend on how diversity is employed within it. This section identifies the relevant issues.

One architectural approach is to use multiple diverse systems in parallel, with the assumption that systematic failures will not occur concurrently between them.

Another architectural approach is defence in depth, where the diversity is applied between layers of protection, such that a failure in one layer does not result in a failure of the overall system architecture.

Both these approaches use diversity as the means for achieving independence (discussed further below).

Any approach needs to consider the number of systems, components or channels, the degree of diversity between them, the selection of the technology for each of them, and possibly the apportionment of reliability targets.

One approach that strongly addresses the objectives for software design diversity for a computer-based system is to employ a simple non-computer-based secondary system.

1.12.2.2 Independence as a design goal

Generally, system architectures are designed with independence claimed (either explicitly or implicitly) for systematic failures, between layers of protection and wherever diversity has been employed. Genuine independence between systems, components or channels can be easily compromised, for example through sharing just one common resource such as a means of communication.

Factors affecting and compromising independence relate, for example, to the conceptual design, the programming language, practice, coding style, tools used for implementation, and physical separation. Total independence, in current practice, is extremely hard (usually impossible) to demonstrate. A safety demonstration that tolerates some limited (often unknown) failure dependence between computer based systems, where the safety function is still achieved, is an accepted (and more realistic) practice.

This practice intends to prevent, as far as possible, defects that are potential common causes of unsafe failures and to structure the design so that the effect of the remaining unknowns on the safety function is insignificant or acceptable. Chapter 2.2 on computer system architecture and design provides relevant requirements. It is at the system architecture level that functions are assigned to the diverse systems, components or channels and an example of the use of functional diversity at this level is discussed below.

The design of the system architecture should employ the technique of defence in depth (eg to ensure multiple defences such that a failure in one system is mitigated elsewhere). An example of this defence in depth would be a protection system providing protective action in case of a control system failure. In this example software might be used in both the control and protection systems; and the potential for common cause failures due to the software would require analysis. More generally the potential for common cause software-related failures across all defence barriers utilising computer-based systems should be analysed.

Decisions taken as to the devices used at the field level (eg sensors and actuators) will also require careful consideration since modern instrumentation increasingly utilises software components (eg in smart sensors and actuators). For example, the use of the same smart sensor/actuator type across diverse systems (eg primary and secondary protection systems and;

control and protection systems) will introduce an element of common software, the implications of which should be fully analysed.

1.12.2.3 Hardware and software intrinsic quality

The correctness and completeness of the hardware and software implementation constitute the primary means of avoiding systematic failures (and hence common cause failures of this type). The evidence obtainable to support claims made for correctness and completeness may not, in some cases, be sufficient (ie may not be commensurate with the level of assurance required in the safety demonstration). Diversity is a secondary approach to prevent common cause failures within the system.

The use of diverse channels, subsystems or components, therefore, neither compensates nor is an excuse for using hardware or software of inadequate quality (with respect to systematic failures). Diversity is intended, like defence in depth, to bring additional evidence on the absence of common cause failures, and not be a substitute for a lack of evidence that the hardware and software of the diverse components are correctly specified and implemented.

Diverse channels or components of inadequate quality do not make a safe system. For example, a channel or component would be unacceptable in any the following cases:

- its requirements specification is not assured for completeness, correctness, consistency and freedom from ambiguity,
- it cannot be shown to satisfy its specification,
- there is insufficient evidence to demonstrate its claimed reliability,
- there is insufficient evidence to demonstrate adequate independence from all other components and/or channels claimed as diverse, with respect to potential common cause failures,
- all anticipated unsafe failure modes will not always be detected, or
- it is not always able to properly take over control when any of the other channels or components fail.

Depending on the application, other conditions for acceptability may apply.

1.12.2.4 Software diversity

Software diversity can be introduced to overcome the problem of software common cause failures in redundant parts of a computer-based system. Software diversity is introduced by different means, N version techniques being the most common. Since such computer-based systems tend to be complex, arguments of simplicity and correctness typically used to support hardware or procedural diversity may not be applicable. N version techniques typically involve the independent creation of programs for the diverse computer-based systems. Those building the programs typically work independently of each other with no direct communication. The teams can either be given complete freedom on the production methods or methods can be imposed (forced diversity). The validity of this approach depends on the assumption that the shared requirements do not contain any errors that cannot be corrected during development. The next assumption is that the faults introduced by the teams will be diverse and common cause failures across the computer-based systems will be reduced; unfortunately, there is no precise way of confirming the extent to which failures are actually reduced as a result. There are many different sources of potential software common cause failures and research in this field shows that statistical independence cannot always be assumed [11], [12] and [13]. Problem complexity is a common factor that could compromise the quality of each implementation. Then, positive correlation of failures cannot be ruled out and this will lead to worse reliability than suggested by assumptions of statistical independence (eg simple multiplication of each of the reliabilities for a one-out-of-two system architecture). As a result, safety case arguments are required for the reliability achieved from software diversity.

1.12.2.5 Software reliability figure

See chapter 1.13 on software reliability for a discussion on the reliability figure to be used following introduction of software diversity.

1.12.2.6 Diversity design options and seeking decisions

Diversity seeking decision-making aims to maximise the effectiveness of methods and techniques to ensure diversity of failure behaviour across diverse software programs. Current practice is founded on engineering judgment, which proposes the best means to force diversity. The possibility of providing further scientific support to these judgments is an open research issue. The guidance provided in this document is, therefore, predominately based on engineering judgment. Examples of methods to ensure diversity include specification of different functions, use of different development environments, tools, languages, design and coding methods, algorithms and V&V methods etc. The intent is to present the development teams with different methods and techniques for specifying requirements, under the assumption that such diversity will reduce the likelihood of common cause failures. Selection of the simplest software design and implementation that satisfies the safety requirements can also reduce the potential for failure, so the methods used to enforce diversity ideally should not compromise this.

Among the various methods to force diversity, the use of functional diversity is generally accepted as a particularly important measure (eg [1]; and [14] which specifically recommends functional diversity implementation unless it is impossible or inappropriate; also see eg [13]). Functional diversity typically involves the use of different parameters in each system responsible to achieve the same safety objective (eg response to a plant event requiring reactor trip is based on temperature measurement in one system and pressure in the other, using diverse sensing technology, placement etc). The aim is to find the best available means of implementing functional diversity; this might include ensuring that at least one system uses a simple function (eg temperature level trip) where the other(s) employ functions involving non trivial calculations. Functional diversity may be implemented by a computer-based system (ie across different channels or components utilising common hardware) but the the overall gain could be compromised by common elements in the software (eg while the application software might be different, identical components such as operating system software may be the source of commn cause failure). Therefore, the means of implementing functional diversity and the benefit that this approach delivers will need to be carefully evaluated.

1.12.2.7 Common cause failure analysis

While diversity design options aim to use diverse software components, the satisfaction of the intended objective must be assured. This requires a common cause failure analysis of the design and its development processes. This analysis should address the potential for and implications of the use of common modules, algorithms, code and processes across (and within multi-channel) systems. Further guidance on common cause failure analysis can be found in [5].

1.12.3 Common Position

1.12.3.1 The system architecture design shall use diversity as needed to protect against common cause failures and achieve defence in depth (see 2.2.3.2).

1.12.3.2 The decision to use diversity, the type of diversity, the objectives for its use in each case (for example protecting against a particular hazardous condition), or the decision not to use diversity shall be documented and justified.

1.12.3.3 Where a justification not to use diversity is based on the use of particular measures or techniques, it shall be shown how these techniques avoid the potential for common cause failure.

1.12.3.4 The specification of any diverse component shall be demonstrated to be adequate in the context of the documented objective.

1.12.3.5 Diversity shall not be used as a substitute for a lack of evidence that the hardware and software of the diverse components satisfy their own specifications.

1.12.3.6 Software used in a system employing software diversity shall satisfy the requirements of part 2 of this document.

1.12.3.7 The plant design and the computer system architecture shall ensure that the necessary independence is maintained between the systems, components or channels for which diversity is claimed.

1.12.3.8 Functional diversity shall be used whenever possible in the implementation of systems, components or channels intended to be diverse.

1.12.3.9 The design of computer-based systems utilising software diversity shall be justified in the context of the particular application and the objectives, against identified good practice, including but not limited to:

a functional diversity,

b technology diversity,

c independent teams for specifying, decomposing and deriving requirements,

d independent development teams with no direct communication,

e independent teams for performing verification and validation,

f different allocation of requirements to software components,

g different allocation of software components to hardware components,

h different timing and different order of execution,

i avoiding unnecessary complexity in the software design and implementation,

j different description/programming languages and notations,

k different development methods,

l different development platforms, tools and compilers,

m different operating systems,

n different hardware,

o diverse verification and validation.

1.12.3.10 The safety demonstration provided for diverse computer-based systems, components or channels shall include an analysis of any potential common mode or common cause failures. This demonstration shall include consideration of the techniques for diversity noted above. A specific justification shall be given as to the impact on reliability and potential for common cause failure arising from any commonalities (eg identical software components, compilers, V&V methods etc). For each major component of executable target software, the justification shall pay special attention to the possibility of potential common causes of failure specifically presented by this component, and show with adequate and specific analysis and evidence that the corresponding risks are acceptable.

1.12.3.11 The system and software architecture design shall avoid unnecessary complexity while still satisfying the design requirements.

1.12.3.12 When diversity is required to implement the functions allocated originally to one particular system, the diverse software-based systems shall be associated with the same class (see 1.2 and 1.11), unless appropriate justification is provided (eg by applying the classification approach in IEC 61226 [15]).

1.12.3.13 The design of application software, which is usually new software being developed, shall be structured so as to preserve the functional diversity designed at plant level.

1.12.4 Recommended Practices

1.12.4.1 Where diverse safety systems are required, and one is computer-based, consideration should be given to implementing the second one using a simple non-computer-based system.

1.12.4.2 The reliability target assigned to any combination of computer-based systems (eg one out of two protection configuration), components or channels, should be demonstrably conservative having fully considered the factors that could lead to coincident failures (see 1.13.2.6).

1.12.4.3 Where a claim is made that very high reliability has been achieved through software diversity then it must be shown that dissimilar means have been employed in all aspects of the development lifecycle. Any divergence from this should cause the claim to be down rated.

1.12.4.4 Since research is active in this area, the position should be reviewed to enable use of emerging scientific and technical foundations, in particular, to determine claim limits on diverse computer-based systems, components or channels produced by means of software diversity techniques.

1.12.4.5 The application software processing the input data of a same safety function in redundant trains can be implemented by code with adequate protections against common cause failures, eg with variations in variable names, in memory locations, in execution sequences or by other software diversity methods (see1.12.2.41.12.3.9).

* * *

1.13 Software Reliability

Note: This chapter is only applicable where reliability targets (expressed as probabilities of failure) have been set for the computer-based systems.

1.13.1 Rationale

Traditional deterministic approaches for safety demonstration are supported by the use of system classification discussed elsewhere in this document. This approach typically requires the highest international standards to be applied to safety systems. However, use of probabilistic techniques (eg PSA and QRA) may lead to reliability targets (expressed as probabilities of failure) being set for I&C systems important to safety. *This chapter is only applicable where such targets have been set for the computer-based systems.* Within this chapter references to reliability or probability of failure refer to software unless stated otherwise. It should also be recognised that software reliability is only one issue of dependability (see 1.1.1).

Consideration has to be given to (i) what is the best reliability that can be claimed for the software of computer-based systems, and (ii) use of graded requirements to support different reliability levels. It is widely recognised that there is no way of precisely measuring the actual reliability of a computer-based system. Reliability estimates are usually obtained by expert judgment based on operational experience and/or best engineering practices.

1.13.2 Issues Involved

1.13.2.1 Quantitative and qualitative assessment

Qualitative assessment involves using engineering/expert judgement to determine the reliability that can be claimed following application of defined software engineering requirements. Quantitative assessment uses techniques that provide a numerical estimate of the reliability of the developed software (eg software statistical testing). The use of qualitative or quantitative approaches alone or some combination of the two needs to be determined.

The determination of acceptable sets of graded requirements including justification of appropriate techniques and measures to support different reliability levels is a key issue requiring resolution. For instance the IEC 61508 standard [14] addresses this issue through the use of four safety integrity levels (SILs) with recommendations on software techniques and measures applicable for each SIL (the higher the SIL the more onerous the techniques/measures). For example, techniques and measures that have to be considered for satisfaction of the detailed software design and development requirements include formal methods, defensive programming and modular approaches (IEC 61508-3:1998). It should be noted that current approaches are dominated by qualitative requirements. However, the use of Software Statistical Testing (SST) techniques offers the possibility of a quantitative demonstration (see below).

1.13.2.2 Claim limitation

Claims of ultra-high software reliability cannot be demonstrated with current techniques [12]. Therefore, a decision has to be made as to the limit that can be claimed for a computer-based system. A review of nuclear sector standards [15] and [16] shows that claims of lower than 10^{-4} probability of failure on demand for a computer-based system are required to be treated with caution. There are additional problems in determining claim limits when multiple computer-based systems are associated with the same plant initiating event fault (eg multiple protection systems) arising from, for example, common cause failures and these issues are discussed further below and in chapter 1.12 on diversity.

1.13.2.3 Multiple safety/safety related functions

Where a computer-based system implements multiple safety and/or safety related functions it is possible that different reliability targets are assigned to the different functions. This might suggest that different approaches could be taken to the design of the software implementing these functions. Such a claim would require a demonstration of independence between the functions (ie to demonstrate that a worse reliability target function could not prevent the correct operation of a better reliability target function). Without such a demonstration the best individual function reliability target (and hence the more onerous requirements) should be applied to all of the software.

1.13.2.4 Balance safety system versus safety related system

The deterministic approach generally allows no relaxation of requirements within a class but does across classes. Hence higher standards are usually required for a safety system than a safety related system. Using relaxations based on assigned reliability targets alone might lead to the situation where the safety system requirements are less onerous than those for safety related systems (eg as a result of requiring better reliability for a safety related control system than a protection system) and such situations should be viewed with caution. Computer-based safety related systems such as control systems tend to be more complex than protection systems (usually implementing simple shutdown algorithms) and as a result it is recognised as good practice to keep reliability targets for such control systems less onerous.

1.13.2.5 Software statistical testing

The use of software statistical testing (SST) provides the potential to derive an estimate of demonstrated system reliability. There are two main models used to estimate reliability, one of which uses classical [17] and the other Bayesian statistics [18]. For example, using the classical approach we can determine the number of required tests (representative operational transients randomly selected from the input space) to support a reliability claim at a desired confidence level (eg of the order of 50,000 tests with no failure for a 10^{-4} pfd demonstration to 99% confidence). Similar test figures can be derived using the Bayesian approach. For SST to be valid it has to be shown that a number of assumptions hold, which are typically: 1) tests reflect the actual operating conditions of the software, 2) the tests are independent, 3) any occurring failure would be detected by the oracle, and no failures of the SST method and tools are actually detected during the SST. Research into SST is an open area with progress being made into code coverage related techniques and component-based approaches to facilitate reuse. As a result those intending to use SST should carry out a review of SST research to ensure that the steps needed to generate a convincing statistical reliability demonstration are fully understood.

Generation of the test data for SST may be both difficult and time consuming. As noted above the amount of test data required for higher reliability demonstrations is not insignificant. Additionally the time to undertake the test runs and analyse the results has to be factored into the project timescales. It is important that the test runs are truly representative of the plant operational behaviour under the anticipated fault conditions (eg plant parameter data such as temperature and pressure vary in the way expected during demands for plant protection). This might require the use of plant fault simulation software (which will also require validation) to generate representative test data. However, given the representative nature of the tests to actual plant behaviour the use of SST provides an important reliability demonstration.

The role that SST plays in the overall safety demonstration has to be considered, for example, whether to employ as part of the software design and implementation verification or as a confidence check during commissioning. In addition the use of reliability growth models may be considered since they might also provide useful guidance on the reliability.

1.13.2.6 Determination of reliability values when software diversity is employed

A reliability claim intended to be supported by the use software diversity has to be justified. For example, where two diverse computer-based systems in a one-out-of-two protection configuration have been introduced to provide defence against plant faults, the probability of failure for the combination is often taken to be the product of the two system reliabilities. This is, however, a particular case of statistical independence; the actual value could fall between that of the more reliable system and zero (both systems never fail on the same demand), see eg [11], [12] and [19].

Thus, the use of software diversity (see chapter 1.12) presents particular challenges when attempting to determine the probability of failure of diverse systems. Attention, therefore, should focus on those conditions that support claims of statistical independence for software versions. At present there is no precise method to guarantee the achievement of statistical independence: demonstrations are based on qualitative arguments of best engineering practice (eg such as use of diversity seeking decisions) supported by examination of bounding cases and use of conservatism.

1.13.2.7 Assignment of reliability targets when software diversity is employed

The assignment of reliability targets to systems employing software diversity shall be appropriate given the need to ensure software of the highest possible quality (ie up to the defined common cause failure limitation) is used in each system to reduce the chance of residual errors. In particular, claims of high reliability involving multiple low reliability computer-based systems shall not be allowed (eg usage of four 10^{-1} pfd systems as opposed to a single 10^{-4} pfd system).

1.13.3 Common Position – Applicable when reliability targets are used

1.13.3.1 Reliability claims for a single software-based system important to safety of lower than 10^{-4} probability of failure (on demand or dangerous failure per year) shall be treated with extreme caution.

1.13.3.2 The safety demonstration plan shall identify how achievement of the reliability targets will be demonstrated.

1.13.3.3 The sensitivity of the plant risk to variation of the reliability targets shall be assessed.

1.13.3.4 Software of the highest possible quality (ie up to the defined common cause failure limitation) shall be used in each system employing software diversity. Claims of high reliability involving multiple low reliability as opposed to a single high reliability computer-based systems shall not be allowed.

1.13.3.5 The reliability expected from each executable software component (operating system, library, application software, intelligent drivers, sensing and actuation devices, communication protocols, etc.), shall be defined and shown to satisfy these targets. See 2.3.3.8.

