Licensing of safety critical software for nuclear reactors

Common position of seven European nuclear regulators and authorised technical support organisations

BEL V, Belgium

BfS, Germany

CSN, Spain

ISTec, Germany

ONR, United Kingdom

SSM, Sweden

STUK, Finland

REVISION 2013

Disclaimer

Neither the organisations nor any person acting on their behalf is responsible for the use which might be made of the information contained in this report.

The report 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 http://www.belv.be/	**Health and Safety Executive** **Office for Nuclear Regulation (ONR)** Redgrave Court, Merton Road, Bootle, Merseyside, L20 7HS, UK http://www.hse.gov.uk/nuclear
Federal Office for Radiation Protection (BfS) P.O. Box 100149 D-38201 Salzgitter, Germany http://www.bfs.de/Kerntechnik/	**Strålsäkerhetsmyndigheten (SSM)** **Swedish Radiation Safety Authority** SE-17116 Stockholm Sweden http://www.ssm.se
Consejo de Seguridad Nuclear (CSN) C/ Justo Dorado 11 28040 Madrid Spain http://www.csn.es/	**STUK • Radiation and Nuclear Safety Authority** Laippatie 4, P.O. Box 14 FIN-00881 Helsinki, Finland http://www.stuk.fi/
Institut für Sicherheitstechnologie (ISTec) GmbH Forschungsgelände D-85748 Garching b. München, Germany http://www.istec.grs.de/	

Contents

Executive Summary

Objectives

It is widely accepted that the assessment of software cannot be limited to verification and testing of the end product, i.e. 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 limited guidance on the regulatory and safety assessment of these factors. An undesirable consequence of this situation is that the licensing approaches taken by nuclear safety authorities and by technical support organisations are determined independently with only limited informal 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 difficulties in co-ordinating the development and qualification process.

It was thus felt necessary to compare the respective licensing approaches, to identify where a consensus already exists, and to see how greater consistency and more mutual acceptance could be introduced into current practices. Within this comparison, the term software also includes firmware and microcode.

This document is the result of the work of a group of regulator and safety authorities' experts. The 2007 version was completed at the invitation of the Western European Nuclear Regulators' Association (WENRA). The major result of the work is the identification of consensus and common technical positions on a set of important licensing issues raised by the design and operation of computer based systems used in nuclear power plants for the implementation of safety functions. The purpose is to introduce greater consistency and more mutual acceptance into current practices. To achieve these common positions, detailed consideration was paid to the licensing approaches followed in the different countries represented by the experts of the task force.

The report is intended to be useful:

- to coordinate regulators' and safety experts' technical viewpoints 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 and major I&C suppliers on the international market.

Document Structure

From the outset, attention focused on computer based systems used in nuclear power plants for the implementation of safety functions (i.e. the functions of the highest safety criticality level); namely, those systems classified by the International Atomic Energy Agency as "safety systems". The recommendations of this report therefore mainly address "safety systems"; "safety related systems" are addressed in certain common positions and recommendations only where explicitly mentioned.

In a first stage of investigation, the task force identified what were believed to be, from a regulatory viewpoint, some of the most important and practical issue areas raised by the licensing of software important to safety. In the second stage of the investigation, for each issue area, the task force strove for and reached: (1) a set of *common positions* on the basis for licensing and evidence which should be sought, (2) consensus on best design and licensing *recommended practices*, and (3) agreement on certain alternatives which could be acceptable.

The common positions are intended to convey the unanimous views of the Task Force members on the guidance that the licensees need to follow as part of an adequate safety demonstration. Throughout the document these common positions are expressed with the auxiliary verb "shall". The use of this verb for common positions is intended to convey the unanimous desire felt by the Task Force members for the licensees to satisfy the requirements expressed in the clause. The common positions are a common set of requirements and practices considered necessary by the member states represented in the task force.

There was no systematic attempt, however, at guaranteeing that for each issue area these sets are complete or sufficient. It is also recognised that – in certain cases – other possible practices cannot be excluded, but the members felt that such alternatives will be difficult to justify.

Recommended practices are supported by most, but may not be systematically implemented by all of the members states represented in the task force. Recommended practices are expressed with the auxiliary verb "should".

In order to avoid the guidance being merely reduced to a lowest common denominator of safety (inferior levelling), the task force – in addition to commonly accepted practices – also took care not to neglect essential safety or technical measures.

Background (history)

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 report [4] 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 [5]). 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 when the NRWG was disbanded by the EC, the task force was invited by the WENRA association to pursue and complete the 2007 version of this report. The common positions and recommended practices of EUR 19265 [5] are included. The task force continues to work on missing and emerging licensing aspects of safety critical software.