1.13.4 Recommended Practices

1.13.4.1 If the software is required to meet a reliability target, schemes using graded requirements assigned to reliability levels should be used during the development and assessment processes. An example of such a scheme would be four reliability levels (eg level 4

– 10^{-4}, 3 – 10^{-3}, 2 – 10^{-2}, 1 – 10^{-1}) with best use made of existing standards such as [14] and [16] to determine the graded requirements applicable at each level. Another approach is the assignment of a probability of failure figure to the class relaxations shown elsewhere in this document.

1.13.4.2 The reliability target assigned to systems of higher safety importance should normally be better (ie lower probability of failure) than that assigned to systems of lower safety importance.

1.13.4.3 The demonstration requirements applicable to the function with the best reliability target (ie lowest probability of failure) of all of the functions implemented within a single computer-based system should be taken to apply to all of the software unless a demonstration of independence between the functions and their implementations is provided.

1.13.4.4 Reliability claims for complex computer-based systems such as control systems should be kept low so as to ease the safety demonstration process. The necessary risk reduction should be provided elsewhere (eg within low complexity protection systems).

1.13.4.5 Where practicable, SST techniques should be used to support the safety demonstration. Consideration should also be given to the use of other reliability models (eg reliability growth) that provide a quantification of the expected reliability.

1.13.4.6 In the case of a replacement or upgrade, the software may be required to have a reliability target that is at least as good as that of the system being replaced. Under appropriate conditions – in particular adequate hardware redundancy – it may be acceptable to compare levels of common mode failure probabilities, ie to require a level of common mode failure probability for the computer-based system that is equivalent to or less than the level for the hardware being replaced.

* * *

1.14 Use of Operating Experience

Über die Antwort des Kandidaten Jobses
Geschah allgemeines Schütteln des Kopfes.
Der Inspektor sprach zuerst hem! hem!
Drauf die anderen secundum ordinem.

Wilhelm Busch, "Bilder zur Jobsiade",
Wiedensahl (Hanover), 1872

1.14.1 Rationale

This chapter addresses requirements for collection and use of operating experience data. By "operating experience", we mean a collection of information on how a piece of software has been performing in service.

Data on operating experience may be collected in different contexts and for different purposes. Data may be collected on systems, components, and tools of different safety classes.

Chapter 1.4 "Pre-existing Software (PSW)" and chapter 1.11 "Graded Requirements for Safety Related Systems (New and Pre-existing Software)" consider, among other factors, the contribution of operating experience to the validation of pre-existing software. This chapter deals with the operating experience in a more general context in order to determine how it can provide useful evidence.

The monitoring and recording of operational experience data may serve different useful purposes, in particular:

– following the classical engineering and scientific approach, to take advantage of the past experience to improve the understanding of systems and of their failure modes, which is necessary to exercise expert and engineering judgment,
– to ensure, for example, through periodic safety reviews, that the safety case remains valid,
– to improve system design and their methods of construction, verification and validation,
– to obtain information on the stability and the maturity of software,
– to support quantitative claims of reliability and availability based on statistical data,

- to provide qualitative confidence in some specific safety properties of similar designs under similar profiles of usage,
- to provide insight, after installation and operation, into the appropriateness of the software qualification process,
- to provide a contribution to the qualification of software development and validation tools.

1.14.2 Issues Involved

1.14.2.1 Operational experience alone does not give enough evidence for the justification of the safety of a safety system or even a safety related system. On the other hand a system that has been used for a long time faultlessly in several installations may offer in practice, with proper documentation and under certain conditions, substantial supporting evidence.

1.14.2.2 Traditionally, I&C operating experience is recorded and evaluated only if disturbances and failures occur. However, information about the I&C system's correct performance during various plant operating modes may provide additional operational experience data.

1.14.2.3 There are many diverse industrial applications of digital I&C systems. Thus, various operating experience data may be available from non-safety applications. Regarding safety applications, however, special qualification requirements have to be met. The fulfilment of these requirements is difficult to demonstrate and in general, no credit can be taken from the operational experience of non-safety applications unless very stringent conditions are met.

1.14.2.4 The acceptability and usability of operating experience data is dependent on several factors, in particular:

a Software configuration management
Configuration management allows effective assessment of whether the product's operating experience is valid in the presence of changes, such as additions to functionality or correction of faults. Uncontrolled changes to the executable code may invalidate the use of operating experience.

b Effectiveness and relevance of problem reporting
In-service problems should be reported together with information about the experienced operating and environmental conditions, and problems observed should be recorded in a way which allows identification of the affected items within the products configuration management system.

c Relevance of the product service operating profile and environment
An analysis should show that the software will perform the same function in the proposed new application as it performed during previous applications. Aspects to compare might be the parameter input and output ranges, data rates as well as performance and accuracy

requirements. For instance, operating experience is not applicable to software functions that were not exercised in the previous applications.

d Impact of maintenance and changes
Maintenance activities contribute to operating experience. As a main aspect of the root cause analysis after a distinct event, the possible impact of earlier maintenance activities like periodical testing under special test conditions or software upgrading have to be analysed. The issue is to properly identify the contribution resulting from:

– incorrect design or execution of maintenance activities, or
– incorrect design of the I&C system.

1.14.2.5 The objectives of operational experience data collection cannot be satisfied unless the involved parties accept the required resource commitments and confidentiality agreements. Licensees may not be required to communicate to the regulator data associated to events that are not related to safety. The issue of concern is that the potential impact on safety of an event may vary from one application to the other.

1.14.2.6 With regard to tool certification, the relevance of operating experience is also dependent on the above-mentioned issues, in particular on adequate configuration management, stability and maturity of the tool.

1.14.3 Common Position

1.14.3.1 Detected faults and failures that may affect safety shall be analysed by the licensee and reported to the regulator. In particular, the potential for common cause failure, the relevance to other software-based systems important to safety and the impact and the necessary improvements and corrective actions to the design, to the software development and the V&V and qualification processes shall be assessed.

1.14.3.2 Operating experience data shall be collected by the licensee in a systematic manner, for instance by means of a computerised database. For safety related events the data record shall comprise at a minimum the general data to identify the event (contextual and historical information) and as much data as possible to reproduce the event.

1.14.3.3 Arguments and evidence shall be provided to confirm that no safety significant faults have been omitted during the data collection activities.

1.14.3.4 Whenever credit is to be sought from operating experience in a safety case, the objectives and criteria for collecting operating experience data shall be identified and shown to be adequate in relation to the safety properties for which credit is sought.

1.14.3.5 Credit can be sought from the operating experience of previous hardware/software installations provided that the following conditions are satisfied:

a The credit is considered as being of no other nature than that which would be sought from factory or site integrated tests; and not, in particular, as evidence for quality of design or for proper functionality.

b The value of the credit as an acceptance or complementing factor is judged against the criteria stated in IEC 60880 Ed. 2 (2006) [5];

c The operating experience data are demonstrably relevant, ie they are shown to satisfy the conditions of relevance of 1.14.3.4, 1.14.3.6 and 1.14.3.8;

d The set of implemented functions and the input profiles (input parameters, data ranges and rates) of the system/component under consideration are demonstrably the same as those of the installations on which data are collected;

e When only parts of a software package for which operating experience credit is sought are used, analysis of the collected data shall show their validity for the intended application and the unused parts of the software must be shown to have no impact on safety.

1.14.3.6 Relevant operating experience data shall at least include:

a meaningful information on the occurrences, the severity of the effects and the actions taken to correct or mitigate all detected faults, anomalies and failures,

b information on malfunctions detected in other applications,

c information on system and software configuration and version, on system operational conditions and on application environment profiles; this information must be shown to be consistent with the configuration, version, conditions and profile of the I&C system under consideration.

1.14.3.7 Where applicable, the above information shall be given separately for the distinct operating modes of the system in relation to the different operating modes of the plant, eg full power, shutdown.

1.14.3.8 When evaluating the relevance of operating experience, the impact of configuration management and maintenance activities shall be taken into account (see in particular 1.14.2.4 d above and 2.7.3.10 to 2.7.3.17 addressing software maintenance, ie software changes after installation).

1.14.3.9 Operating experience data on a system/component that are obtained from other relevant installations of that system/component must be maintained, updated and made available to the licensee and to the regulator.

1.14.4 Recommended practices

1.14.4.1 For different systems designed and qualified on a unique common platform, the operating experience may be coherently evaluated by the vendor to increase the knowledge about their performance according to the common platform requirements. The extent to which the operating experience can be applied more widely should be determined through comparison of the operating profiles of the various systems. The analysis should determine how the platform functions have been exercised by each operating profile.

1.14.4.2 The analysis of a distinct event (see 1.14.2.4 d and 1.14.3.1) needs to assess whether the main design principles and qualification requirements were met during the course of the event. In this evaluation, the vendor should identify and analyse the main design principles and qualification requirements item by item. The evaluation result should make clear whether the event was caused by a system internal fault or by external influences like manual actions, eg as part of maintenance, calibration or testing activities.

1.14.4.3 Appropriate data for the analysis may include documentation of the design, the development process for the system [16], the verification results, the system architecture, and information about associated resources and about malfunctions detected in other relevant applications.

1.14.4.4 If the intention is to use the operating experience for reliability analysis, special data like the time of occurrence, elapsed time for repair and maintenance, processor and data communication load, etc. should be collected.

1.14.4.5 The system supplier should have an organisation in place to allow users to exchange information on their respective operating experience and be kept informed of the system/component evolution.

*   *   *

1.15 Smart Sensors and Actuators

1.15.1 Rationale

Conventional sensors/actuators are becoming unavailable and are being replaced by smart sensors/actuators. Smart sensors/actuators contain microprocessors and the use of firmware and software in these microprocessors presents challenges to the nuclear industry, particularly when they are used in safety systems.

Smart sensors/actuators support sensing and/or control functions and provide computation and communication facilities. Smart sensors/actuators typically have the following hardware: a microcontroller for computation, a small random-access memory for dynamic data, one or more flash memories that hold the program code and long-lived data, an analogue-digital converter, one or more sensors, a communication interface and a power source.

An advantage of these smart sensors/actuators is the modular fashion in which they can be connected to central processing units. As one might expect, many products and variants are available. For example, some smart sensors/actuators contain digital signal processor chips while others support wireless communication.

Most smart sensors/actuators have a minimal operating system that handles interrupts and performs simple task scheduling; sometimes features such as priority scheduling, logging to files on flash or emulating virtual memory are added. The application software layer, using these basic capabilities, implements the application specific functionality, which of course varies considerably.

As far as licensing is concerned, the software of smart sensors/actuators presents many aspects similar to those discussed in chapters 1.4 and 1.11 for pre-existing software (PSW). Thus, several issues, common positions and recommended practices discussed in these chapters apply to the use of smart sensors/actuators.

1.15.2 Issues Involved

1.15.2.1 In addition to the issues discussed in section 1.4.2, the following more specific problematic aspects need to be addressed.

1.15.2.2 The demonstration that smart sensors/actuators are fit for use in a nuclear safety application is not straightforward. High reliability of smart sensors/actuators can only be demonstrated by means of an independent assessment with full access to the design documentation. A user of a smart sensor/actuator, however, would not usually have access to the design documentation. The supplier would not normally allow other parties to review its design documentation and as a result it is not usually produced for that purpose.

1.15.2.3 Access to design documentation presents the main challenge to nuclear operators/licensees. In practice, this access is hindered by several factors such as:

- need to protect the suppliers' intellectual property rights;
- the potential revelation of anomalies in the suppliers' processes and products
- time and effort required from the manufacturer;
- the argument that previous certification (if available) should suffice;
- the relatively small size of the nuclear market which typically means that nuclear licensees have limited influence on smart sensor/actuator suppliers.

1.15.2.4 Smart sensor/actuator suppliers produce their products to a quality that they regard as appropriate, but this cannot account for the requirements of specific use such as those of nuclear plant applications. Since the supplier does not know the context of use, the supplier cannot show that the instrument reliability is sufficient for the intended application.

1.15.2.5 There is currently no nuclear sector standard specifying the design documentation for smart sensors/actuators, nor even the issues that should be addressed. However, some smart sensor/actuator suppliers claim compliance to the IEC 61508 standard [14] but there remains no generic mechanism to satisfy those users who require independent evidence to support a functional claim of high reliability.

1.15.2.6 Assessments will be performed on a specific version of a smart sensor/actuator and the validity of the assessment may be challenged if the supplier changes its product (eg the software version). As a result a process will need to be in place to ensure that the installed smart sensors/actuators are the same versions as those subjected to assessment. In addition a supplier might upgrade a product from a conventional device to a smart device without the end user being aware of this change. Again a process will need to be in place to ensure such changes are detected and the smart sensor/actuator subjected to assessment.

1.15.2.7 The use of smart sensors/actuators often implies a drive to distributed intelligence. It is often thought that this move will increase reliability. This is however not necessarily the case. The use of distributed intelligence replaces one central data acquisition system and simple field sensors with sensors integrating complete acquisition and communication systems in less benign and possibly aggressive environments, hence potentially reducing the reliability of the whole system.

1.15.2.8 When multiple copies of an instrument are used across the plant, the potential for common cause failure both within and across systems is another risk that must be addressed.

1.15.2.9 Engineering competence is another hurdle to the successful implementation of distributed systems. Conventional 4-20mA systems are well understood, easily tested and adequately addressed by current technician skills. Digital signalling requires new skills and understanding, for example, leaving aside the topic of wireless communications, consider the problem of demonstrating that the software has high integrity when Fieldbus communications are employed. Fieldbus contains most levels of the Open Systems Interconnection basic reference model architecture and the volume and complexity of its software is large (eg one smart sensor that has been analysed by the nuclear industry was found to contain 16k lines of code for the implementation of the sensor functionality and another 100k lines of code for the implementation of the Fieldbus communications).

1.15.2.10 As a consequence of all these issues, a pragmatic assessment approach is needed which accounts for scenarios where design information is lacking by making use of compensating evidence (addressing for instance specific operational conditions, failure modes and past operating experience) coupled with specific independent (ie from the smart sensor/actuator supplier) confidence building activities.

1.15.2.11 The approach needs to be compatible with nuclear safety standards and current research, and needs to be graded by importance to safety.

1.15.3 Common Position

Functionality and categorisation

1.15.3.1 The make, model, version and application of the smart sensor/actuator subjected to assessment shall be clearly identified and it shall be confirmed that the type testing applies (ie that the smart sensor/actuator subject to previous tests was fully representative of the smart sensor/actuator being assessed). The licensee shall ensure that the installed smart sensors/actuators are the same versions as those subjected to assessment.

1.15.3.2 The licensee's procurement arrangements shall ensure that technology changes from conventional sensor/actuator to smart sensor/actuator are recognised.

1.15.3.3 The licensee shall demonstrate that the smart sensor/actuator has the functional and performance properties appropriate for the intended application.

1.15.3.4 The smart sensor/actuator properties necessary to satisfy the plant application requirements shall be identified and documented by the licensee. This documented identification is essential for two reasons: (i) in its absence, one would not know what the safety properties to be justified are; and (ii) the identification helps to circumscribe the evidence which is needed. (See also 1.2.3.4, and 1.13 if applicable).

Failure analysis

1.15.3.5 The potential failure modes of the smart sensors/actuators shall be identified. Their consequences on the safety of the plant shall be identified and shown to be acceptable (see 1.1.3.4). Special attention shall be paid to the interface between smart sensors/actuators and other components to which they are connected and to the possible side-effects that may occur at these interfaces.

1.15.3.6 When multiple use of smart sensors/actuators is employed (eg in redundant configurations), the risk of common cause failure is of special concern and shall be analysed. Note that failures due to software are not random and that replacing redundant hardware devices by redundant embedded software systems does not necessarily provide the reliability of the original design.

Assessment and qualification

1.15.3.7 The smart sensor/actuator's production process shall be compared to that of an applicable safety standard. The applicability of the safety standard shall be justified. Any gaps revealed by the comparison exercise shall be addressed by compensating activities and/or arguments (see also 1.4.3.9 d). If any of the safety standard clauses are not used, then the omissions shall be evaluated and justified.

1.15.3.8 The licensee shall undertake a programme of independent (ie from the smart sensor/actuator supplier) confidence building activities appropriate to the safety requirements. (These activities might include commissioning tests, static analysis, statistical testing and analysis of operating experience.) The requirement for confidence building measures and their extent may require agreement between the licensee and regulator.

1.15.3.9 For safety systems, the smart sensors/actuators software implementation that realises the properties mentioned in 1.15.3.4 above shall be subjected by the licensee to the same assessment (analysis, review and testing) of the final product (not of the production process) as new software developed for the application. If necessary, reverse engineering shall be performed. Should this assessment not be feasible in whole or in part, complementary and compensating valid evidence shall be sought in compliance with 1.15.3.10 below.

1.15.3.10 The compensating activities and/or arguments (eg additional tests and independent verification and validation) required to compensate for the gaps identified by the comparison activity (see 1.15.3.7 above) shall be determined and justified by the licensee in relation to the:

a specific nature of the missing evidence;

b properties identified under 1.15.3.4; and

c potential failure modes of the smart sensor/actuator identified under 1.15.3.5.

1.15.3.11 The compensating activities and/or arguments shall be shown to be adequate to address specific evidence deficiencies and to bring the specific credit needed to support the properties identified under 1.15.3.4. Sufficient evidence shall be available to justify:

a the quality of the production process (quality and verification and validation plans, pedigree and experience of designers and supplier, etc);

b the quality of the product (test and verification coverage, etc);

c the acceptability of the smart sensor/actuator in the light of its operational experience as witnessed by product returns and failure notices etc. (documentation, etc);

d that specific risks (eg common cause failure, unused existing software, flooding (overloading), etc) are not a concern.

1.15.3.12 When the software of the candidate smart sensor/actuator is qualified for its intended use, the safe envelope and limits within which it is acceptable to use the candidate smart sensor/actuator shall be identified and documented. Limits can include requirements on procedures for installation, periodic inspections, maintenance, training of staff, or assumptions on the environment and system in which the smart sensor/actuator is used, etc.

1.15.3.13 The assessments required here above shall be undertaken for each smart sensor/actuator based on the make, model, version, configuration and intended application (see also 1.4.3.3). Reliance should not be based on a generic assessment unless this can be shown to be adequate for the specific application. Evidence from a previous qualification or a generic pre-qualification shall be given careful consideration as to the limits of its applicability to the current application.

Documentation and other requirements

1.15.3.14 The licensee shall ensure that access to all information required to support the safety case is available to those with a need to have such access (eg licensee, regulator and authorised technical support organisations). This information shall be sufficient to assess the smart sensor/actuator to the required level of quality. For high integrity applications this could include access to source code (see also 1.4.3.8 and 1.15.4.1).

1.15.3.15 For smart sensors/actuators embedded in safety systems, the licensee shall have full access to information on the software production process (see 1.15.4.1 for those cases where this access is limited).

1.15.3.16 Common positions 1.4.3.5, 1.4.3.7, 1.4.3.9 a to c, 1.4.3.9 e, 1.4.3.10 and 1.4.3.11 shall apply without change to smart sensors/actuators.

1.15.3.17 More generally, the following common positions apply:

Chapter 1.1 on safety demonstration:

Common positions 1.1.3.1 to 1.1.3.11 (note that references to "system" in these common positions should be interpreted as meaning "smart sensor/actuator").

Chapter 1.10 on independent assessment:

Common positions 1.10.3.1 to 1.10.3.8.

Chapter 1.11 on graded requirements for safety related systems (new and pre-existing software):

Common position 1.11.3.3.

Chapter 1.12 on software design diversity:

Common positions 1.12.3.1 to 1.12.3.13.

Chapter 1.14 on use of operating experience:

Common positions 1.14.3.1 to 1.14.3.9.

Chapter 1.16 on programmable logic devices:

Common positions 1.16.3.1 to 1.16.3.14, if the smart sensor/actuator contains one or more PLDs.

1.15.4 Recommended Practices

1.15.4.1 The recommended practices of chapter 1.4 apply.

1.15.4.2 Recommended practices 1.15.4.3 to 1.15.4.5 are only applicable where reliability targets have been set for the computer-based systems[2].

1.15.4.3 Common position 1.15.3.7 above requires access by the licensee to the production process of the smart sensor/actuator. If reliability targets are used, different levels of access to the source code can be catered for using a pragmatic graded approach dependent on the required reliability. If the required reliability for the software in a safety system or safety related system is proven to be sufficiently low, ie where the probability of failure on demand is greater (ie less onerous) than 10^{-2}, access to the source code may not be required.

1.15.4.4 If reliability targets are used and a probability of failure on demand of 10^{-2} is required, access to and analysis of the source code and circuit information is expected. However, this may not always be possible and typically, three options are available, presented here in order of preference:

a access to source code obtained:
  - assess in accordance with the common positions shown above,
  - apply static and dynamic analysis to the firmware.

b access to source code not obtained:
  - present evidence to show attempts to access data have been unsuccessful and that there are no dumb or alternative justified smart sensor/actuators are available,
  - assess in accordance with the common positions shown above,
  - perform statistical testing.

c diverse instruments used (and access to source code not obtained), each of which meet the requirements for systems with a probability of failure on demand greater than 10^{-2}:
  - provide a justified worst-case estimation of the probability of failure on demand of each diverse instrument,
  - assess each smart sensor/actuator as previously described, providing compensating evidence where gaps are identified,
  - provide explicit arguments of diversity (see chapter 1.12),
  - audit the manufacturers to support the diversity arguments.

[2] These recommended practices are not applicable in Germany.

1.15.4.5 If reliability targets are used and a probability of failure on demand of 10^{-3} is required, all common positions, including analysis of the source code and circuit information, shall be rigorously applied.

* * *

1.16 Programmable Logic Devices

1.16.1 Rationale

A programmable logic device (PLD) is an integrated circuit with logic that can be configured by the designer to implement the required functionality. The connections between logic elements may be configured, and in some cases reconfigured whenever required, to build the desired logic. This chapter is focussed on PLDs that are custom-built for the nuclear safety application. A PLD may be embedded in an instrument, control, field device, electrical device, component, or system.

PLDs in this context exist in a range of capabilities and configurations. There is an increasing-capability trend with correspondingly increasing V&V challenges. Examples include:

- a standalone single-input single-function single-output device – standalone.
- a multi-input multi-function multi-output device, initially standalone, which could have any of the following features cumulatively added:
  - many logic paths in concurrent execution,
  - test and diagnostic inputs and outputs,
  - network capability,
  - the ability to reconfigure in the field (which can change the execution paths),
  - internal clock and global clock synchronization,
  - an embedded microprocessor.

The memory element is used to configure the logic circuit and define its functionality. The memory may be write-once, non-volatile or volatile. Configuration methods used include Antifuse, SRAM (static random-access memory), EPROM (erasable programmable read only memory) or EEPROM (electrically erasable programmable read only memory) cells, and Flash memory.

A PLD may have different kinds of fabrication base, eg PLAs (programmable logic arrays), PAL (programmable array logic), CPLDs (complex PLDs), FPGAs (field programmable gate arrays) and custom microprocessors.

The different types of memory and configuration, and different fabrication bases, each have corresponding associated hazards.

ASICs (application specific integrated circuits) are customized for a particular use, eg mass produced for a non-nuclear application such as digital voice records, and are not configurable after production. Therefore, ASICs are not PLDs (as defined in this document) and are not considered in this chapter. However, some of the common positions may provide helpful guidance for an ASIC implementation.

PLDs can have different degrees of programmability, entailing different amounts of hazardous conditions. For example, the hazard space is relatively small in a once-programmable, fixed-configuration PLD, but it is larger if the configuration can be changed without changing behaviour (eg by changing the device address), and is even larger if the program can be altered through configuration (eg bit pattern) changes.

A reference process for implementing the PLD design is described in Figure 1. This uses a hardware description language (HDL) to encode PLD behaviour.

Despite perceptions, a PLD is not just hardware, but also includes encoded logic (a form of software). Therefore all the issues usually associated with software development apply to PLDs. Table 1 includes examples showing the application of common positions in other chapters to PLD-specific issues.

While the choice of a PLD over a traditional microprocessor-based system may be motivated with the desire to avoid certain verification difficulties and security vulnerabilities in the latter, its less constrained engineering environment (ie increased customizability) could introduce its own set of difficulties in verifiability and comprehensibility (otherwise referred to as complexity). For example, a soft-core microprocessor may be integrated within a PLD to enhance its configurability.

1.16.2 Issues Involved

1.16.2.1 Because PLDs may be embedded in instruments, controls, field devices, electrical devices, other components and larger systems, PLDs might be present in a nuclear power plant without the explicit knowledge of those responsible for its safety.

1.16.2.2 An asynchronous circuit introduces verification challenges at two levels:

– Asynchronous designs are prone to metastability, bus skews, and other timing issues.
– Currently available engineering tools do not generally support asynchronous timing constraints specification and analysis.

1.16.2.3 The timing performance of a PLD will vary according to external factors, such as temperature and voltage. In a microprocessor, this will be catered for by the software (eg microcode). The PLD design will need to include equivalent functionality, such as clocking, to address this variance.

1.16.2.4 Reuse of sub-components, such as Intellectual Property (IP) cores from vendors and/or third-party suppliers, to speed up the design process and reduce the design costs introduces hazards through very limited transparency, limiting verifiability and maintainability.

1.16.2.5 Use of a programming language that is not formally defined also creates verification challenges.

1.16.2.6 If the execution environment (eg hardware platform) is being engineered concurrently, there is a need to ensure consistency with the logic, requiring especially rigorous configuration control, change management, and change impact analysis.

1.16.2.7 The internal signals and states in an actual hardware circuit may not be observable unless explicit provisions are included to make them accessible to make the PLD verifiable.

1.16.2.8 The development (including verification and validation) of a PLD design heavily depends on tools specific to the implementation environment, entailing corresponding hazards:

– the tools are very complex and the electronic design automation vendors do not provide certified versions of the tools for nuclear safety applications.
– the tools are evolving at a rapid pace, but their qualification is not - the burden to ensure suitability falls on the PLD developer.
– a development toolset will consist of a number of different tools that need to work together. This introduces challenges in configuration management and impact analysis as the tools evolve.
– proficiency is needed in the correct, hazard-free use of the tools; but it is difficult to maintain proficiency with the low-volume usage in the nuclear industry.

- some synthesis tools use optimisation which may negate design objectives such as transparency, analysability and verifiability. For example, a tool may not fully account for unused functionality on the PLD, which makes it harder to ensure that unused functions do not interfere with the defined functions.

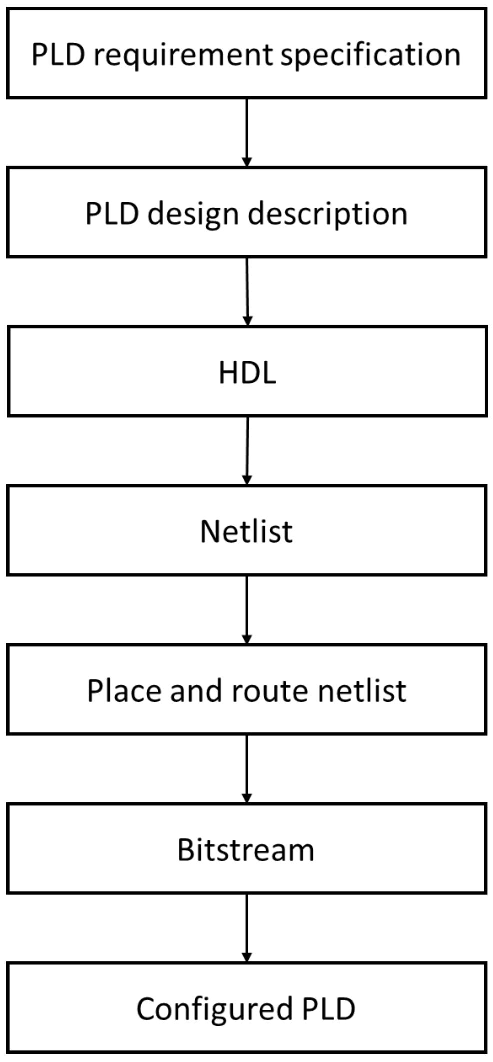
1.16.2.9 Cyber security related hazards remain during the development phases (this is the same issue as for conventional software, but may be less recognised).

1.16.2.10 The development process for PLDs increases the potential for the introduction of systematic errors. Figure 1 illustrates a typical development process with the following steps.

- The functional requirements and associated properties are refined into a design description, in a similar way to software development. The requirements and design need to account for those properties more prevalent in PLDs, such as concurrency through parallel logic paths.
- The design is realised using HDL. Typical industrial designs use third party intellectual property (IP) cores (equivalent to software libraries). This introduces issues similar to the use of pre-existing software (see chapter 1.4). It is common to generate HDL automatically using tools, which introduces issues similar to conventional automatic code generation (see 1.5.3.8).
- The HDL description is synthesized (like software is compiled) to produce a netlist.
- The netlist is translated to a device-specific place and route netlist by selecting the components on the chip to be used and determining the connections between them.
- The place and route netlist is converted to a bitstream, which is transferred to the PLD. This implements the design on the chip, giving a configured PLD. It is difficult to verify that the bitstream is an accurate representation of the place and route list, because production of the bitstream uses proprietary information not normally accessible to the developer. Similarly, it is difficult to verify that the configured PLD matches the bitstream generated from the place and route netlist, ie that the transfer and PLD configuration occurs without any error. This will place greater reliance on validation of the configured PLD.

1.16.2.11 Simulation is useful for verification and validation at multiple stages in the design process, but, as in testing, it can only reveal the presence of an anomaly or defect, but cannot assure its absence. Consequently, preventative engineering methods appropriate for PLDs are needed.

Figure 2: PLD development – reference lifecycle



1.16.3 Common Position

1.16.3.1 All PLDs that can affect nuclear safety shall be identified, and their safety shall be demonstrated by satisfying the common positions below, with special consideration to the issues identified in 1.16.2. For examples of PLD capabilities and configurations, see 1.16.1.

1.16.3.2 The PLD-specific aspects of the safety demonstration plan shall cover all the applicable common positions (ie the common positions stated in this chapter including 1.16.3.14), in order to address all the issues identified in 1.16.2).

1.16.3.3 The use, selection and configuration of the PLD for the safety system application shall be demonstrably fit for purpose, based on the allocated requirements, eg including requirements for the following characteristics:

a capability to carry out safety functions with the required level of performance

b avoidance of unnecessary complexity (ie characteristics that limit verifiability and comprehensibility and preclude a credible safety demonstration plan)

c predictability and determinism in behavior, including timing

d explicitly identified underlying assumptions, eg about the execution environment of the item

e facilitating adequate verification.

1.16.3.4 The ability of the PLD to perform its functional and non-functional requirements shall be demonstrated.

1.16.3.5 If the PLD contains an embedded microprocessor, all common positions in this document shall be considered and applied to the embedded microprocessor software, depending on the extent of its functionality and safety dependence. Any relaxation from these common positions shall be justified.

1.16.3.6 The set of requirements specification techniques used shall be adequate to demonstrate that the requirements are complete, correct, consistent, and unambiguous (see also 1.1.3.9, 1.1.3.11, 1.4.3.1, 1.4.3.2, 2.1.3.2, 2.1.3.4 and 2.3.3.1), and may include for example:

a formal specification of behaviour, including order of execution and timing (see also 1.9)

b specification of constraints to prevent hazardous behaviour.

1.16.3.7 The set of V&V techniques used shall be adequate to demonstrate safety (see also 2.5.3.4), and may include for example

a formal verification (see also 1.9)

b behavioural simulation

c logic simulation (including for example metrics for the coverage of all combinations of inputs, internal states, internal execution paths, and states of the PLD logic execution environment

d post layout simulation

e (static and dynamic) timing analysis, including delay modelling and path delay calculation

f timing uncertainty analysis and deriving margins to preserve correctness

g formal equivalence checking

h black box testing.

1.16.3.8 Timing requirements and constraints shall be specified, validated and verified, including:

a identify potential timing problems (eg metastability, slow paths and clock skew) and provisions to protect against their effects

b verify that all signals arrive on time, accounting for potential sources of variation (eg temperature, voltage, manufacturing differences).

1.16.3.9 If an asynchronous design is used for any reason, appropriate measures shall be taken to make sure that any timing issue does not affect the safe operation of the design.

1.16.3.10 Verification inputs, eg simulated input stimuli, shall be semantically equivalent and consistent across various verification activities performed at various development phases, including the following: inputs used for software-based simulation (eg behavioural simulation, logic simulation and post-layout simulation) and for prototype device verification (see 2.5.3.4 and 2.6.3.16).

1.16.3.11 The execution environment for the logic within a PLD shall be explicitly identified, specified, and configuration-controlled. It includes all items, eg hardware, IP cores, libraries, and resources on which the correct, timely execution of the logic is dependent. Note that if a system is replaced by a new system containing a PLD, it may be necessary to re-characterize the environment of the PLD, because its earlier characterization might not have been appropriate and adequate for this technology. The re-characterization might require tests and experiments.

1.16.3.12 If a reliability target is set as a probability of failure for a PLD, the requirements of 1.13.3 apply.

1.16.3.13 Personnel engaged in the development of a PLD and logic therein shall have the requisite competence. For example, the PLD designer and verifier shall both have the necessary competence to perform appropriate timing analysis, including awareness of common timing problems in PLD logic.

1.16.3.14 The following common positions also apply, in which case (computer-based) systems include PLD-based systems:

Chapter 1.1 on safety demonstration

All common positions in 0

Chapter 1.4 on pre-existing software

All common positions in 1.4.3, if the PLD contains pre-existing software, eg libraries and IP cores, or its logic is generated from pre-existing software. Note that the execution environment is part of the interface to the PLD.

Chapter 1.5 on tools

All common positions in 1.5.3 apply to PLD tools.

Chapter 1.8 on security

All common positions in 1.8.3.

Chapter 1.11 on graded requirements for safety related systems (new and pre-existing software)

Common position 1.11.3.3.

Chapter 1.14 on use of operating experience

All common positions in 1.14.3.

Chapter 1.15 on smart sensors and actuators

All common positions in 1.15.3, if one or more PLDs are contained within the smart sensor/actuator.

Chapter 2.3 on software requirements, architecture and design

Common positions 2.3.3.1, 2.3.3.2, 2.3.3.3, 2.3.3.6, and 2.3.3.9 to 2.3.3.16, applied analogously for PLDs. All common positions in 2.3.3 apply to any embedded microprocessor.

Chapter 2.4 on software implementation

Common positions 2.4.3.2, 2.4.3.3 to 2.4.3.7, 2.4.3.11, 2.4.3.26 to 2.4.3.31, 2.4.3.35, 2.4.3.37 to 2.4.3.40, applied analogously for PLDs (for example code equates to HDL and where compiler equates to a tool, and associated libraries, to transform the PLD software/configuration from a higher level to a lower level language). All common positions in 2.4.3 apply to any embedded microprocessor.

Chapter 2.5 on verification

Common positions 2.5.3.1 to 2.5.3.9 and 2.5.3.22 to 2.5.3.29, where this applies to every level of integration within the PLD.

Chapter 2.7 on change control and configuration management

All common positions in 2.7.3.

1.16.3.15 Chapters 1.2, 1.6, 1.7, 1.10, 1.12, 2.1, 2.2, 2.6 and 2.8 will still apply if the computer-based system is actually a PLD based system.

1.16.4 Recommended Practices

1.16.4.1 Advanced simulation techniques that should be considered include constraint-based random stimuli generation (related to expected input conditions), assertion-based verification, and formal verification together with HDL and/or logic path coverage analysis.

1.16.4.2 To increase the confidence in a design, the input stimuli should not only cover the expected input combinations but also those input combinations that are not expected in normal use of the design. Fault injection techniques can also be used to change the internal state of the design to simulate single event effects.

1.16.4.3 Asynchronous designs are prone to timing issues such as glitches, bus skews, and metastability. Design tools do not generally support asynchronous timing constraint and analysis. For the purpose of better timing control, the design should be synchronous where possible. Appropriate measures to ensure that any timing issue from an asynchronous design does not affect safe operation (see 1.16.3.9) includes resynchronizing the asynchronous reset signal to avoid metastability when the reset is released. In addition, synchronisation chains can be implemented to synchronise any asynchronous input signal.