The U.S. Nuclear Regulatory Commission (NRC) has, since 2009, participated in the meetings of the task force and provided input to this version of the report. Although the NRC has not endorsed this report, it plans to continue to participate in task force meetings to inform the task force of NRC technical positions on matters of interest, to better understand the technical views and common positions and to better inform the NRC of the task force's position on technical positions of mutual interest.

* * *

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 assessment of software cannot be limited to verification and testing of the end product, i.e. 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 limited guidance on the regulatory and safety assessment of these factors. An undesirable consequence of this situation is that the licensing approaches taken by nuclear safety authorities and by technical support organisations are determined independently and with only limited informal 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 difficulties in co-ordinating the development and the qualification process.

It was thus felt necessary to compare the respective licensing approaches, to identify where a consensus already exists, and to see how greater consistency and more mutual acceptance could be introduced into the current practices. Within this comparison, the term software also includes firmware and microcode.

This document is the result of the work of a group of regulator and safety authorities' experts. The 2007 version was completed at the invitation of the Western European Nuclear Regulators' Association (WENRA). The major result of the work is the identification of consensus and common technical positions on a set of important licensing issues raised by the design and operation of computer based systems used in nuclear power plants for the implementation of safety functions. The purpose is to introduce greater consistency and more mutual acceptance into current practices. To achieve these common positions, detailed

consideration was paid to the licensing approaches followed in the different countries represented by the experts of the task force.

The report is intended to be useful:

- to coordinate regulators' and safety experts' technical viewpoints 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 and major I&C suppliers on the international market.

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 (i.e. the functions of the highest safety criticality level); namely, those systems classified by the International Atomic Energy Agency as "safety systems". Therefore, *recommendations of this report – except those of chapter 1.11 – primarily address "safety systems" and not "safety related systems"*.

In certain cases, in the course of discussions, the task force came to the conclusion that specific practices were clearly *restricted* to safety systems and could be dispensed with – or at least relaxed – for safety related systems. Reporting the possibility of such dispensations and relaxations was felt useful should there be future work on safety related systems. These practices are therefore explicitly identified as applying to safety systems. Some relaxations of requirements for safety related systems are also mentioned in chapter 1.2. All these relaxations for safety related systems 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. Moreover, it was felt that the most difficult aspects of the licensing of digital programmable systems are rooted in the specific properties of the technology. The objective was therefore to delineate practical and technical licensing guidance, rather than discussing or 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.

This report represents the consensus view achieved by the experts who contributed to the task force. It is the result of what was at the time of its initiation a first attempt at the international level to achieve consensus among nuclear regulators on practical methods for licensing software based systems.

This document should neither be considered as a standard, nor as a new set of European 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 no essential divergence with the common positions. It is precisely from this common agreement that regulators can draw support and benefit when assessing safety cases, licensee's submissions, and issuing regulations. The document is also useful to licensees, designers, suppliers for issuing bids and developing new applications.

Safety Plan

Evidence to support the safety demonstration of a computer based digital system is produced throughout the system life cycle, 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 independent types of evidence can and must be produced: evidence related to the quality of the development process; evidence related to the adequacy of the product; and evidence of the competence and qualifications of the staff involved in all of the system life cycle phases. In addition, convincing operating experience may be needed to 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 plan[1] shall be agreed upon at the beginning of the project between the licensor 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, and how and when this evidence shall be produced.*

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

This report neither specifies nor imposes the contents of a specific safety plan. All the subsequent recommendations are founded on the premise that a safety 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 plan. It is therefore implied that all the evidence and documentation recommended by this report, among others that the regulator may request, should be made available to the regulator.

The safety 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 Life Cycle Specific

As described earlier, in a first stage, the task force selected a set of specific technical issue areas, which were felt to be of utmost importance to the licensing process. In a second stage phase, each of these issue areas was studied and discussed in detail until a common position was reached.

These issue areas were partitioned into two sets: “generic licensing issues” and “life cycle 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 report, namely:

PART 1: GENERIC LICENSING ISSUES

1.1 Safety Demonstration

1.2 System Classes, Function Categories and Graded Requirements for Software

1.3 Reference Standards

1.4 Pre-existing Software (PSW)

1.5 Tools

1.6 Organisational Requirements

1.7 Software Quality Assurance Programme and Plan

1.8 Security

1.9 Formal Methods

1.10 Independent Assessment

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

1.12 Software Design Diversity

1.13 Software Reliability

1.14 Use of Operating Experience

1.15 Smart Sensors and Actuators

PART 2: LIFE CYCLE PHASE LICENSING ISSUES

2.1 Computer Based System Requirements

2.2 Computer System Architecture and Design

2.3 Software Requirements, Architecture and Design

2.4 Software Implementation

2.5 Verification

2.6 Validation and Commissioning

2.7 Change Control and Configuration Management

2.8 Operational Requirements

This set of issue areas is felt to address a consistent set of licensing aspects right from the inception of the life cycle up to and including commissioning. It is important to note, however, that although the level of attention given in this document may not always reflect it, a balanced consideration of these different licensing aspects is needed for the safety demonstration.

Definition of Common Positions and Recommended Practices

Apart from chapter 1.3 which describes the standards in use by members of the task force, for each issue area covered, the following four aspects have been addressed:

- Rationale: technical motivations and justifications for the issue from a regulatory point of view;
- Description of the issue in terms of the problems to be resolved;
- *Common position* and the evidence required;
- *Recommended practices*.

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

The use of the verb "shall" for common positions is intended to convey the unanimous desire felt by the Task Force members for the licensees to satisfy the requirement expressed in the clause. The use of "shall" and "requirement" shall not necessarily be construed as a mandatory condition incorporated in regulation.

These common position clauses can be regarded as a common set of requirements and practices in member states represented in the task force. The common position (set of clauses) for a particular issue is identified as necessary, but may not be sufficient or complete. It should also be recognised that – in certain cases – other possible practices cannot be excluded, but the members felt that such alternatives would be difficult to justify.

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

In order to avoid the guidance being merely reduced to a lowest common denominator of safety (inferior levelling), the task force – in addition to commonly accepted practices – also took care not to neglect essential safety or technical measures.

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 report 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 report [4] 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 [5]). 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 when the NRWG was disbanded by the EC, the task force was invited by WENRA to pursue and complete the 2007 version of this report. The common positions and recommended practices of EUR 19265 [5] are included. The task force continues to work on missing and emerging licensing aspects of safety critical software.

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

Belgium	P.-J. Courtois, BEL-V (1994-) (Chairman 1994-2007)
	A. Geens, AVN (2004-2006)
	S. van Essche, BEL-V (2011-)
Finland	M.L. Järvinen, STUK (1997-2003)
	P. Suvanto, STUK (2003-2010)
	M. Johansson, STUK (2010-)
Germany	M. Kersken, ISTec (1994-2003)
	E.W. Hoffman, ISTec (2003-2007)
	J. Märtz, ISTec (2007-)
	F. Seidel, BfS (1997-)
Spain	R. Cid Campo, CSN (1997-2003)
	F. Gallardo, CSN (2003-)
Sweden	B. Liwång, SSM (1994-)
United Kingdom	R. L. Yates, ONR (1999-) (Chairman 2007-)
	M. Bowell, ONR (2007-)

A consortium consisting of ISTec, NII, and AVN (chair) was created in March 1998 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 R. L. 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, J.C. Schwartz, H. Zatlkajova from the EC, and G. Cojazzi and D. Fogli from JRC, Ispra. P. Govaerts (AVN) was instrumental in setting up the task force in 1994.

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 U.S. Nuclear Regulatory Commission (NRC) has, since 2009, participated in the meetings of the task force and provided input to this version of the report. The NRC participants in task force meetings have included Dr. Steven Arndt, William Kemper, Norbert Carte, and Dr. Sushil Birla. Additionally, Michael Waterman and Milton Concepcion of the NRC staff have provided input. Although the NRC has not endorsed this report, it plans to continue to participate in task force meetings to inform the task force of NRC technical positions on matters of interest, to better understand the technical views and common positions and to better inform the NRC of the task force's position on technical positions of mutual interest.

* * *

II GLOSSARY

The following terms should be interpreted as having the following meaning in the context of this document.

These terms are highlighted as a defined term the first time they are used in each chapter of this document.

(Operating) Availability: Fraction of time the *system* is operational and delivers service (readiness for delivering service).

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 Safety Guide NS-G_1.3)

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*, *channels* or *components* due to a single specific event or cause. (IAEA Safety Guide NS-G-1.3)

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.

Component: One of the parts that make up a *system*. A component may be hardware or software and may be subdivided into other components. (IEC 61513, 3.9)

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, I/O 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 *channel* architecture of the *system*; 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)

Dependability: Trustworthiness of the delivered service (eg a safety function) such that reliance can justifiably be placed on this service. *Reliability*, *availability*, *safety* are example attributes of dependability.