1.16.4.4 Libraries of predefined functions and circuits with appropriate performance and properties should be employed wherever applicable to avoid having to recreate previously developed logic. These should be demonstrated to be appropriate for the safety application, for example by analysis and testing.

1.16.4.5 The design tools used for HDL programming and synthesis should not make use of any optimisation features that may be available. The use of optimisation features may cause the circuit described by the HDL code to differ from the netlist, which will make verification more difficult.

1.16.4.6 The FPGAs should be programmed using HDL that conforms to the relevant standards (eg IEEE Std 1076 for VHDL, IEEE Std 1364 for Verilog, and IEEE Std 1800 for SystemVerilog). Extensions which are not part of those standards should be avoided.

1.16.4.7 The programming procedures and standards used for design and simulation should provide unambiguous and safe instructions for the relevant tools. Code that is difficult to get materialized into logic or leads to an unsafe state should be avoided.

1.16.4.8 Further guidance on PLDs is available in [25], [29], [30] and [31].

* * *

1.17 Third Party Certification

1.17.1 Rationale

Certification generally asserts that a computer-based system or device has certain properties, and/or has been developed according to a particular process or standard. A certification is not a safety demonstration since it is not specific to the given application or plant environment. However, certification by a third party (ie a party (sometimes accredited) that is neither the manufacturer of the device, nor the licensee, nor the regulator) may provide useful evidence that a computer-based system or device has properties generally necessary to ensure safety. This useful evidence can contribute to the safety demonstration for a particular nuclear application.

The aim of this chapter is to set out the issues that need to be addressed to maximise the contribution that third party certification provides to the safety demonstration. It also highlights pitfalls that can make these certifications less valuable or misleading.

1.17.2 Issues Involved

1.17.2.1 Although the basis for a certification can usually be determined (for example it could be compliance with a particular standard), a certification often asserts that a particular system or device is safe to use. Consequently, users of the certification, such as developers or even licensees, sometimes misunderstand and hence misapply it, for example by considering that a certified device is always safe to use in every application.

1.17.2.2 A certification may state caveats, context or limits to its assertion, but these are not always obvious, for example they are sometimes documented in a supporting report that is not immediately available.

1.17.2.3 Confidence in certification may rely on an acceptance of the competence, independence and rigour of the third party in making an effective judgment. Without supporting evidence and detail, it can be difficult for anyone making use of the certification to gain sufficient confidence that this reliance is justified.

1.17.2.4 The judgment of the third party may be based on assumptions and considerations, which if not visible cannot be checked against the requirements and expectations of the user.

1.17.2.5 The value of a third party assessment to the licensee can be increased if the licensee is able to provide a specification for the certification, and to exercise close control, to ensure that the resulting evidence matches regulator expectations.

1.17.2.6 A key problem with assessing pre-existing software is access to appropriate development documentation and source code. Certification may provide some help here, in that a supplier may be prepared to allow access once for many. But this is only useful if the certification assessment is sufficiently rigorous and transparent, and the properties being certified hold for all environments and applications within the scope of the certification.

1.17.2.7 The value of a certification significantly decreases if it was performed on a version of the software that differs from the version used in the safety application.

1.17.2.8 Most certification will have some value as supporting evidence; the difficulty is in deciding how much dependence to place on it in the safety demonstration.

1.17.3 Common Position

1.17.3.1 The certification shall give a clear definition of scope, including:

a boundaries of the system or device (eg inclusion of sensors and actuators)

b functionality

c interfaces

d assumed purpose (often associated with implicit or explicit assumptions about the limits on input and output conditions)

e environmental limits

f hardware and/or software covered (including version numbers)

g communications covered

h any other assumptions or limitations

i properties (eg reliability, availability, maintainability, security)

j assessment process or reference standard (including which parts of a standard and any scope limitations, eg restricted to particular lifecycle phases).

1.17.3.2 Any difference in scope between the certification and the system or device being used in the safety application (eg software version number, environment etc) shall be identified and its impact assessed. The safety demonstration shall include an explicit acknowledgement of all differences and a justification for the extent to which the certification is still considered valid and used as supporting evidence.

1.17.3.3 The certification shall describe what is being asserted in a way that can be verified, ie to facilitate review that the evidence does in fact support the assertion about the system.

1.17.3.4 The process leading to the certification judgment, and the associated evidence, shall be transparent to the extent necessary for the user and regulator to judge that it was appropriate and equivalent to accepted good practice in establishing that judgment (for example detailed compliance with relevant standards), and of sufficient breadth, depth and rigour. For example, the detailed certification report shall be available for review by the licensee and the regulator.

1.17.3.5 When contracting a third party certification, the user shall obtain an up-front agreement on the level of detail to be made available on the assessment process and the underlying evidence supporting the judgment.

1.17.3.6 If a certification covers a family or range of systems/devices, it shall make clear the basis on which the certification is applicable to the whole family/range (eg every code module and hardware combination was included in the assessment, rather than results being inferred based on similarity).

1.17.3.7 If the main aim of certification is to declare compliance with a particular standard, the standard needs to be appropriate for the application and the safety importance; for example matching the target application and properties, and an up-to-date standard matching regulator expectations.

1.17.3.8 If operating experience forms part of the certification approach, 1.14.3.4 to 1.14.3.8 apply.

1.17.3.9 The licensee shall have sufficient competence to understand the evidence generated by the assessment to be able to come to its own view of its adequacy. Certification is not an acceptable substitute for competence of the licensee.

1.17.3.10 When using certification results as evidence in a safety demonstration, the user shall describe how the certification is applicable and ensure that the reliance on it is proportionate and sufficiently complemented by other evidence, so that the safety demonstration is complete (see chapter 1.1).

1.17.3.11 For platform qualification, the common positions of chapter 1.18 shall also apply.

* * *

1.18 Platform Qualification

1.18.1 Rationale

The common positions in this chapter provide guidance on the software aspects for performing or evaluating a platform qualification for systems important to safety [32, 33]. They address the scope, methods and documentation of platform qualification, as well as the use and maintenance of qualified platforms for the development and implementation of computer-based systems. No guidance on hardware or system lifecycle specific aspects is included as this is outside the scope of this document.

A platform consists of an integrated set of pre-developed hardware and software components designed to work co-operatively to realise required functions using application specific software.

Typical hardware components include processors for logic, interfaces and communication. Typical software components include function block or application function libraries, user interfaces, operating systems and communication protocols.

The platform also includes tools that aid the development, programming, code generation, verification and validation of the application specific software. These help manage and utilise the application function libraries, and support maintenance and testing.

A computer-based system based on a platform has two levels of software: the application-independent platform software (including operating system, application function libraries and operator interfaces); and the application-specific software (plant-specific programs). Commonly, an application-specific system is created from a generic, configurable platform by selecting the needed components and libraries, integrating these into an application-specific architecture, and setting the values of configurable parameters.

The scope of this chapter is restricted to computer-based (microprocessor or programmable logic device (PLD) based) platforms, although there are platforms using other technologies such as hardwired systems.

Platform qualification is the confirmation, through the provision of objective evidence, that the platform is able to achieve its required safety objectives, within its specified conditions of use. Conditions of use include anticipated application requirements, system class, constraints and configuration bounds.

Commonly, because so many specific configurations can be derived from a generic platform, it is infeasible to qualify them all. For technical and economic efficiency, a platform is qualified for conditions of use determined by a set of anticipated applications. The specified conditions of use provide a defined boundary of application requirements, properties and all other relevant characteristics, which enables a platform user to determine whether an intended application is within the anticipated set. The platform qualification can be credited for a fully defined application within this boundary.

During qualification, a representative system is developed to demonstrate that the platform works as expected for a typical application. The attributes of this application will fall within the conditions of use of the qualification but does not cover all of them.

Qualification ensures all constraints on the application are explicitly stated in the conditions of use, and are acceptable. It also considers the adequacy of documentation, including instructions for using the platform components in building the anticipated applications.

Platform qualification facilitates reuse and standardisation, and can take account of documented operational experience (see 1.14), which will accumulate from further reuse.

The team (often a third party) carrying out platform qualification may be more independent than the project qualification team. Tools and libraries developed elsewhere can also be qualified, independently of the platform. Application-specific qualification is still necessary, but the need to retest and analyse the platform is substantially reduced.

This chapter provides guidance on qualification of a platform and how a platform qualification performed for an anticipated application may contribute to a safety demonstration for a particular system operating in a specific environment (see 1.1).

Many common positions in other chapters of this document also apply to issues involved in platform qualification, or to the use of qualified platforms. Instead of repeating them, this chapter emphasizes those aspects that are particularly important for platform qualification and use.

1.18.2 Issues Involved

1.18.2.1 All platform users (including the system owner, system integrator, system developer and application developer) need to know that their chosen platform will adequately support the intended application. This is difficult for a platform intended for a broad range of application sectors, partly because there are many configurations. The platform user needs criteria to determine that the intended application is within the set of anticipated applications specified by the conditions of use.

1.18.2.2 A platform cannot be qualified without the context of its intended use. The plant specific applications may differ between nuclear power plants. The platform qualifier needs to anticipate these applications because full specifications are generally not available at this point. It can be difficult to specify conditions of use for all the plant specific applications.

1.18.2.3 It is not sufficient to consider platform qualification and plant-specific aspects separately. Certain aspects, such as communication load, can only be fully evaluated by considering the interaction of the platform and the application-specific environment.

1.18.2.4 Sometimes platform qualification is performed concurrently with application development. While this provides opportunities for increased efficiency and easier integration, it also introduces uncertainties due to delayed confirmation of the properties of the platform and its conditions of use.

1.18.2.5 New platforms may be developed according to nuclear standards from the outset, or existing platforms originally developed according to non-nuclear standards may be refined or re-qualified to meet nuclear standards. Platform qualification requirements need to account for both scenarios.

1.18.2.6 The requirements against which a platform is qualified may be incompatible with the application-specific requirements. For example, environmental conditions and/or constraints may differ.

1.18.2.7 New issues will arise in the application that will need addressing in the safety demonstration, in addition to the platform qualification

1.18.2.8 For third-party tools, or pre-developed software that is part of the platform, it may be difficult to demonstrate that the requirements of every common position have been met.

1.18.2.9 Due to the high modularity and flexibility of platform software and hardware, configuration and version control of platform components plays an important role.

1.18.2.10 The potential value of a large base of operating experience from reuse will only be realised if the issues raised in 1.14.2 are adequately addressed during qualification and its use.

1.18.2.11 If the same platform is used across the computer system architecture for otherwise independent applications, platform-related mishaps may increase the potential for common cause failure, compromising the intended independence.

1.18.2.12 Changes to the platform hardware or software can introduce hazardous or hazard-contributing conditions. Due to the high versatility of a platform, such changes can introduce hard-to-discover incompatibilities.

1.18.2.13 Many commercially available platforms are targeted to support safety classes lower than safety systems. This can introduce complexity that makes the platform harder to qualify for safety systems.

1.18.2.14 It can be difficult to access all the necessary evidence for a safety demonstration (such as development documentation) from the platform supplier.

1.18.2.15 Platform qualification has some issues in common with third party certification (see 1.17.2).

1.18.3 Common Position

Performing platform qualification

1.18.3.1 Since a relevant platform qualification will be used to support a safety demonstration, the qualification of the platform shall satisfy the common positions for a safety demonstration (see 0). The platform qualification shall demonstrate that the platform is able to meet the conditions of use.

1.18.3.2 The common positions of chapter 1.17 (third party certification) shall apply. Since platform qualification is carried out with nuclear requirements for safety already in mind, the common positions of chapter 1.17 will in general be satisfied in addressing the common positions of this chapter and chapter 1.1, but still provide additional perspective and clarification.

1.18.3.3 The scope of platform qualification shall be defined, such as architecture, configuration, components and any constraints (see 1.18.3.10 for more detail). Tools shall also be qualified appropriately (see 1.5). All software, including firmware and configuration data of PLDs (see 1.16), shall be explicitly covered.

1.18.3.4 All components (and subcomponents on which the components are dependent) of the platform shall be uniquely identifiable and be under configuration control to clarify the validity of the qualification.

1.18.3.5 Platform qualification shall specify the highest safety class it is designed to support.

1.18.3.6 The qualification documentation shall clearly identify the criteria, including environmental conditions, to which the platform is qualified (eg dependability, temperature, electromagnetic compatibility and humidity). Test cases shall include coverage of all requirements and environmental conditions to which the platform is claimed to be qualified.

1.18.3.7 For any pre-developed software being part of the platform, the common positions of chapter 1.4 shall apply.

1.18.3.8 For any tools that are part of the platform, the common positions of chapter 1.5 shall apply.

1.18.3.9 If operating experience is applied to the qualification, the common positions of chapter 1.14 shall apply

1.18.3.10 The platform qualification shall consider its expected applications, and define the conditions of use for which the platform is qualified, including supported requirements for the application specific software, user instructions, constraints and potential implications (including information identified in 1.18.3.11 to 1.18.3.14).

1.18.3.11 Comprehensive information covering both process and product aspects needed to perform platform qualification shall be provided. As a minimum, the following information is required:

a overall description of the platform, including all its components (and subcomponents on which the components are dependent);

b specification of platform properties, application constraints (including environmental conditions) and supported requirements;

c acceptance and performance criteria;

d conceptual design and manufacturing drawings, and schematics of circuits, components, sub-assemblies and all other relevant constituents, with descriptions and explanations necessary to understand their operation;

e documentation and artefacts generated during development, including all verification and validation records such as test documentation at component, integration and system level;

f architecture of the platform covering hardware and software components (and subcomponents) including their versions, interfaces and behaviours;

g information on software implementation (the level of access to the code of components that is necessary for evaluation depends on the safety class of intended use);

h information on the software development environment, procedures and software tools;

i information on security aspects (see 1.8);

j user documentation;

k records of product maintenance, product failures and other relevant incidents (necessary for evaluation of operating experience, see also 1.14);

l quality assurance records, including verification and validation reports;

m evidence of qualifications and competence;

n additional third party certification, if any (see 1.17);

o information on configuration management (for example architecture and component versions), sufficient to determine if it matches the basis of any existing qualification;

p information on error tracking and notification, sufficient to ensure that errors reported with respect to the qualified platform are reported to all platform users.

1.18.3.12 A representative system based on the platform shall be implemented as part of platform qualification. The representative system shall cover typical applications of the platform. It shall include platform features, constraints and requirements to demonstrate suitability of the platform for a typical intended plant-specific application and environment. The demonstration of suitability does not cover all conditions of use.

1.18.3.13 When the platform qualification is performed concurrently with application development, opportunities shall be considered for communication, shared personnel and co-ordinated actions, to ensure that they are integrated effectively and to confirm that no gaps remain.

1.18.3.14 Platform qualification shall address platform properties essential for the anticipated plant-specific applications, including

a initialization and cyclic operation;

b response time;

c self-test or self-diagnostic capabilities;

d signal status flag processing;

e error handling and fail-safe behaviour;

f sufficient lifetime (in terms of availability of maintenance support and replacement parts);

g expandability (reserve capacities);

h portability;

i maintainability;

j usability and human factors aspects;

k accuracy;

l deterministic scheduling;

m capacity of data processing.

n reliability, if applicable (see 1.13).

1.18.3.15 Maintenance of platform qualification shall follow a stringent change management process. A clear strategy shall be defined for requalification of the platform when changes to a hardware or software component or configuration are made. Every change requires an impact analysis to determine the extent to which the platform needs requalifying. Compatibility with other components shall be considered as part of this analysis. Requalification (which may include regression testing) shall follow an effective change management process. Particular emphasis shall be given to retaining compatibility with other components of the platform.

1.18.3.16 The common positions in this whole document shall be used to determine requirements to be demonstrated by the platform qualification.

Using a qualified platform

1.18.3.17 A qualified platform shall only be used if it has been found to be suitable for the development and implementation of the required systems. This shall be achieved through detailed comparison of the application conditions against the documented platform qualification conditions of use (produced by satisfying 1.18.3.10 to 1.18.3.14).

1.18.3.18 The safety demonstration (see 1.1) for a system that uses a qualified platform shall ensure that all requirements necessary to achieve safety have been met. This will utilize the evidence provided by the platform qualification, together with additional evidence required by a safety demonstration plan to show that the platform is suitable for the application, and application-specific aspects not covered in the platform qualification.

1.18.3.19 The safety demonstration of a system that uses a qualified platform shall include consideration of the scope of the existing platform qualification, the results based on analysis of the existing documentation, available feedback from operational experience and its intended use.

1.18.3.20 A programme of supplementary activities shall be defined, including functional and environmental testing, addressing any differences in scope, the application-specific properties and requirements not already covered by platform qualification, and additional shortfalls as they become evident. This comprises consideration of the final configuration of the system used in the plant, including mounting arrangement, as well as load and temperature distribution inside the cabinets.

1.18.3.21 The programme of activities shall include a comprehensive test strategy that makes appropriate use of the test results of the platform qualification when testing and demonstrating the safety of the system configuration.

1.18.3.22 The requirements of the system shall be traceable to the requirements, and the results of verification and validation, of the platform and its components.

1.18.3.23 The safety demonstration of a system that uses a qualified platform shall identify all relevant issues due to the interaction of the platform-specific and the application-specific environment. Such issues include

a conditions specific to the environment;

b interactions between the platform and the application software;

c interactions between the platform and its environment;

d application-specific configurations;

e application-specific constraints on the platform, including the development environment, such as access controls;

f any effects from sharing resources during runtime, including malfunction or misbehaviour of tasks that access a shared resource;

g the worst-case workload exerted by the application and the system environment on memory, timing and other resources (eg resulting in deadlocks and priority inversion);

h effects of faults, failures, and other mishaps or misbehaviours (including common cause failures of the platform across the computer system architecture for otherwise independent applications).

1.18.3.24 Platform features and elements that increase system complexity and that are not needed to satisfy the system requirements shall be identified and justified with respect to safety.

* * *

PART 2: LIFECYCLE PHASE LICENSING ISSUES

2.1 Computer-Based System Requirements

Även en liten tuva kan stjälpa ett stort lass

(Old Swedish proverb)

2.1.1 Rationale

The development of the computer-based system requirements (for short called hereafter system requirements) assumes that certain activities have already been completed. It is assumed that the system concept of the facility and associated systems has already been established and that the associated analyses have resulted in certain requirements being placed on the computer-based system. In addition, national regulations will also impose certain functional or performance requirements on the computer-based system.