Diversity: Existence of two or more different ways or means of achieving a special objective. (IEC 60880 Ed. 2 [12])

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 coincident *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: Inability of a structure, *system* or *component* to function within acceptance criteria. (IAEA Safety Glossary). Note that in this document the acceptance criteria are the specified functional or non-functional requirements.

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.

Functional diversity: Application of *diversity* at the functional level (for example, to have trip activation on both pressure and temperature limit). (IEC 60880 Ed.2 (3.18) [12] and IEC61513 (3.23) [15])

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)

I&C: instrumentation and control.

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: requirements under which the *system* is required to operate or exist, or system properties. Examples are quality requirements and human factor requirements. Quality requirements include transportability, survivability, flexibility, portability, reusability, reliability, maintainability, and security. (ISO/IEC/IEEE 29148)

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 (pfy) (eg 10^{-4} pfd or 10^{-4} pfy).

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, autotests, communication, information displays. Examples are relays, recorders, regulators, smart sensors and actuators.

PSA: Probability *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)

In this document nuclear safety is abbreviated to safety.

Safety demonstration: The set of arguments and evidence elements which support a selected set of claims on the *dependability* – in particular the *safety* – of the operation of a *system important to safety* used in a given plant environment.

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 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 plan is not necessarily a specific document.

Safety related systems: Those instrumentation and control *systems important to safety* that are not included in *safety systems* (IAEA safety guide NSG 1.3).

Safety system: A *system important to safety* provided to assure the safe shutdown of the reactor and the heat removal from the core, or to limit the consequences of operational occurrences and accident conditions. (IAEA safety guide NSG 1.3)

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 amongst 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.

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 [5]).

System: When used as a stand-alone term, abbreviation for *computer based system*.

Systems important to safety: *Systems* which include *safety systems* and *safety-related systems*. In general, all those items which, if they were to fail to act, or act when not required, may result in the need for action to prevent undue radiation exposure (IAEA safety guide NSG 1.3).

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 (of the requirements): Demonstration of the correctness, completeness and consistency of a set of requirements.

Validation (of the system): Obtaining evidence, usually by testing, that the integrated hardware and software will operate and deliver service as required by the *functional* and *non functional computer based system requirement specifications*. The validation of computer based systems is usually performed off-site.

Verification: Checking or testing that the description of the product of a design phase – for the coding phase this is the actual product – is consistent and complete with respect to its specification, which is usually the output of a previous phase.

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 these requirements may not be sufficient to assure safety in all cases. They usefully describe what is recommended in fields such as requirements specification, design, verification, validation, maintenance, operation, etc. and contribute to the improvement of safety demonstration practices.

However, the process of 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, which formalise negotiations between licensee and licensor, 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 and cost-effectiveness of the safety demonstration. The benefit can be at least three-fold:

- To allow the parties involved to focus attention on the specific safety issues raised by the system and on the corresponding specific system requirements as defined in chapter 2.1 that must be satisfied;
- To allow system requirements to be prioritised, with a commensurate allocation of resources;
- To organise system requirements so that the arguments and evidence are limited to what is deemed necessary and sufficient by the parties involved.

A safety demonstration addresses the properties of a particular system operating in a specific environment. It is therefore specific and carried out on a case-by-case basis, and not once and for all. This does not mean, however, that the demonstration could be performed "à la carte" with a free choice of means and objectives.

1.1.2 Issues Involved

1.1.2.1 Various approaches are possible

There are several approaches offered to a licensee and a regulator 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 (rule-based approach). It also may be conditioned on the provision of evidence that certain specific residual risks are acceptable, or that certain safety properties are achieved (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.

None of these approaches is without problems. 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 three different and orthogonal directions, which remain unrelated, one may 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.

1.1.2.2 The plant-system interface cannot be ignored

Most safety requirements are determined by 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.

1.1.2.3 The system lifetime must be covered

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 life cycle, and well known to application engineers, but often not sufficiently addressed in the computer system design.

Besides, as already said in the introduction, evidence to support the safety demonstration of a computer based system is produced throughout the system life cycle, and evolves in nature and substance with the project.

1.1.2.4 Safety Demonstration

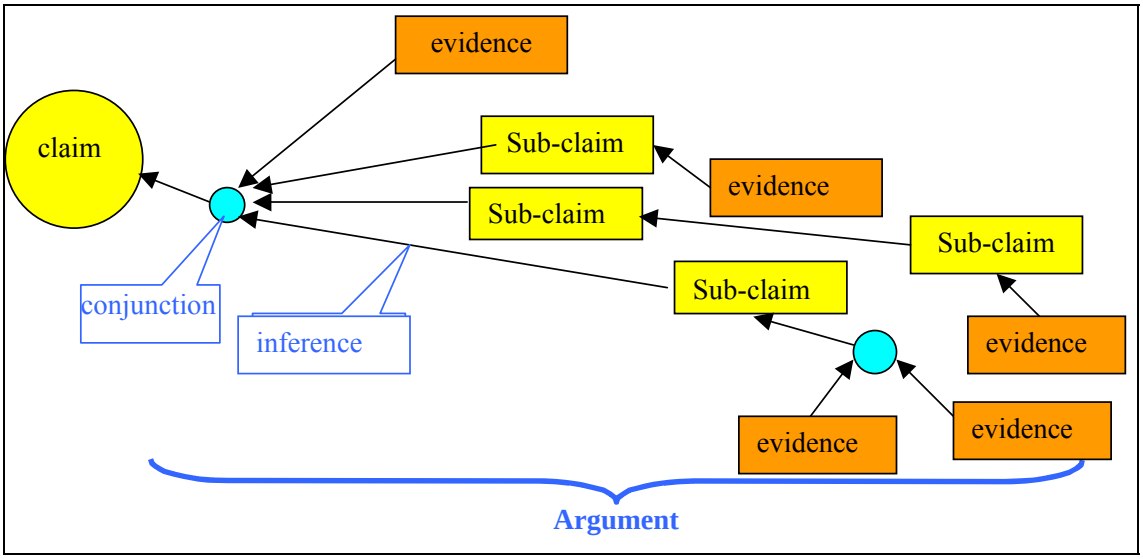
The key issue of concern is how to demonstrate that the system requirements as defined in chapter 2.1 have been met. Basically, a safety demonstration is a set of arguments and evidence elements that support a selected set of dependability claims– in particular the safety – of the operation of a system important to safety used in a given plant environment. (see Figure 1 below and eg [2]).

Claims identify functional and/or non-functional properties that must be satisfied by the system. A claim may require the existence of a safe state, the correct execution of an action, a specified level of reliability or availability, etc… The set of claims must be coherent and as complete as possible. By itself, this set defines what the expected dependability of the system is. Claims can be decomposed 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 COTS for instance.

Claims and sub-claims are supported by evidence components that identify facts or data, for which there is confidence beyond any reasonable doubt in the axiomatic truth of these facts and data, without further evaluation, quantification or demonstration. Such a confidence inevitably requires a consensus of all parties involved to consider the evidence as being unquestionable. As already stated in the introduction, a number of distinguishable and independent types of evidence exist on which the demonstration can be constructed: evidence related to the quality of the development process; evidence related to the adequacy of the product specifications and the correctness of its implementation, and evidence of the competence and qualifications of the staff involved in all of the system life cycle phases. Convincing operating experience may be needed to support the safety demonstration of pre-existing software (see Chapter 1.14). These types of evidence may not be merely juxtaposed. They must be organised so as to achieve the safety demonstration.

An *argument* is the set of evidence components that support a claim, together with a specification of the relationship between these evidence components and the claim.

Figure 1: Claim, arguments and evidence structure



1.1.2.5 System descriptions and their interpretations are 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 and 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 using *accurate descriptions of the system architecture, of the hardware and software design of the system behaviour and of its interactions with the environment, and using models of postulated accidents*. These descriptions must include unintended systems behaviour, be unambiguously understood and agreed upon by all those who have dependability case responsibilities: users, designers and assessors. This is unfortunately not always the case. Claims for dependability – although usually based on a huge engineering and industrial past experience – may be only poorly specified. The simplifying assumptions behind the descriptions that are used in the system and environment representations and for the evaluation of safety are not always sufficiently explicit. As a result one should be wary of attempts to "juggle with assumptions" (i.e. to argue and interpret system, environment and/or accident hypotheses in order to make unfounded claims of increased safety and/or reliability) during licensing negotiations. A need exists in industrial safety cases for more attention to be paid to the use of accurate descriptions of the system and of its environment.

It is worth noting that the software itself is the most accurate available description of the behaviour of the computer.

1.1.3 Common Positions

1.1.3.1 A safety plan shall be agreed upon at the beginning of the project between the licensor 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, and how and when this evidence shall be produced. A safety 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 software identified shall be covered by a safety plan.

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

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

1.1.3.5 The safety plan shall define

- the activities to be undertaken in order to demonstrate that the system is adequately safe for its intended use and environment,
- the organisational arrangements needed for the demonstration (including independence of those undertaking the safety demonstration activities) and
- the programme (the activities and their inter-relationships, allocated resources and time schedule) for the safety demonstration.