The system requirements are derived from the plant safety analyses, other relevant analyses and the applicable regulations. They are considered here as being either functional (eg the application functions to be implemented on the computer) or non-functional (qualities expected from the computer implementation such as reliability level, accuracy, time response). They should be implementation independent and should deal with the computer-based system as a black box. While the system requirements are expected to be decoupled from their implementation to the extent feasible, the non-functional requirements may also be supported with design and implementation constraints imposed for various justifiable reasons.

The elicitation of the system requirements is a critical step in the design. They are the result of co-operative work between experts of different disciplines and, as such, are prone to ambiguities, misunderstandings, inconsistencies and defects. Experience shows that these defects can have an important impact on all subsequent stages of development, and can be difficult to correct at later stages.

2.1.2 Issues Involved

If the system requirements and their documentation are not understandable to all parties involved, including those in charge of writing the computer system hardware and software specifications, the suppliers, and those involved in verification, validation and licensing activities, they are less likely to be met. Likewise, the description must be unambiguous and accurate, and this implies a certain degree of formalism. A compromise which preserves safety must be made between understandability and formality, which can conflict with each other [20].

Transforming intended objectives into consistently implementable, verifiable, comprehensible requirements generally entails conservative approximation to allow for uncertainty. Unnecessary conservatism can compromise overall safety because resources are not optimally allocated, and can unnecessarily impede non-safety objectives. Hence, any advances that reduce uncertainty, such as in requirements validation and verification, will improve the design.

2.1.3 Common Position

2.1.3.1 The system requirements shall be validated, ie their correctness, consistency, completeness, accuracy and absence of ambiguities shall be established in relation to the results of a prior plant safety analysis, and other relevant analyses at the plant level, in accordance with 1.1.3.5, 1.1.3.6, 1.1.3.9 and 1.1.3.16. The traceability of the system requirements to the relevant analysis shall be documented.

2.1.3.2 The system requirements shall be established based on the applicable regulations, standards and guidelines.

2.1.3.3 The system requirements documentation shall at least cover the following items:

a A precise definition of the system boundaries, interfacing systems (including human-machine interface), and interface requirements (including the interactions needed between the system and its environment). It shall, in particular, include the documentation of the restrictions and constraints placed on the system by compatibility requirements with other existing systems in the plant, by physical and natural environment constraints, by protection against propagation of non-safety system failures.

b A specification of system requirements – functional and non-functional (see 1.1.1) – required from the system.

c The results of hazard and failure analyses performed at the system concept level. In particular, the results of the analysis shall include required responses to hazardous conditions at the system interface with the plant such as sensor failures and input variables being out of range. Potential incorrect system outputs shall be identified, their impact on the plant and environment evaluated, and the presence of adequate safety defences confirmed.

d A precise definition of the input variables to be monitored by the system (process variables, operator signals) and of the output variables to be controlled (output to actuators and indicators), together with a specification of the valid ranges of these variables. Criteria to validate the values of the input variables shall be specified. In some cases, the validation criteria need to include limits on the rate of change of a variable.

e The specification of special engineered features where there is a need to change specific system parameters such as calibration constants for operational reasons.

f The results of the validation required in 2.1.3.1.

g All modes of operation (eg power operations, refuelling, post accident monitoring) and human interactions (eg operation, maintenance, tests).

2.1.3.4 The documentation of the system requirements and of their demonstration of correctness, completeness and consistency (requirement validation), shall be made available to the regulator as early as possible in the project.

2.1.3.5 The system requirements shall ensure that correct completion of a safety function will be monitored and any detected violation against the monitoring criteria will be notified to the plant operator.

2.1.4 Recommended Practices

2.1.4.1 When practical, the system requirements specifications should be simulated to aid understanding and communication.

2.1.4.2 The system requirements should be informed by a library or catalogue of hazardous conditions (previously experienced, or analysed and identified).

* * *

2.2 Computer System Architecture and Design

2.2.1 Rationale

The means by which the computer system architecture is demonstrated to be adequately safe and to meet the computer-based system requirements (functional and non-functional) must be addressed at this stage.

The methodical and systematic derivation of the requirement specifications for this architecture is important for the safety demonstration. It is not a trivial task, and it must be addressed carefully and properly. To a large extent, the design process is an empirical and heuristic process based on past experience gained in previous designs. In addition to safety, new designs are driven by technological and commercial considerations (such as use of a new, more efficient or more reliable technology, better configurability and easier maintenance).

For an adequate safety demonstration, the proposed computer system architecture should be traceably derived from the system requirements with documented arguments and rationale for the choices being made.

Executable software components can be of different types: operating system, library, application software, intelligent drivers, sensing and actuation devices, communication protocols, human-machine interfaces, etc. These components do not contribute in the same way to the safety of the whole system. Defence in depth principles should be applied to the design of these different components: a component (for example the application software) can be designed so as to mitigate potential unsafe failures of the components it is using.

A well-designed system architecture facilitates and utilises fault tolerance. This mitigates the effect of faults by preventing the faults from turning into failures that jeopardise system safety. Diversity in design and defence in depth are examples of passive fault tolerance. Protection systems provide active fault tolerance, which requires the right detection, control and decision mechanisms, and appropriate interconnections. The effectiveness of fault tolerant mechanisms depends on the knowledge of the faults to be protected against. Fault tolerance works best when reliable components are employed, when the defect rate is already low, and the fault mechanisms are well identified. It is important to recognise that the failure of a fault tolerant mechanism can itself jeopardise system safety if not adequately mitigated.

2.2.2 Issues Involved

2.2.2.1 The computer system architecture design has to be shown to result from a consideration of alternatives and from a mapping (systematic, structural or formal) of the system requirements onto the selected solution for:

- hardware or programmable logic devices,
- purchased software,
- software to be developed,
- software to be produced by configuring pre-developed software.

2.2.2.2 At this stage, the specification documents typically need to incorporate additional requirements relating to:

- equipment qualification (for all environment conditions), including electromagnetic compatibility and seismic qualification,
- communication systems, multiplexers, network protocols,
- fault detection and self-supervision,
- hardware/software fault tolerance mechanisms,
- testability, in service testing and maintenance, self-testing
- systems outside the computer-based system boundaries which require design decisions (eg instrumentation requiring software compensation or filtering),
- human interfaces,
- security,
- constraints on subsequent lifecycle activities and work products.

2.2.3 Common Position

2.2.3.1 The computer system architecture, with its centralised versus distributed organisation, and its derivation and allocation of functions to either hardware or software shall be demonstrated to result from the system requirements and from the added requirements mentioned above in issue 2.2.2.2.

2.2.3.2 The principles of redundancy, diversity, physical separation, electrical isolation and independence between safety functions, safety related functions and functions not important to safety shall be applied to the design of the computer system architecture. The contribution made by each principle to meeting the requirements – including design criteria such as defence in depth and the single failure criterion – shall be documented. In particular, the non-functional requirements, including reliability, shall be explicitly addressed.

2.2.3.3 The boundaries between components associated with different safety classes shall be well defined. Failure propagation across the boundaries from a component of lesser criticality to a component of higher criticality shall be prevented.

2.2.3.4 The design shall achieve the required accuracy and response times to satisfy the system accuracy and real-time performance requirements. In particular, attention shall be paid to the cycle times, signal sampling and processing functions that will be implemented.

2.2.3.5 It shall be confirmed that the design includes an explicit, verifiable specification for the order of execution and timing inter-relationships, especially considering multiple concurrent physical processes, inter-process synchronization and shared resources.

2.2.3.6 The design shall ensure that failure of a part of a redundant safety system shall not adversely affect other redundant or common parts of this safety system.

2.2.3.7 The system shall be designed so that all hardware faults which may affect safety or reliability can be detected, by self-testing or by periodic testing, and are appropriately communicated. In the presence of faults, the system must move to a subset of states which can be demonstrated to satisfy the system safety requirements. This shall be true even in the presence of an external fault like a loss of electrical power to the system or to its input/output devices.

2.2.3.8 Computer system architecture failure and hazard analyses shall demonstrate that:

a the possible failure modes of the architecture that may compromise the safety functions have been taken into account, and

b adequate exception handling mechanisms and hazard mitigating functions have been included in the design.

2.2.3.9 It shall be ensured that the use of these fault tolerant, exception handling and hazard mitigating mechanisms is appropriate and that they do not introduce unnecessary complexity.

2.2.3.10 The computer system architecture shall be designed so that it is periodically testable in service without degradation of reliability. Test periodicities shall be derived from the system reliability and availability requirements. Hardware faults shall be postulated, even those that will remain undetected; these faults shall be taken into account in the consideration of the required hardware redundancy and tests periodicities.

2.2.3.11 The computer system architecture shall be designed to support the common positions concerning verification and validation (see chapters 2.5 and 2.6).

2.2.3.12 The computer system architecture shall be designed to support the common positions concerning security (see chapter 1.8), particularly those concerning communications and connectivity between systems and their associated equipment (see also 2.4.3.35).

2.2.3.13 For safety systems, 2.7.3.11 is applicable.

2.2.3.14 The evidence of satisfying the requirements of the above common positions shall be documented and made available to the regulator as early as possible in the project.

2.2.3.15 It shall be confirmed that the system architecture ensures that

a all system states are analysed to identify those that are hazardous,

b the safe state space of the system is defined,

c the system components are constrained to those whose safe state space is observable, and departure from it detectable.

2.2.3.16 Systems dependent on features not specified by the manufacturer (or outside the specified conditions of use), for example undocumented instructions or clock rates, shall not be used unless fully justified.

2.2.4 Recommended Practices

2.2.4.1 Fault Tolerance and Recovery
All fault tolerant architectural mechanisms should be justified against the following principles.

a Potential failures should be identified, and their impact on the fault tolerance capabilities should be assessed.

b Fault avoidance using highly reliable design elements should be a prerequisite of fault tolerance. If a fault can be identified and avoided, it should be designed out first before fault tolerant mechanisms are considered. Thus, fault tolerance should not be used to compensate for deficiencies in the design. It should be used only if the deficiencies are well understood, unavoidable and specific of a fault type.

* * *

2.3 Software Requirements, Architecture and Design

2.3.1 Rationale

Software development is typically an iterative process that consists of three stages: software requirements definition, software architecture and design, and implementation (ie the production of code).

The software requirements are the subset of the system and computer system architecture requirements and properties (see chapter 2.1), or requirements, properties and constraints derived from these, which are allocated to software during system design (see chapter 2.2), and will be implemented as computer programs. It is important that all requirements are allocated to or transformed into hardware or software requirements completely and correctly.

Software design must meet all the design requirements and use a simple modular structure clearly understood for implementation and testing. It must be clear enough for people who did not produce the requirements (reviewers and regulators) to trace the origin of the software requirements and verify their allocation and transformation.

The software used in a computer-based system can be from different origins (see eg [21]):

– new software: software created specifically for the application, with complete documentation;
– existing accessible software: typically software from a similar application that is to be reused and for which complete documentation is available, including the code and an unambiguous characterisation of former application environments;
– existing proprietary software: typically a commercial product or software from another application that meets all or some of the current application requirements but for which little documentation is available;
– configurable software: typically software that already exists but is configured for the specific application using simple parameter values and/or an application oriented specific input language.

These different types of software are essentially produced by two distinct processes:

– the classical development process through refinement stages of specifications and design;
– development by means of code generating tools, which receive as input a high level language application oriented description of the system requirements.

The choice between these two approaches depends on the tools and resources available, and must, in particular, take into account trade-offs between design and tool qualification (see chapter 1.5). However, in all cases, programming in a given computer language, and the use of tools always take place to some extent.

2.3.2 Issues Involved

2.3.2.1 Often, software requirements are not written as they should be. They should be written so that they can be understood and used by many different categories of people: software designers, computer system engineers, regulators and, sometimes, process engineers and safety engineers. They should be written in a form that is independent of the particular implementation, and which essentially aids understanding of how the requirements are derived from the system and safety requirements [20].

2.3.2.2 Often, software is not adequately structured. Software design is the division of the software into a set of interacting modules, and the description of those modules and of their interfaces. It is important to design so that the result is well structured and understandable to the implementers, maintainers, testers, and regulators. The design should demonstrably cover all software requirements and should not contain any unnecessary item.

2.3.2.3 The demonstration that the code is a correct implementation of the software specifications is also a major issue.

2.3.2.4 Despite all best endeavours to produce fault free software through good design practices and thorough testing, there is always the potential for unforeseen error conditions to arise. Therefore the technique of incorporating error checking (which may be based on formal assertions) into software is regarded as a sound policy. This technique is known as defensive programming. It should cover both internally and externally arising exceptions, without adding unnecessary complexity to the software.

2.3.2.5 The code usually cannot be produced correctly at once in a single version. Therefore, change requests and modifications have to be carefully controlled.

2.3.2.6 Very often, changes are not anticipated properly. This is frequently because the architecture does not localise their impact, so it is difficult to fully analyse the impact of the change. A software change is more likely to be conceived and implemented correctly if the system's software is well structured, modular, well documented and containing features which aid changes.

2.3.3 Common Position

Software functional and non-functional requirements

2.3.3.1 The software requirements shall be demonstrated to satisfy the system and architecture requirements, both functional and non-functional (see 2.1.3.2 b). The requirement documentation shall describe the design correctly and completely, and shall be understandable unambiguously by the community of its users (typically from different disciplines and backgrounds), eg implementers, maintainers, verifiers, reviewers and regulators.

2.3.3.2 The software requirements document shall define the interfaces between the software and the following: the operator, the instrumentation (sensors and actuators), the computer hardware and other systems with which communication links exist.

2.3.3.3 The software requirements shall address the results of the computer system failure and hazard analyses (see 2.2.3.8). The software requirements shall include the necessary supervision features to detect and report hardware failures. They shall also include the necessary supervision features for the software. In accordance with 2.8.3.2, the adequacy of automatic self supervision in combination with periodic testing during operation shall be demonstrated.

2.3.3.4 Defensive programming techniques shall be employed where known states and ranges of variable and parameter values can be predicted. For example, where the state of a valve can be either open or closed, a check shall be made that, after a finite change-over time, one state or the other is achieved. This should be done by checking open AND not closed, or closed AND not open. In the case of parameter ranges, if the output of a computation returns an out of range value then an exception shall be notified (see also 2.1.3.2 c, 2.1.3.2 d, 2.2.3.7, 2.2.3.8 and 2.2.3.9).

2.3.3.5 Requirements shall include an explicit and verifiable specification for the order of execution and timing inter-relationships, especially considering multiple concurrent physical processes, inter-process synchronization and shared resources. It shall be confirmed that constraints are included to ensure analysability and verifiability.

2.3.3.6 The software shall be designed so that, where practicable, the correct operation of the plant actuator selection and addressing system is verified. The design shall also include means for verifying that the final actuated equipment has achieved the desired state in the required time.

2.3.3.7 The computer system shall employ a hardware watchdog of demonstrably adequate reliability. This watchdog shall be serviced by a program operating at the base level of priority so as to detect problems such as programs locked in loops or deadlocked.

2.3.3.8 If reliability targets are set for the computer-based system, then the common positions and recommended practices of chapter 1.13 on reliability are applicable. The level of reliability required from the software shall be explicitly stated, with the understanding that the achievement of a software reliability level is less demonstrable than other requirements.

Software architecture and design

2.3.3.9 The software architecture and design shall satisfy the architecture and design requirements.

2.3.3.10 The software architecture shall be organised on a modular basis in a manner which minimises the potential for design faults and also facilitates verification by an independent team. The use of hierarchical modularisation, encapsulation and information hiding is recommended such that the design of a module is restricted to a small clearly defined set of functions that requires only minimum interaction with other functions and minimises the impact of changes. The interfaces between the various modules shall be simple, completely identified and documented.

2.3.3.11 The separation between safety functions, safety related functions and functions not important to safety, which is defined by the computer-based system requirements, shall be maintained in the software design and its achievement demonstrated.

2.3.3.12 Documented evidence shall be given that the supervision features mentioned in 2.3.3.3 maintain the system in a safe state.

2.3.3.13 The design shall constrain the implementation so that the execution of the software is verifiable and predictable.

2.3.3.14 Library modules shall benefit from an adequate level of defensive programming at the application level, in particular with respect to the "within range" validation of their input parameter values, the detection of anomalies and the generation of safe outputs.

2.3.3.15 The design of application software – which is usually new software being developed – shall be shown to be defensive. In particular, it will be shown to properly react to the potential failure modes identified for the system it makes use of, according to the requirements of 2.2.3.8 and 2.3.3.3.

2.3.3.16 The software shall be designed to support the common positions concerning verification (see chapter 2.5), for instance by minimising retention of state information between system execution cycles and by maximising independence between software input-output paths (see 2.4.3.3 and 2.4.3.8 on coding).

2.3.3.17 All system software (as opposed to the application software), including the runtime environment etc, shall ensure correct execution of application functions under the full range of possible system operating conditions.

2.3.4 Recommended Practices

Functional and non-functional software requirements

2.3.4.1 The software requirement specifications should be based on a functional model of the system to be implemented, as for example, a finite state machine model or an input/output relation model (see, for example [22]).

2.3.4.2 For better traceability and verifiability, the naming and encoding of data structures, input, output and internal variables should be meaningful in the system environment (for example, names as set point, temperature, trip, should be used). Names should be used consistently throughout the whole software.

2.3.4.3 For each relevant software output, a safe state should be indicated that can be used in case of a detectable but not correctable error.

2.3.4.4 Assertions aiding fault detection should be inserted into the code, and the system should be designed and implemented to be as fault tolerant as possible. For example, data errors occurring in inputs and on transfer between modules should be trapped and alarmed; instrument readings should indicate when they are off scale.

Software architecture and design

2.3.4.5 Module interfaces should be simple, and such that the consequences of expected changes should be confined to a single or to a small number of modules.

2.3.4.6 Modules should be defined in a way which makes it possible to reason about and test a module part (program) in isolation from other module parts and from other modules. The contextual information which is required for these activities should be part of the module documentation.