1.1.3.6 Production of three different types of evidence shall be addressed in the safety plan:

- evidence related to quality of the development process
- evidence related to adequacy of the product
- evidence of the competence and qualification of the staff involved in all of the system life cycle phases.

1.1.3.7 The safety plan shall be implemented by the licensee.

1.1.3.8 If the safety demonstration is based on a claim/evidence/argument structure, then the safety 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 requirements that need to be satisfied. These requirements shall address at least:

- The validity of the functional and non-functional system 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;
- The correctness of the design and the implementation of the embedded computer system for ensuring that it performs according to its specifications;
- 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 timing constraints etc.

1.1.3.12 If a claim/evidence/argument structure is followed, the safety demonstration shall accurately document 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 guidelines that are used for the safety demonstration. The applicability of the standards to be used shall be justified, with potential deviations being evaluated and justified.

1.1.3.14 When standards are intended to support specific claims or evidence components, this shall be indicated. Guarantee shall also be given that the coherence of the regulations, standards, or guidelines that are used is preserved.

1.1.3.15 If a claim/evidence/argument 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 documented in the safety demonstration.

1.1.3.16 The system descriptions used to support the safety demonstration shall accurately describe the system architecture, system/environment interface, interaction, constraints and assumptions, the system design, the system hardware and software architecture, and the system operation and maintenance.

1.1.3.17 The safety 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 plan and safety demonstration shall be updated and maintained in a valid state throughout the lifetime of the system.

1.1.4 Recommended Practices

1.1.4.1 A safety claim 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:

- Sub-claims that the functional and/or non-functional requirement specifications of how the system has to deal with the hazard/threat are valid,
- Sub-claims that the system architecture and design correctly implement these specifications,
- and sub-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 sub-claims from which the dependability 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 (i.e. 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 NS-G-1.3):

– 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 in EUR countries and elsewhere.

The three-level system classification scheme serves the purpose of this document, which is principally focusing on software of the highest safety criticality (i.e. 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 common position 1.2.3.9, 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 errors, 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 plan at the beginning of the project.

1.2.3.4 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.

1.2.3.5 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)

1.2.3.6 Common position 1.12.3.11 is applicable.

1.2.3.7 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

- the data produced is independently checked by a diverse method, or
- 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 2.8.3.3.4 restates this position for the loading of calibration data.)

1.2.3.8 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

1.2.3.9 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.

1.2.3.10 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.

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

1.2.3.12 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 plan.

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

1.2.3.14 The graded requirements (defined in section 1.2.1.2) that are applicable to safety systems include all "shall" requirements of the IEC 60880, ref. [11].

1.2.3.15 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:
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.
However, if reliability targets are used, exceptions to the common position 1.2.3.14 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, i.e. 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.
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 with 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, QA 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

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 common position 1.1.3.13 in chapter 1.1)

The applicable standards mentioned below in sections 1.3.1 and 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.

1.3.1 Reference standards for the software of safety systems in Belgium, Finland, Germany, Spain, Sweden and UK

At the time of writing this revision, the only common reference standard for the software of safety systems in all states represented in the task force is the IEC 60880 (1986) standard: "Software for computers in the safety systems of nuclear power stations" 1st edition [11].

1.3.2 Additional 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 1 (draft regulatory guide DG-1130, December 2004) of proposed revision 2 of US NRC Regulatory Guide 1.152: "Criteria for programmable digital computer system software in safety related systems of NPPs".
- IEC 61226, Edition 2 (2005) "Nuclear power plants – Instrumentation and control systems important to safety – Classification."

Finland

- Guide YVL 1.0 Safety criteria for design of nuclear power plants, Helsinki 12.1.1996.
- Guide YVL 2.0 Systems design for nuclear power plants, Helsinki 1.7.2002.
- Guide YVL 2.1 Nuclear power plant systems, structures and components and their safety classification, Helsinki 26.6.2000.
- Guide YVL 5.5 Instrumentation systems and components at nuclear facilities, Helsinki 13.9.2002.

Germany

- RSK Guidelines, Chapter 7.6, amended August 1996.
- DIN IEC 61226, 2nd edition (2005).
- DIN IEC 62138 (2004).
- DIN IEC 60880 (2007).
- KTA. 3503, November 2005, Type testing of electrical modules for the safety related instrumentation and control system.
- DIN VDE 0801, January 1990, Grundsätze für Rechner in Systemen mit Sicherheitsaufgaben.
- DIN V 19250 Grundlegende Sicherheitsbetrachtungen für MSR-Schutzeinrichtungen. January 1989.
- VDI/VDE 3527 Kriterien zur Gewährleistung der Unabhängigkeit von Sicherheits-funktionen bei der Leittechnik-Auslegung.