2.3.4.7 The interface between the system software and the application software should be completely defined and documented.

*　　*　　*

2.4 Software Implementation

2.4.1 Rationale

Software implementation is the phase which follows the design phase. It is the process – typically using a programming language – of transforming the software specification into a set of instructions executable by computer.

Well structured programs employing good programming practices will, in the main, be more dependable than those that do not follow these practices, in particular, when coding is performed manually. A careful selection of the language features and program constructs that are used also makes the software easier to test.

Programming directives and coding seek to ensure that good programming practices are used, that the formats of the code and of its documentation are consistent throughout the system and that they are well structured and understandable.

2.4.2 Issues Involved

2.4.2.1 The instruction sets of digital computers allow, through the use of branch instructions (including computed branches), index registers (used as address pointers) and indirect addressing, extreme complexity to be introduced into the code (both through poor structuring or the addition of unnecessary functionality). This complexity in coding can result in the introduction of faults during the coding phase and when making software changes. It also makes the detection, during the verification and validation phases, of any fault introduced more difficult.

2.4.2.2 To address these problems, the code needs to be verifiable, statically analysable and contain no unknown or undefined states. Hence, strict coding rules need to be defined and enforced. These coding rules should impose a stringent discipline on the writing of the safety system software and ensure that code legibility is given priority over ease of writing.

2.4.3 Common Position

2.4.3.1 The detailed recommendations given in appendix B of IEC 60880 for the coding of software shall be followed. For the implementation of safety system software, all exceptions to these recommendations shall be duly justified. However, the common positions contained in this section 2.4.3 shall be followed under all circumstances.

2.4.3.2 A document containing the coding directives (including design constraints in accordance with 2.3, coding rules, tool configuration restrictions and use of predefined functions) that are enforced on the programmers to support the system properties required (eg verifiability, static analysability, absence of unknown and undefined states etc) shall be produced before the start of the coding phase. This document shall be available to all those with a need for such information (eg regulator and third parties).

Coding

2.4.3.3 The programming languages that are used shall have a rigorously defined syntax and semantics or they shall be demonstrably restricted to a safe language subset.

2.4.3.4 The code shall be traceable to and verifiable against the software requirements. If verification is planned to include human inspections, the code shall be readable, adequately commented, and understandable.

2.4.3.5 The code of the different programs of a module shall be displayed in specific sections of the module documentation, together with the textual information necessary to verify the correctness of the program to the module requirements.

2.4.3.6 The code shall be written in a style and with language features and program constructs that support the common positions concerning verification (see chapter 2.5), for example by reducing the number of dynamic test cases required.

2.4.3.7 The code shall be designed so as to facilitate static analysis, testing and readability.

2.4.3.8 In particular, the code shall – as much as possible – run in a direct and fixed sequence pattern, ie:

a Computed branching shall be prohibited. Branching into loops, modules or subroutines shall be prohibited.

b Dynamic instruction changes shall be prohibited.

c Interrupts shall be avoided unless they lead to a significant simplification. Where interrupts are used, their usage and masking during time and data critical operations shall be proven correct and shall be well documented. The use of high-level synchronisation programming primitives shall preferably be used to deal with interrupts. The hardware and software shall be designed so that every interrupt is either serviced or explicitly masked.

d Recursion – not amenable to static analysis – and re-entrancy shall be avoided unless they are shown to produce simpler code than by other means.

e Nesting of loops shall be limited to a specified maximum depth for safety system software, and modification of loop control variables within the loops shall be prohibited.

f All alternatives in switch or case statements shall be explicitly covered by the code. Appropriate actions shall be implemented to deal with default conditions.

g Indirect addressing shall be used carefully so that computed addresses shall be easy to identify.

2.4.3.9 Dynamic storage allocation shall be prohibited.

2.4.3.10 Module size shall not exceed a limit specified for the system without justification.

2.4.3.11 Unnecessary code compacting, programming tricks and unnecessary optimisations which make the code less understandable, less amenable to static analysis and more difficult to test shall be avoided.

2.4.3.12 To facilitate static analysis and be independent from particular compiler conventions on typing, operations involving different types of variables shall be avoided unless conversions appear explicitly.

Subroutines

2.4.3.13 For safety system software, a maximum shall be specified for the depth of nested subroutines (eg to avoid stack overflows).

2.4.3.14 To reduce complexity, avoid potential coding mistakes, and facilitate static analysis, failure mode analysis and testing, subroutines shall

a communicate with their environment exclusively via their parameters;

b for safety system software, have only one entry point and return from only one point.

Data structures and addressing

2.4.3.15 Variables and data shall be explicitly declared and assigned. Explicit initialisation of all variables shall be made before their first use. A variable and its identifier, if not local to a subroutine, shall not have more than one meaning.

2.4.3.16 In order to facilitate code reading, static analysis, failure mode analysis and testing, distinctly different symbolic names shall be used to represent constants, variables, modules, procedures and parameters.

2.4.3.17 To facilitate static analysis, failure mode analysis and testing, only explicit mechanisms such as mutual exclusion or message passing programming primitives shall be used to access data shared by concurrent processes. Equivalence statements used to specify the sharing of storage units by two or more entities in a program unit shall be prohibited.

2.4.3.18 In agreement with 2.4.3.6 and 2.4.3.7, arrays shall have a fixed, pre-defined length. The number of dimensions in every array reference shall be equal to the number of dimensions in its corresponding declaration. To facilitate static analysis, failure mode analysis and testing, dynamic computation of indexes shall be prohibited.

2.4.3.19 Arithmetic expressions shall reflect the equations they represent.

2.4.3.20 Arithmetic expressions shall be executed in binary fixed-point arithmetic, unless the use of floating-point arithmetic can be demonstrated to contribute to safety.

2.4.3.21 The mapping of real continuous variables into fixed point variables shall be explicit and documented in the code, and its accuracy justified. If different mappings are used, their mutual compatibility shall be justified.

2.4.3.22 If execution in floating-point arithmetic is necessary, the hardware and software used to implement floating point calculations and functions shall be suitably qualified.

Defensive programming

2.4.3.23 If complicated calculations are necessary, the program shall be written so that – where feasible – a simple function or assertion will provide an exception check and back-up action.

2.4.3.24 In addition to requirements 2.2.3.8 and 2.3.3.4, defensive coding shall be applied by programmers with the objective of maintaining program behaviour within its specifications, including the following:

a known states and program properties, and parameter ranges shall be verified;

b variable values that can cause faulty operations shall be detected, (for example, division by 0).

2.4.3.25 Preventing overflow and underflow of assigned memory (eg in the case of arrays, lists, queues, buffers) shall be statically addressed through strong data types and type checking. If static prevention cannot be guaranteed, dynamic checking shall be performed.

Language and compiler requirements

2.4.3.26 The programming language used to write the code shall be restricted to a safe subset. In particular, a high level, strongly typed, programming language that prohibits the unsafe mixing of variable types is preferred.

2.4.3.27 The use of assembly language or machine language shall be limited to the code needed to interface with hardware. If used for other purposes, properties such as verifiability and traceability shall be confirmed, and the usage justified.

2.4.3.28 The compiler of the programming language shall satisfy the applicable common positions of 1.5.3. In particular, the defect detection capabilities of the compiler shall be clearly defined and verified to the extent that the safety demonstration depends on these capabilities.

2.4.3.29 The same compiler version shall be used for all compiled parts of the software running on the same processor.

2.4.3.30 If a new version of the compiler has to be used, the software affected by the change shall be re-tested unless it can be demonstrated that the machine code has not changed.

2.4.3.31 Any compiler code optimisation shall be provably correct. The same optimisation options shall be used on all the software compiled by the same compiler.

Operating systems

2.4.3.32 In support of 2.3.3.13, the behaviour of the operating system shall be predictable, analysable and verifiable (see also 2.4.3.34), for example by ensuring that:

a the organisation of the operating system into tasks, including the allocation of services to tasks, is as simple as possible, clearly identified and documented;

b the task generation should take place statically in a linear sequence in the initialising phase (no dynamic task creation and deletion);

c the task priorities are statically assigned;

d no dynamic memory allocation is used;

e the system includes all the self diagnosis features necessary to detect its own failures;

f use of interrupts is restricted to cases where this simplifies the design and there are no unanalysed conditions (see also 2.4.4);

g the absence of overflow is demonstrated;

h the absence of overflow and underflow for stacks, lists, queues and buffers, and the absence of deadlocks and starvation are demonstrated.

2.4.3.33 In support of 2.3.3.13, it shall be demonstrated that all platform services are available when needed with the necessary capacity.

2.4.3.34 If the requirements of 2.4.3.32 are not satisfied by a complete operating system, a properly selected and configured subset of the operating system shall be used in conjunction with the specific application. The specification shall cover the qualified subset and configuration, the environment including the application for which it is qualified and any other constraints under which the operating system was qualified.

Communications

2.4.3.35 Any communications bus used shall comply with an appropriate bus standard, justified to support the safety functions, considering the main characteristics of the bus standard and the network design, including:

- a the security features of the software run time environment used to implement the bus;
- b the principle applied to ensure safe communication (eg black channel combined with a safety layer, or white channel with necessary hardware features, redundancy checking, special hardware drivers and transmission monitoring);
- c the network topology necessary to support safety (eg point to point);
- d features for detecting, annunciating, preventing and/or mitigating different types of communication failures and malfunctions (eg message loss, corruption, delay, unintended repetition or wrong sequence, wrong addressing, broadcast storm, unintended masking, inconsistencies in data from different sources, jitter, collision);
- e predictability;
- f ability to satisfy real-time constraints;
- g support for encryption protocols;
- h the means for logical separation of systems (eg using an application specific gateway within the protocol, to determine whether or not a data transmission is allowed).

Support software and tools

2.4.3.36 All support software and tools used in the production and testing of code (compilers, assemblers, linkers, loaders, operating systems, configuration control systems, static analysers, test harnesses, coverage analysers, etc.) shall be qualified or otherwise suitably justified, and shown to comply with the requirements of section 1.5.3.

Documentation

2.4.3.37 The source code shall include commentaries to a level of detail that is consistent throughout the whole software. Commentaries shall be consistent with the documentation of the software requirements and of the design. They shall be a sufficient complement to this documentation to make the code readable and understandable.

2.4.3.38 In particular, commentaries, related to the code of every software module or program, shall contain all the complementary information necessary for an independent and stand alone review and verification of this code against its specification.

2.4.3.39 Commentaries shall also include any specific information required for maintenance of the code, in particular, its history and the designers' names.

2.4.3.40 If a tool has generated the code, 2.4.3.37 to 2.4.3.39 apply to the tool inputs.

2.4.4 Recommended Practices

Coding

2.4.4.1 If a requirement specification language is used, it should have a well-defined syntax and semantics.

2.4.4.2 The coding phase should start only after the verification team has approved the design phase.

2.4.4.3 Evidence should be provided that structured programming techniques have been followed.

2.4.4.4 Branching out of loops should be avoided, except in the case of exceptions.

Data structures and addressing

2.4.4.5 The use of global variables within subroutines should be avoided.

2.4.4.6 The symbolic names and types of variables, constants, modules, procedures and parameters should be meaningful at the application level.

Language and compiler requirements

2.4.4.7 For application software, it is strongly recommended that a problem-oriented language rather than machine-oriented language be used.

2.4.4.8 Assembler and machine code insertions into high level language code should not be allowed without justification.

2.4.4.9 Automatic generation of source code by validated tools is preferable to manual writing. If a generation tool is used, it shall comply with the recommendations of chapter 1.5.

2.4.4.10 The compiler should have been previously and widely used in comparable applications, and should comply with the recommendations of chapter 1.5. For safety system software, the use of compiled code optimisation options should be avoided.

Operating systems

2.4.4.11 When the evidence required in 2.4.3.32 for using interrupts cannot be provided, other techniques can be used to support satisfaction of timing constraints (see 2.4.3.33) and correct order of execution. Examples of evidence to be provided for monitoring hardware such as sensors and actuators, and for confirming interrupt-avoiding provisions are:

a correct identification of rates of change in the phenomena to be monitored,

b for monitoring a continuously varying phenomenon, correct identification of the sampling interval that characterises the monitored variable with accuracy,

c for reacting to sporadic events, eg sudden change, correct identification of the minimum time between arrival of successive events, based on the physics of the event-generating process,

d evidence of definition of commensurate periodic tasks,

e evidence of adequate periodicities of the tasks to achieve cyclic, deterministic timing behaviour,

f provision of an explicit, verifiable specification for the order of execution and timing inter-relationships, especially considering multiple concurrent physical processes, inter-process synchronisation and shared resources,

g provision of an explicit, verifiable specification for the detection of the occurrence of faults and the existence of faulted states, including the required monitoring interval (or sampling period) and timing relationships,

h avoidance of hardware that does not support the above monitoring and fault detection,

i provision of an explicit, verifiable specification for the mechanisms to recover from faults.

* * *

2.5 Verification

2.5.1 Rationale

In order to obtain sufficient evidence that the software implementation complies with the functional and non-functional requirements, appropriate verification procedures have to be established and implemented at well-defined milestones in the course of the software lifecycle process.

Most of the commonly used software development lifecycle models include a verification step after each software development phase to ensure that all requirements of the previous development phase have been implemented completely, consistently and correctly.

There is a wide spectrum of verification methods from which ideally an optimal and well balanced subset has to be selected.

Some of the main methods currently used for software verification are

- Reviews, walks-through, audits: Manual or tool supported measures, particularly used to verify the phases of requirement specification and design specification;
- Static analysis: Source code verification, mainly with respect to design and coding standards, symbolic execution and data-flow;
- Testing: Verification of the executed object code with respect to software correctness (note: In the framework of validation, tests will be performed on target hardware and under target environmental conditions). The IEC 60880 standard [5] lists the most common software testing methods. (See also for example [23]).

2.5.2 Issues Involved

2.5.2.1 Demonstration of software correctness and of its contribution to reliability:

- As already stated in chapter 2.2, executable software components can be of different types: operating system, library, application software, intelligent drivers, sensing and actuation devices, communication protocols, human-machine interfaces, etc. These components do not contribute in the same way to the safety of the whole system. Specific evidence and methods of validation and verification are necessary to show that these components comply with the safety requirements of the system.

- Software behaviour can be highly sensitive to "small" programming, coding or even typing mistakes. Further, the functionality of software-based systems is usually more complex than that of their analogue counterparts. Therefore it is difficult to estimate how and to what extent the software may affect the overall system reliability.
- Although validation is an important final step to obtain evidence of the software correctness, experience shows that it must be complemented by the results of well-defined verification prior to the final validation.
- A formal proof of correctness is sometimes possible for certain requirements that can be modelled and analysed by formal and mathematical techniques. In practice, this will not be possible for the overall software system which may include an operating system, communication protocols, etc.
- The digital nature of a computer-based system prevents advantage being taken of extrapolation and interpolation during the determination of the types and number of tests needed to demonstrate compliance with the requirements. This limitation obliges the verifier to carefully justify the amount of testing performed in relation to the required system reliability.

2.5.2.2 There is a wide spectrum of verification methods and tools. A selected set of these must be planned and shown to be appropriate to the lifecycle phase to which they are to be applied, and to the safety claims that are to be made.

2.5.2.3 Recent experience shows that the verification of safety critical software may be costly in time and resources. A verification plan therefore has to be carefully established early in a project in order to gain the maximum safety benefit from the expenditure of this time and resources.

2.5.2.4 A detailed understanding of the relationship between verification results and software requirements is necessary in order to obtain evidence that the software fulfils its requirements and demonstrate the absence of undesired functionality and behaviour.

2.5.2.5 Confidence in verification results depends also on organisational measures; therefore those in charge of the verification are required to have the necessary qualifications, independence and knowledge of the system.

2.5.2.6 Some further issues such as the verification of pre-existing software and use of formal methods and tools are separately discussed in chapters 1.4, 1.5 and 1.9.

2.5.3 Common Position

Selection of verification methods and tools

2.5.3.1 At the end of each phase of the lifecycle (requirements, design, coding, hardware-software integration), the output of the phase shall be verified against its input specifications.

2.5.3.2 The scope and depth of analysis and verification that is planned for the end of each phase shall be determined and justified in terms of the evidence it is intended to contribute to the demonstration of safety. This justification shall be documented, and made accessible to the regulator.

2.5.3.3 For each major component of target executable software (operating system, library, application software, intelligent drivers, sensing and actuation devices, communication protocols, etc.), as far as reasonably practicable, the absence of potentially unsafe remaining defects that can cause the loss of a safety function of the system shall be demonstrated.

2.5.3.4 In addition, different verification methods shall be combined in order to achieve a sufficient coverage of the functional and the non-functional requirements. The coverage achieved by this combination shall be determined and justified (see for example 1.9.3). This justification will be accessible to the regulator.

2.5.3.5 Every software module shall be verified by executing the code to a degree of coverage that complies with 2.5.3.10 to 2.5.3.21.

2.5.3.6 The verification of the system integration shall include testing by executing the code.

Verification planning

2.5.3.7 A verification plan which specifies the verification steps, their schedule, the procedures, the persons in charge, the contents of the verification reports, and the follow-up for detected faults and anomalies will be established and made accessible to the regulator early in the project.

2.5.3.8 The verification plan shall cover in detail all distinct software development phases as well as the different levels of system integration and software architecture (module, program, and subroutine).

2.5.3.9 The verification plan shall address the criteria, strategy and equipment that are necessary for each verification step. Furthermore the verification plan shall clearly justify and define the intended test coverage.

Coverage

2.5.3.10 The range of values and all the discontinuity and boundary neighbourhoods of every input variable shall be tested. Attention shall be paid to the use of equivalence partitioning.

2.5.3.11 All modes of system operations shall be considered during the design of test cases.

2.5.3.12 Each distinct test at system or module level shall monitor all outputs of the system or module.

2.5.3.13 All interfaces (between modules, and between programs), and all communication protocols shall be tested.

2.5.3.14 Fault tolerant and exception mechanisms (eg divided by zero, out of range values, overflows) shall be tested. If this is unfeasible, evidence of their correct behaviour shall be provided by other means.

2.5.3.15 Test specifications shall ensure that all lines of code, and – at system integration – all module calls are exercised, and coverage analysis shall justify the coverage of these tests.