Spain

- UNESA CEN-6 “Guía para la implantación de sistemas digitales en centrales nucleares”, Revision 0, May 2002.
- UNE 73-404-91. “Garantia de la Calidad en los Sistemas Informaticos Aplicados a Instalaciones Nucleares”.
- CSN 10.9 Safety Guide. “Garantia de Calidad de las Aplicaciones Informaticas Relacionadas con las Instalaciones Nucleares”.
- ANSI/IEEE-ANS 7-4.3.2. “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”.
- USA 10 CFR 50.59 “Changes, tests and experiments”, particularly as derived for upgrading projects from EPRI TR-102348, Revision 1, NEI 01-01 “A Revision of EPRI TR-102348 to Reflect Changes to the 10 CFR 50.59 Rule”, according to USNRC Regulatory Issue Summary 2002-22.
- RSK Guidelines, Chapter 7.6, amended August 1996.

Sweden

- IAEA Safety Guide NS-G-1.1
- IEC 61226
- IEC 61513

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

- T/AST/046 Technical Assessment Guide “Computer Based Systems”.
- IAEA Safety Guide NS-G-1.1 “Software for Computer Based Systems Important to Safety in Nuclear Power Plants”.
- IEC 61513:2001 “Nuclear Power Plants – Instrumentation and control for systems 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 report, 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 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 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 which are attached to three safety categories (A, B, and C).

The second edition of IEC 60880 comprises both the first edition issued 1986 and its supplement issued 2000 with updated requirements covering the whole software life cycle (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, amongst 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: 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 above) and is being accommodated.

More information on the current work and publications of the IAEA and IEC can be found on their websites:

- http://www-pub.iaea.org/MTCD/publications/series1.asp
- http://www.iec.ch/ourwork/iecpub-e.htm

* * *

1.4 Pre-existing Software (PSW)

1.4.1 Rationale

The inclusion of appropriately verified pre-existing software (PSW) components into a computer based system may not only be beneficial for productivity but, if applied correctly, may also increase confidence that the system is safe. The benefit stems from the premise that PSW components have often been used in many applications, and their operating experience, when assessable and representative, can be taken into account. 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. Proper application of such components, supported with appropriate justification, may reduce the assessment effort. Licensees may wish to make use of such software, along with its previous assessment and with contributing evidence such as independent verification records and documented operating experience.

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 and non-functional (eg dependability, performance) behaviour 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 life cycle 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 QA 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:

1.4.3.9.1 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.

1.4.3.9.2 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.

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

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

1.4.3.9.5 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, error 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 based on the last release except if an adequate impact analysis shows that previous experience based on unchanged parts of the PSW is still valid because these parts have been unaffected by later releases.

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

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 dependability 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 dependability.

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, may not be explicit or even determined.

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 dependability requirements for all tools used for the development, operation and maintenance of the software shall be determined and the requirements of this section 1.5.3 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 dependability requirements for an individual tool will be a function of its impact on the target software, and of how other tools or processes may detect errors

introduced by that tool. The combined set of tools used in the development of safety system software shall provide the same level of dependability as the level required from the target software.

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

1.5.3.4 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.

1.5.3.5 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.

1.5.3.6 For safety system software, only tools that are validated against the requirements defined in common position 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. 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.

1.5.3.7 For safety system software, only tools that are validated against the requirements defined in common position 1.5.3.1 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

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

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:

- the output of a tool can be verified by inspection;
- 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;
- the tool itself can be certified as processing correctly all valid statements and combinations thereof contained in the definition of a programming language;
- the tool can be validated by appropriate profiles and a sufficient amount of previous operational experience.

1.5.4.2 Formal methods which are applied purely manually usually require the involvement of very well trained humans. In addition, the applied techniques are highly error-prone. 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:

- tools for editing and type-setting;
- tools for syntax-checking, data type-checking and consistency-checking;
- tools to prove properties of specifications and discharge proof obligations in refining specifications;
- tools to animate formal specifications;
- tools to derive usable software components automatically from formal specifications;
- configuration control tools to relate specifications to development and to keep track of the level of verification of each step;
- 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, i.e. 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 slipshod work 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 life-cycle.

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

1.6.3.2.1 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.

1.6.3.2.2 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.

1.6.3.2.3 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.

1.6.3.2.4 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 dependability of the safety system is not jeopardised.