2.5.3.16 Data structures and declared constant values shall be verified for correctness.

2.5.3.17 Operations on – and access to – all data items shall be exercised, and the coverage of these tests shall be justified.

2.5.3.18 All time critical sections of code shall be tested under realistic conditions, and the coverage of these tests shall be justified.

2.5.3.19 The result of calculations performed by software shall be verified against pre-calculated values.

2.5.3.20 The adequacy of commentaries and code documentation shall be verified.

2.5.3.21 Compliance with the appropriate project design and coding directives and standards shall be verified.

Traceability and documentation

2.5.3.22 Test plans, procedures and results shall be documented. The expected test results shall also be documented, together with the method for their derivation. This documentation should be accessible prior to the execution of the tests.

2.5.3.23 Test results shall clearly indicate which tests met expectations and which did not.

2.5.3.24 All anomalies revealed by the verifications shall be recorded and investigated. The results and follow-up of these investigations shall be reported and made accessible to the regulator. If software changes are necessary, change requests shall be issued and processed according to the procedures of chapter 2.7.

2.5.3.25 Using the documentation, it shall be possible to trace each verification result back to the associated functional and non-functional requirement(s). For this purpose appropriate comments shall be contained in the source code documentation.

Independent verification

2.5.3.26 For safety system software, the planned verification shall be designed and performed by personnel that are independent from the team that designs and produces the software. This requirement, however, does not apply to the verification of the hardware-software integration phase.

2.5.3.27 All communications and interactions between the verification team and the design team, which may have a significant bearing on the verification results, shall be recorded in writing.

2.5.3.28 All verification activities shall be comprehensively documented to enable auditing.

2.5.3.29 Personnel conducting the tests shall have received proper training in the use of the applicable test tools and methods.

2.5.4 Recommended Practices

2.5.4.1 In order to limit the high effort and costs of verification that may be involved, an optimised ratio between static analysis techniques and testing should be found.

2.5.4.2 By using development tools, some of the neighbouring development steps of the classical software lifecycle model can be combined – for instance the steps from design specification to software implementation. In such cases, a special verification effort should be dedicated to the qualification of the applied tools (see chapter 1.5) and of the target object code.

2.5.4.3 In addition to the requirements of IEC 60880, the application of verification concepts based on formal methods – eg symbolic code execution and formal code verification – and automated transformation tools are recommended.

2.5.4.4 As a precaution against the introduction of erroneous or unwanted code – introduced for example by translation tools – the machine code should be translated into a form suitable for comparison with the specifications (reverse engineering). This process can be carried out either manually or using software aids.

2.5.4.5 Anomalies only should be indicated and reported by the verification team; they should not recommend design changes.

* * *

2.6 Validation and Commissioning

2.6.1 Rationale

Validation and commissioning are processes that demonstrate through testing that the system will perform its intended functions. These processes of the safety demonstration are regarded as essential since the current analysis techniques do not enable the correctness of a system to be fully demonstrated.

More precisely, validation in this consensus document is regarded as the test and evaluation of the integrated computer-based system hardware and software to demonstrate compliance with its functional and non-functional requirements specifications. Commissioning is regarded as the onsite process during which plant components and systems, having been constructed, are made operational and confirmed to be in accordance with the design assumptions and to have met the safety requirements and the performance criteria. For the purposes of this Consensus Document it will be assumed that commissioning follows the principal stages given in the IAEA Safety Guide No NS-G-2.9 [24], "Commissioning for Nuclear Plants", namely: pre-operational testing; cold performance testing; hot performance testing; loading of fuel and sub-critical testing; start-up to initial criticality phase; low power tests.

2.6.2 Issues Involved

2.6.2.1 The digital nature of a computer-based system precludes the use of extrapolation and interpolation in the determination of the type and number of tests needed to demonstrate compliance with the requirements. This limitation obliges the validater to carefully justify the amount of testing performed in relation to the required system reliability. As for verification tests, use may have to be made of equivalence partitioning and boundary conditions. There is also the problem of demonstrating that inaccessible features (those not readily testable at the validation stage) have been tested satisfactorily at the commissioning stage. Finally, the level of input simulation for an acceptable demonstration has to be shown to be adequate.

2.6.2.2 As already stated in chapter 2.2, executable software components can be of different *software types*. These components do not contribute in the same way to the safety of the whole system. Specific evidence and methods of validation and verification are necessary to show that these components comply with the safety requirements of the system.

2.6.2.3 During software production, there are occasions when the designer recognises the need for an "added feature". It is important that these added features are reflected in the appropriate requirement specifications and that they are included in the validation tests.

2.6.2.4 Since it is neither safe nor reasonably practicable to test the behaviour of a safety system using real accident scenarios on the plant, this aspect of the system has to be tested using simulated scenarios which represent the dynamic behaviour of the plant. The derivation of representative scenarios can be problematic.

2.6.2.5 There is a possibility that the assumptions made about the behaviour of a device connected to the system may be incorrect – and, therefore, that the simulation model might be inadequate. This possibility leads to the need for a justification of the correctness of the simulation of all devices – in terms of input and output – during validation and commissioning.

2.6.2.6 It is important that the design team does not influence the validation activity since this could be a source of common cause failure due to inappropriate test specifications or incorrect interpretation of results. Therefore, there should be some agreed level of independence between these two teams. This independence, however, should not deprive the validation team from acquiring sufficient knowledge of – and familiarity with – the system.

2.6.2.7 There is also the problem of determining the degree of test coverage required and the adequate level of simulation at each of the commissioning phases mentioned in the rationale section 2.6.1.

2.6.3 Common Position

Validation

2.6.3.1 The computer system validation shall be conducted in accordance with a formal validation plan.

2.6.3.2 The plan shall identify the intended coverage of the validation and justify it with respect to the requirement of 2.1.3.1 (including how the system validation tests are derived from the system requirements).

2.6.3.3 The plan shall identify acceptance criteria, techniques, tools, test cases and expected results.

2.6.3.4 Validation tests shall be performed on target hardware and under representative environmental conditions.

2.6.3.5 The test programme shall encompass all modes of operation (covering particularly maximum loadings) and ranges of variables under conditions which are as representative of the operating environment as possible, including relevant normal operations, anticipated operational occurrences and accident conditions.

2.6.3.6 Testing shall be conducted in a disciplined and orderly manner and in accordance with IEC 60880 [5]. During every test, the status of all outputs shall be monitored to ensure that unintentional changes do not occur. The proposed tests shall exercise the software across the full range of the requirement specifications paying particular attention to boundary conditions and making justified use of equivalence partitioning.

2.6.3.7 Timing conflicts and bus contention problems shall be thoroughly tested during validation. This is important because they cannot be tested comprehensively in earlier phases. Adequacy of spare processor time, scan times and data transfer rates shall be demonstrated.

2.6.3.8 It shall be ensured that the requirements specifying the fault, exception and failure mitigating mechanisms (see 2.2.3.6 and 2.2.3.7), and the requirements which specify the system behaviour in the presence of abnormal inputs and conditions (see 2.1.3.2 d), are correctly implemented.

2.6.3.9 The system shall also be subjected to appropriate validation tests to check that the set of system requirements is complete, consistent and correct.

2.6.3.10 All time critical sections of code shall be tested under realistic conditions, and the coverage shall be justified. Where calculations are performed in the software their result shall be checked against pre-calculated values.

2.6.3.11 Appropriate tests shall be done to verify the fault tolerance of the system. These shall include a series of tests that demonstrate that the system is tolerant of input sets that are outside its operational input space. The chosen set of tests shall be justified in terms of its appropriateness to the operational profile.

2.6.3.12 All tests shall be fully documented and analysed, and each test shall be traced to the associated system requirement specification. This documented analysis shall be used to demonstrate the test coverage achieved.

2.6.3.13 The expected outputs shall be stated in the test procedures before the tests are undertaken. In each test, all outputs shall be monitored, and any anomaly investigated and its resolution fully documented.

2.6.3.14 All changes to either software or test specifications shall be subjected to change control procedures that comply with the recommendations of chapter 2.7.

2.6.3.15 Features added during the design process shall be fully tested. These shall be traced to the changes in the system or design requirement specifications to demonstrate the consistency, completeness and correctness.

2.6.3.16 The validity of simulated inputs shall be justified, and tests shall demonstrate the correct operation of simulated items.

2.6.3.17 The system shall be tested using a simulation of the accident scenarios. These tests shall be based on an analysis of the plant transients induced by postulated initiating events. The number of tests executed shall be sufficient to provide confidence in the system safety.

2.6.3.18 The validaters shall be independent from the team producing the requirement specifications and the software. All communications and interactions with significance to validation results shall be recorded in writing. The validation team shall review the specifications for correctness and completeness, and shall devise and implement the test strategy that demonstrates that the system meets its requirements.

2.6.3.19 All validation activities shall be comprehensively documented to enable auditing. The validation team shall indicate anomalies only; they shall not recommend design changes.

2.6.3.20 The system operation and maintenance manuals shall be validated, as far as possible, during the validation phase.

Commissioning

2.6.3.21 Commissioning tests shall ensure that both the system and the plant safety requirements are complete and correct.

2.6.3.22 The number and type of commissioning tests shall be justified. The omission of any validation test from the commissioning tests shall be justified.

2.6.3.23 The validation of the system operational and maintenance manuals shall, if necessary, be completed during the commissioning phase.

2.6.3.24 On-site testing of the full system (software and hardware) over an extended period (the duration will depend on system complexity) shall take place. During this period the system shall be subjected to routine test and maintenance activities. A log shall be kept of any revealed faults, and appropriate actions shall be taken in agreement with chapter 2.7.

2.6.4 Recommended Practices

2.6.4.1 For safety systems, the system reliability should be estimated using statistical testing. The applied tests should be randomly selected from the operational input space and their number should be based upon the safety system required reliability and confidence level.

2.6.4.2 The validation plan shall be defined as soon as possible during the project.

2.6.4.3 The traceability of the tests to the requirement specifications should be based on a requirement matrix, that contains for every software requirement a reference to the tests which cover this requirement, and to the lifecycle phase (verification step, validation, commissioning) where they have been performed.

* * *

2.7 Change Control and Configuration Management

2.7.1 Rationale

Changes to the software of safety systems in a nuclear plant have the potential to affect significantly the safety of that plant if incorrectly conceived or implemented. Any alteration to the software of these systems at any phase in the system lifecycle, whether due to an enhancement or a need to adapt to a changing environment (resulting in a modification to the system requirements) or to correct a defect must be conceived and implemented at least to the same standards as the original implementation. This point is particularly important in the case of changes made after delivery. Furthermore, there should be procedures in place to ensure that the effects of such changes, on all parts of the system, are assessed to reduce to acceptable levels the potential for faults to be introduced.

A software-based safety system is formed from many different items of software and hardware and includes many documents that describe these items. Hence it is important that there is a full index of the items involved in the construction of a system, and that the status of each item (including changes made) is established and tracked so that faults are not introduced due to the incorrect versions of these items being used. A well-ordered configuration management system provides the means to ensure this, as well as playing a significant role in a properly managed change control process.

2.7.2 Issues Involved

2.7.2.1 Software changes, as with any other changes, can occur either during the development of a system up to and including the end of commissioning (usually classed as "software modifications"), or in operation following the completion of commissioning at site (usually classed as "software maintenance").

2.7.2.2 The international standards IEC 60880 [5] and IEC 61513 [4] contain the necessary elements for an acceptable software change process. However, for regulatory purposes and for safety systems there is a need to add an additional dimension to the change process. This additional dimension results from the need for: (i) an analysis of the effect of the change on safety; (ii) the provision of documentary evidence that the change has been conceived and implemented correctly; and (iii) a process of independent review and approval of the change. The reviewing process described in 2.7.3 is a clarification for licensing purposes of the requirements of the standard.

2.7.2.3 The need for a software change has to be carefully evaluated because the consequences of a change are not always easily anticipated. For example, other faults may be introduced.

2.7.2.4 Consideration should be given during the design phase to software maintainability since this attribute will reduce the number of faults introduced by any software change process.

2.7.2.5 If the configuration management is inadequate, then earlier versions of software and documentation may be inadvertently incorporated into the current version of the system or documentation. These earlier version items may include faults or properties that are unsafe when used in the current version.

2.7.2.6 For changes enacted post delivery to site (particularly during commissioning and operation), there is a need to ensure continuity of a rigorous regime of software change control.

2.7.3 Common Position

General

2.7.3.1 The software change procedure and documentary process shall apply to all software elements of the system including its documentation. This procedure shall apply equally to system functionality enhancements, environmental adaptations (resulting in a modification to the system requirements), changes in the elements (including hardware) on which the software depends and corrections of implementation defects.

2.7.3.2 An appropriate software architecture (see chapter 2.3 for guidance on software design features which aid maintainability) and a suitable software configuration management system (see 2.7.3.18 to 2.7.3.22) shall be used during the lifecycle process in order to maintain safety. For safety systems, no distinction shall be drawn between major and minor software changes since a wrongly implemented change could affect safety.

2.7.3.3 Once a software or documentation item has been approved, ie has been placed under configuration control (usually following its initial verification and placing in the software/documentation library for release), any changes to this item shall be controlled by a procedure containing the elements given in 2.7.3.4, 2.7.3.5 and 2.7.3.6.

2.7.3.4 The software change procedure and the configuration management system shall include an adequate problem reporting and tracking system.

2.7.3.5 The software change procedure shall contain the following basic elements:

a the identification and documentation of the need for the change;

b the analysis and evaluation of the change request, including: a description of the design solution and its technical feasibility; and its effect on the safety of the plant and on the software itself;

c the impact analysis of each software change: for each software change, the implementers of the change shall produce a software impact analysis containing a short description of the change plus the relationship with the hardware and software items, a list of software parts affected by the change and the extent to which the functional and non-functional requirements remain satisfied. Particular attention shall be applied to the interfaces between modified and unmodified software. Objective evidence that the full impact of each software change has been considered by the implementer for each software part affected shall be provided. As a minimum this shall consist of a summary description of the change to be implemented, plus documentary evidence of the effect of the change on other software parts. The software impact analysis shall also list data items (with their locations – scope of the data item) that are affected by the change plus any new items introduced and the relationship of the software items with the affected or introduced data items.

d the implementation (consistent with the standards employed in the original production process), verification, validation (as appropriate) and release of the changed software or document item. The V&V phases may make use of the software impact analysis to perform regression testing.

e The requirements of IEC 60880 [5] sections 9.1, 9.2 and 9.3 shall apply.

2.7.3.6 Faults shall be analysed for cause and lack of earlier detection. Any generic concern shall be rectified, and a report produced.

2.7.3.7 Change control and configuration management tools shall be qualified according to 1.5.3.

Software modification

2.7.3.8 A correction to a wrongly implemented software change, if found at the stage of site testing (ie following installation of the software on the plant), shall be processed as though it were a new software change proposal.

2.7.3.9 The commissioning team shall use the software impact analysis and the factory V&V report to develop its own test specification and site commissioning test schedule.

Software maintenance

2.7.3.10 All software changes in operation following the completion of commissioning at site (called software maintenance in this document), shall be controlled by procedures which meet the requirements of 2.7.3.11 to 2.7.3.17).

2.7.3.11 For safety systems, program and fixed data (including operational data) shall be held in read only memory (ROM) so that they cannot be changed on-line either intentionally or due to a software malfunction.

2.7.3.12 For safety systems, software maintenance changes shall be tested on the computer system installed at site. Any divergence from this shall be justified in the test documentation.

2.7.3.13 Proposed software changes shall be analysed and reviewed for effect on safety, by suitably qualified and experienced staff – eg manufacturers (suppliers), system designers, safety analysts, plant and operational staff – that are independent from those persons proposing, designing and implementing the change. The analysis and review shall consider manufacturers (suppliers) V&V and independent assessment reports, as well as test specifications and test reports, or other such documentation as appropriate to the change being proposed. The results of the analysis and review shall be documented and shall include a recommendation for approval, or rejection of the change from the safety perspective.

2.7.3.14 Only software approved for release shall be installed at site.

2.7.3.15 Before the start of site testing of any changes to the software of a safety system, the test specification shall be reviewed by independent reviewers.

2.7.3.16 Before permitting the operational use of software following a change, an updated and checked version of the system and safety demonstration documentation (including any routine test schedules) shall be available, fully reflecting the changes that have been made. Compliance with this requirement shall be confirmed by independent review.

2.7.3.17 The independent reviews shall be documented.

Configuration management

2.7.3.18 All the items of software including documents, data (and their structures) and support software shall be covered by a suitable, readily understood and fully documented configuration management system (CMS) throughout the lifecycle. An item of software or documentation shall be accessible only to those who have a legitimate need (eg to persons responsible for its design and verification) until it is approved and under configuration control.

2.7.3.19 A formal procedure shall be set up for version control and the issuing of correct versions. Provision shall be made for informing all relevant personnel, including the regulator (in accordance with regulatory expectations), of pending changes and approved modifications.

2.7.3.20 All software copies, including any pre-existing software that is being used, shall be clearly and uniquely labelled with at least title, version number and creation or acquisition date. The identification and version number shall be included in the code so that this can be checked. For safety systems, changes made, approval status, and the names of authors, reviewers and approvers, shall be included in the source code listing or explicitly documented.

2.7.3.21 After delivery of the software and its documentation, the same level of configuration management shall be maintained at the site where the delivered software is stored.

2.7.3.22 A configuration audit shall be performed on the safety system software prior to loading to establish that the correct items and versions have been included in the system. Following system loading, the loaded version shall be verified as being uncorrupted.

2.7.4 Recommended Practices

2.7.4.1 Any software item intended to be used as part of a safety system and that has not been subjected to the above software change process should be assessed on the basis of chapter 1.4.

2.7.4.2 Where a pre-existing software item (qualified to chapter 1.4) is introduced into a safety system as part of the change process, its impact on the system should be evaluated as appropriate to the same level as described by 2.7.3.

2.7.4.3 It is always good practice to make only one software change at a time within the same system unless changes are shown to be independent. Exceptions to this practice are permissible but once significant quantities of nuclear material have been introduced into a plant (eg fuel loaded into a nuclear reactor) then there should be no more than one software change being implemented on a safety system at any one time.

2.7.4.4 To aid backward traceability, the configuration management system should maintain lists of items, and their changes, that were included in previous baselines. These lists should include names of individuals and organisations to whom configuration items have been distributed.

* * *

2.8 Operational Requirements

2.8.1 Rationale

If the processes that support and maintain the safety systems of nuclear installations during their operational use are not of a sufficiently high integrity, then they may jeopardise the safety system's fitness for purpose. For example, the accuracy of calibration data, or the process of loading data that are required for core limits' calculation, or the conversion of electrical signals into engineering units can have a significant effect on the behaviour of the safety system; and could, in fact, lead to a failure to trip. Also, the incorrect operation of automatic test equipment could result in a fault remaining un-revealed in the tested system with the potential for spreading out, even over redundant trains during later operation.

Proper operators' training, as well as the development of a periodic test plan, is necessary in order to ensure correct and safe plant operation.

2.8.2 Issues Involved

2.8.2.1 Periodic testing

As a part of a safety system, the computer-based system needs to undergo periodic testing in order to verify maintenance of its basic functional capabilities.

2.8.2.2 Generation of calibration data

A computer-based safety system on a nuclear installation is dependent for its correct operation on a large amount of calibration data that varies during the operational life of the installation due to fuel burn-up, equipment failure and planned outages. These data are derived from a variety of sources often involving both manual and automatic activities. As mentioned above, mistakes in these data could adversely affect the integrity of the safety system. For instance, a mistake in the core flux mapping readings (either in terms of instrument readings or positional measurements) could prevent the reactor's core limits calculator from tripping the reactor on a reactivity event. Also, wrong calibration data could mean that an instrument is generating an improper reading resulting again in a failure to actuate safety equipment.

2.8.2.3 Loading of calibration data

Even when the data is correctly generated there is the potential for faults to be introduced into the safety system due to incorrect entry of that data. Data entry may be either manual or electronic (either via a data link or from magnetic media). Manual data entry has all the usual problems of human error (transposition mistakes, misreading and incorrect addressing). Equally, when data is entered electronically, there is the possibility of data corruption eg due to programming faults or electronic noise.

2.8.2.4 Self-supervision functions and automatic test equipment

Digital computer technology has the capability to perform extensive self supervision functions as well as routine testing of safety systems by means of test equipment. When all trains are subjected to identical supervision or tests via identical software, there is the potential for common cause failure of the safety system due to a design defect in the supervision functions or the tester.

2.8.2.5 Plant component test and corrective maintenance equipment

Certain activities of maintenance on redundant items of equipment are allowed while the reactor is at power. This may involve the replacement of an output card of a guardline, the renewal of cabling between the guard-line and the plant item or the repair of devices controlled by the computer.

When equipment is replaced while the reactor is at power, there is the potential for causing the reactor protective system to spuriously actuate or to place the reactor into an unanticipated state. This type of event should be addressed.

2.8.3 Common Position

Periodic testing

2.8.3.1 A programme for safety systems periodic tests, that includes applicable functional tests, instruments checks, verification of proper calibration and response time tests, and maintenance of all associated records, shall be defined and implemented. The tests shall verify periodically the basic functional capabilities of the system, including basic safety and safety related functions, major functions not important to safety, and special testing used to detect failures unable to be revealed by self-supervision or by alarm or anomaly indications. Periodic testing shall not adversely affect the intended system functions.

2.8.3.2 It shall be demonstrated and documented that the combination of the provisions of automatic self-supervision and periodic testing during operations (see 2.2.3.7 and 2.2.3.10) cover all postulated faults.

2.8.3.3 The operator interface shall include the means to allow the operator to easily obtain confirmation that the computer system is alive, and that the information produced on displays and screens is properly refreshed and updated (see also 2.1.3.5).

2.8.3.4 On termination of the testing operations, the safety system shall be restored to its original status.

Generation of calibration data

2.8.3.5 The calibration data shall be generated according to the safety and reliability requirements of the I&C function to which they are assigned. An appropriate quality assurance procedure shall be established.

Loading of calibration data

2.8.3.6 Where data is loaded manually there shall be a read-back facility that requires the operator to confirm the data entered before he/she is allowed to proceed to the next item.

2.8.3.7 On completion of the data entry there shall be a printout of all data entered. Procedures shall be in place to require that printout to be checked by an independent party before the guardline is put back into service.

2.8.3.8 Where the data is entered electronically, that set of data shall be covered by a checksum which will enable the data to be verified. In addition, the data shall be read back and verified automatically against the original. Printouts of the data entered and that held in the guardline shall be provided and retained for auditing purposes.

2.8.3.9 The software for loading the calibration data and monitoring data values shall be of the same integrity as the safety system. If this is not the case, then it shall be demonstrated by means of a software hazard analysis that the safety system software and data cannot be corrupted by the loading/display software.

2.8.3.10 A full test of the guardline shall be performed following a data change so as to demonstrate its correct operation.

Equipment for periodic testing

2.8.3.11 The test equipment software shall be qualified to a level in accordance with its intended use.

2.8.3.12 So as to avoid common mode failures, constant and fixed parameter values to be used by the testing equipment shall not be obtained from the computer of the safety system, but eg from the system or the computer design requirement specifications.

2.8.3.13 The test equipment shall be subjected to periodic calibration checks.

Equipment for plant component testing and corrective maintenance

2.8.3.14 Integrated means of testing plant items shall be engineered as part of the safety system. Any additional tool shall be qualified according to 1.5.3. Engineered test systems and their user interfaces shall be designed to standards commensurate with their duty. They shall provide a read-back facility, with confirmation by the operator, of the plant item selected before it is activated for test. Such test equipment shall restrict the activation to one plant item at a time. A plant item shall be returned to its original entry status before another item is selected.

2.8.3.15 It shall not be possible to connect this type of test equipment to more than one guardline at any one time.

2.8.3.16 Bypass of actuation signals for maintenance of output devices and components shall be indicated by alarm and be recorded as unavailable.

Operator training

2.8.3.17 Instrumentation and control specialists shall follow a training programme that addresses safety system performances, operation and maintenance during normal and abnormal reactor operation.

2.8.3.18 Operator training shall be conducted on a training system which is equivalent to the actual hardware/software system. Training facilities for each operator interface device shall be provided by the training system. A computer simulator may be used for this purpose.

Operational experience

2.8.3.19 Operational experience shall be recorded and cover all operating situations. The records shall be appropriately structured and analysed so as to allow data on the system functional and non-functional behaviour to be derived over given periods of time. The records shall be made accessible to the regulator. The documentation of the operating experience shall comply with the requirements of 1.14.3.1 and 1.14.3.2. It shall be assured that the safety demonstration remains valid in the light of the operating experience. The requirements of 2.7.3.1 to 2.7.3.17 apply.

Requirements to support inspection

2.8.3.20 The licensee shall facilitate effective inspection by the regulator and make available, and provide on request, all the information necessary for the regulator to carry out its inspection programme. This programme may depend on the applied technology and comprise activities at different points in the development, operation and maintenance of the systems important to safety. Activities for software-based systems may include random observations during operation, specific inspections in the case of an event or modification, and planned inspections to accompany periodic testing.

2.8.3.21 In particular, the licensee shall make available and provide on request all the information necessary to comply with the common positions of this document. The table below highlights some of the common positions that are particularly relevant to inspection.

Table 4: Common positions relevant to inspection

Inspection topic	**Common position**
Safety demonstration plan	1.1.3.3
System upgrades	1.1.3.11
Assessing pre-existing software	1.4.3.1 and 1.4.3.4
Tool qualification	1.5.3.2
Operating experience	1.14.3.1 to 1.14.3.9
Software change procedure	2.7.3.5
Version control procedure	2.7.3.19
Periodic testing	2.8.3.1

2.8.4 Recommended Practices

Periodic testing

2.8.4.1 The quality of the computer equipment and software used for periodic testing functions should comply with the recommendations for tools mentioned in Chapter 1.5 (see in particular 1.5.3.2).

Generation of calibration data

2.8.4.2 Unless the calibration data is generated totally automatically via a system that has been developed to the same standards as the safety systems then there should be a diverse means for the production of the data or an independent verification of the data. This diverse means of verification may be either manual or automatic but it should be demonstrated that the combination of the principal and the diverse means of verification forms a sufficiently high integrity process so as not to degrade the safety system. All calculations should be documented and retained for future audit.

2.8.4.3 The principal means for the generation of the data should be computer assisted so as to reduce the potential for the introduction of human error. The software should be produced to high quality industrial standards. Verification check points should be introduced at convenient points.

Loading of calibration data

2.8.4.4 The process of loading and changing the calibration data (eg in the case of changing the operational mode) should be computer assisted using pre-calculated and fixed stored calibration data sets. The software for loading calibration data should be to the same standards as the safety system software.

*   *   *

References

1 EUR18158. 1998. European nuclear regulators' current requirements and practices for the licensing of safety critical software for nuclear reactors. European Commission, DG Environment, Nuclear safety and Civil Protection, Report EUR18158 (revision 8), 1998.

2 EUR 19265 EN. 2000. Common position of European nuclear regulators for the licensing of safety critical software for nuclear reactors. Nuclear safety, regulation and radioactive waste management unit of the European Directorate General for the Environment. May 2000.

3 US Nuclear Regulatory Commission, NUREG/IA-0463. (Availability of) An International Report on Safety Critical Software for Nuclear Reactors by the Regulator Task Force on Safety Critical Software (TF-SCS). December 2015. **A**gencywide **D**ocuments **A**ccess and **M**anagement **S**ystem" (ADAMS), Accession Number ML15348A206.

4 IEC 61513. 2nd edition (2011-08). Nuclear power plants – Instrumentation and control important to safety – General requirements for systems.

5 IEC 60880. 2nd edition (2006-05). Nuclear power plants – Instrumentaton and control systems important to safety – Software aspects for computer-based systems performing category A functions.

6 Courtois, P.-J. 2008. "Justifying the dependability of computer-based system. With applications in nuclear engineering". Springer Series in Reliability Engineering. 323pp. ISBN 978-1-84800-371-2

7 IAEA NP-T-3.27. Dependability assessment of software for safety instrumentation and control systems at nuclear power plants, Nuclear Energy Series, July 2018.

8 ISO/IEC 15026-2. 2022. Systems and software engineering – Systems and software assurance – Part 2: Assurance case.

9 IEC 60880. 1986. Software for computers in the safety systems of nuclear power stations.

10 Rushby, J. 1993. "Formal Methods and the Certification of Critical Systems". Technical Report CSL-93-7, SRI International.

11 Knight and Leveson. "An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming". IEEE Transactions on Software Engineering, SE-12 (1), pp.96-109, 1986.

12 Littlewood, B., Strigini, L. "Validation of Ultra-High Dependability for Software-based Systems, City University". Communications of the ACM, November 1993, pp. 69-80.

13 Voges, U. “Software diversity”, Proceedings 9th Annual Conference on Software Safety, Luxembourg, 7-10 April 1992. Ed. by the Centre for Software Reliability, City Univ., London.

14 IEC 61508. 2nd edition (2010-04). Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General Requirements, Part 3: Software Requirements.

15 IEC 61226. 3rd edition (2009-07). Nuclear power plants – Instrumentation and control systems important to safety – Classification of instrumentation and control functions.

16 IAEA Specific Safety Guide. Design of Instrumentation and Control Systems for Nuclear Power Plants, SSG-39, 2016.

17 Ehrenberger, W.D. “Probabilistic Techniques for Software Verification”. Paper for IAEA Technical Committee meeting on Safety Implications of Computerised Process Control in Nuclear Power Plants, Vienna, Austria, November 1989

18 Miller W.K., et al. “Estimating the Probability of Failure When Testing Reveals no Failures”. IEEE Transactions on Software Engineering, Vol. 18 No.1, pp.33-43, January 1992.

19 Littlewood B., Popov P., and Strigini L., “Assessment of the Reliability of Fault-Tolerant Software: a Bayesian Approach”, in 19th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2000).

20 Parnas, D.L., Asmis, G.J.K., Madey, J. 1991 Assessment of safety critical software in nuclear power plants. Nuclear Safety, Vol. 32 No. 2.

21 IAEA, Technical Reports Series. Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control, TRS N°384, 1999.

22 IAEA Technical Reports Series. Software Important to Safety in Nuclear Power Plants, TRS N°367, 1994.

23 Perry, W. 1995. Effective methods for Software Testing. John Wiley.

24 IAEA Safety Guide. Commissioning for Nuclear Plants, Safety Standards Series N° NS-G-2.9, 2003.

25 IAEA Nuclear Security Series No 17 – Computer Security at Nuclear Facilities, 2011.

26 IEC 62645 – Nuclear power plants – Instrumentation and control systems – Requirements for security programmes for computer-based systems.

27 ISA/IEC 62443 standards series.

28 CSA N290.7 – Cyber security for nuclear power plants and small reactor facilities.U.S. NRC, NUREG/CR-7006 (ORNL/TM-2009/20), Review Guidelines for Field-Programmable Gate Arrays in Nuclear Power Plant Safety Systems, 2010.

29 IAEA Nuclear Energy Series No. NP-T-3.17, Application of Field Programmable Gate Arrays in Instrumentation and Control Systems of Nuclear Power Plants, 2016.

30 IEC 62566, Edition 1.0, Nuclear power plants – Instrumentation and control important to safety – Development of HDL-programmed integrated circuits for systems performing category A functions, 2012.

31 OECD/NEA MDEP Generic Common Position No. DICWG-05, Common Position on the treatment of hardware description language (HDL) programmed devices for use in nuclear safety systems, http://www.oecd-nea.org/mdep/common-positions/.

32 PD IEC/TR 63084 Nuclear power plants – Instrumentation and control important to safety – Platform qualification for systems important to safety.

33 WGDIC-CP14, Consensus Position on "Qualification of I&C Platforms for Use in Systems Important to Safety", NEA/CNRA/R(2018)3.

* * *

Bibliography

Basic safety Rule II.4.1.a. "Software for Safety Systems". France.

CEMSIS. Cost Effective Modernisation of Systems Important to Safety project. Pavey D., R. Bloomfield, P.-J. Courtois et al. Pre-proceedings of FISA-2003. EU Research in Reactor Safety. Luxemburg, 10-13 November 2003. EUR 20281, pp.301-305.

DeVa. Design for Validation. European Esprit Long Term Research Project 20072. Third Year Report. Deliverables, Parts 1 and 2. December 1998. Published by LAAS-CNRS, Toulouse, France.

Dobbing, A., et al. Reliability of SMART Instrumentation. National Physical Laboratory and Druck Ltd, 1998.

EUR 17311 EN. 1997. Dependability of Extensively Deployed Products with Embedded IT. Brussels Workshop, November 1996 Main Report. M. Masera, M. Wilikens, P. Morris, Ispra, Italy.

HSE 1998. The Use of Computers in Safety Critical Applications. Final Report of the study group on the safety of operational computer systems. 1998. HSE books, UK.

IAEA Safety Glossary, Termination Used in Nuclear Safety and Radiation Protection, 2007 Edition.

IAEA Safety Guide, Instrumentation and Control Systems Important to Safety in Nuclear Power Plants. Safety Standards Series N° NS-G-1.3, 2002.

IEC 60231. 1967. General principles of nuclear reactor instrumentation.

IEC 60231 A. 1969. General principles of nuclear reactor instrumentation. First supplement.

IEC 60639 1998. Nuclear power plants – Electrical equipment of the safety system – Qualification.

IEC 60709 Ed. 2. 2004. Nuclear power plants – Instrumentation and control systems important to safety – Separation.

IEC 60987. 1989. Programmed digital computers important to safety for nuclear power stations.

IEC 60987 Ed. 2. 2007. Nuclear power plants – Instrumentation and control important to safety – Hardware design requirements for computer-based systems.

IEC 62138. 2004. Nuclear power plants – Instrumentation and control important for safety – Software aspects for computer-based systems performing category B or C functions.

IEEE Std 7-4.3.2-1993. IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations.

IEEE Std 730.1-2002. IEEE Standard for Software Quality Assurance Plans.

N290.14-07. Qualification of pre-developed software for use in safety-related instrumentation and control applications in nuclear power plants. CSA Standards Update Service. July 2007.

Randell, B., et al (eds). 1995. Predictably dependable computing systems. EUR 16256EN, Springer-Verlag.

Saglietti, F. 2004. Licensing reliable embedded software for safety-critical applications. Real-Time Systems, 28, 217-236.

Sommerville, I. 1985 Software Engineering, Ed. 2 Addison-Wesley International Computer Science Series. Boston.

Stankovic J.A. Wireless Sensor Networks. IEEE Computer October 2008.

Tanenbaum, A.S. 1992. Modern Operating Systems. Prentice Hall.

Tanenbaum, A.S. 1995. Distributed Operating Systems. Prentice Hall.

U.S. Nuclear Regulatory Commission NUREG/CR-6303, Method for Performing Diversity and Defense-in-Depth Analyses of Reactor Protection Systems. December 1994.

U.S. Nuclear Regulatory Commission NUREG/CR-6463, Review Guidelines on Software Languages for Use in Nuclear Power Plant Safety Systems. June 1996.

U.S. Nuclear Regulatory Commission NUREG/CR-7006, Guidelines for Field-Programmable Gate Arrays in Nuclear Power Plant Safety Systems Plant. February 2010.

U.S. Nuclear Regulatory Commission NUREG/CR-7007, Diversity Strategies for Nuclear Power Plant Instrumentation and Control Systems. February 2010.

*   *   *

Vous dépendez, dans une affaire qui est juste et importante,
du consentement de deux personnes. L'un vous dit:"J'y donne les mains pourvu qu'un tel condescende"; et ce tel condescend
et ne désire plus que d'être assuré des intentions de l'autre.
Cependant rien n'avance; les mois, les années s'écoulent inutilement:
"Je m'y perds, dites-vous, et je n'y comprends rien;
il ne s'agit que de faire qu'ils s'abouchent et qu'ils se parlent."
Je vous dis moi, que j'y vois clair, et que j'y comprends tout:
ils se sont parlé.

La Bruyère, *Les caractères de la cour*, (1688, 1st version – 1694, 8th and last version)