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 produce the required dependability for the computer-based safety 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: it is felt that the final software based system will have a higher degree of dependability if the techniques of quality assurance are used. 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 non-functional attributes of reliability, availability, maintainability and usability, (which shall have been addressed in the specification of the non-functional requirements) 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 error 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 life cycle 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 licensor 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

The objective of information and system security is to guarantee and preserve the dependability of safety systems by preventing security incidents or by minimising their impacts. In this context, security seeks to prevent unauthorised accesses to information, software and data in order to ensure that three attributes are met, namely:

- the prevention of disclosures that could be used to perform mischievous, malicious or misguided acts which could lead to an accident or an unsafe situation (confidentiality),
- the prevention of unauthorised modifications (integrity),
- the prevention of unauthorised withholding of information, data or resources that could compromise the delivery of the required safety function at the time when it is needed (availability).

All systems important to safety in a licensed nuclear installation, be they conventional or computer-based, need to be protected from unauthorised access since such access could result in a system mal-functioning due to either intentional or unintentional interference. The consequence of such interference could be that an accident might be triggered or the system might fail to perform its intended protective action. The restriction of physical access through the use of locked doors and key interlock systems has been the traditional method, and still remains a major contributor to preventing unauthorised access to systems important to safety. However, with the introduction of computer based systems important to safety, there are now a number of security issues that are unique to the technology due to its programmability and communication facilities, and which need to be addressed.

In addition, there are other computer systems such as maintenance scheduling systems, fuel burn-up calculators and other computer codes which are used off-line in support of operational safety, computers used in support of emergency arrangements and dosimetry record systems, that could indirectly affect plant or individual health and safety, and that must be considered.

1.8.2 Issues Involved

1.8.2.1 So far as is reasonably practicable, there should be no significant increase in overall plant risk from computer based systems important to safety in a nuclear installation due to:

- intentional or reckless interference or misuse of these systems,
- well intentioned, but misguided, use by authorised persons of these systems,
- the security provisions themselves.

1.8.2.2 Information can be stored in computer central memories or mass storage devices, can be printed, and can be transmitted by various networks or movable media. This means that there is the potential for intentional or unintentional impairments and misuses by company employees or members of the public, authorised or not. Additionally, specific threats also exist from hackers, viruses, software logic bombs, software time bombs, Trojan horses, and other similar mechanisms.

1.8.2.3 Interfaces and data exchanges between systems important to safety and off-line maintenance systems are a security issue to be taken into account. Loading parameter values or calibration data and periodic testing equipments are examples of the use of such interfaces.

1.8.2.4 Information has to be protected by taking into account the different storage and communication media which are used, not only at the plant but also at a supplier site.

1.8.3 Common Position

The licensee of a nuclear installation shall adopt, as a minimum the following requirements.

1.8.3.1 General organisation

1.8.3.1.1 For all computer based systems important to safety, as a minimum, the security provisions shall incorporate the standard security measures that are regarded as good commercial practice against threats coming from unauthorised access, viruses, software logic bombs, Trojan horses, and other similar mechanisms.

1.8.3.1.2 The security provisions which address the procedures and the environment of the design and development of a computer based system important to safety shall be commensurate with the class of the target system.

1.8.3.1.3 The procedures and equipment for loading software into a system important to safety shall be included in any security evaluation.

1.8.3.1.4 Where systems important to safety interface to computer-based support systems, for instance for the updating of calibration data or plant maintenance activities, then, as a

minimum, good security practices shall be observed. Examples of such practices are: user identification and password protection, the installation of virus protection software (if relevant), and the standard physical restriction of access to only authorised persons.

1.8.3.1.5 Access shall be restricted (for example through the use of password protection and key lock systems) to only those parts of a system to which a person has authority.

1.8.3.1.6 A document on security shall be drawn up so as to identify the security risks, threats and vulnerabilities of the safety system. This document shall be used to derive the security design requirements of the computer-based safety system.

1.8.3.1.7 In this document allocation of information security responsibilities shall be defined. In particular, the licensee shall define security requirements that its contractors and service providers have to satisfy.

1.8.3.1.8 As necessary, security incidents shall be reported to the regulatory authority.

1.8.3.1.9 Special security attention shall be taken during vulnerable phases, for example, on the occasion of software or hardware changes.

1.8.3.1.10 For safety systems, security specialists in association with the safety specialists shall evaluate the adequacy of the security provisions.

1.8.3.2 Physical security measures

1.8.3.2.1 For safety systems, security threats shall be avoided by design where possible.

1.8.3.2.2 The safety system software (programs and fixed data, including operational data) shall be held in suitable fixed read only memory. A systematic comparison of this read only memory code with the source code shall be performed for the purpose of detecting unauthorised code.

1.8.3.2.3 No inward data transmission link into the safety system shall be allowed except the direct connection required by the specified local maintenance and testing equipment. These systems and other specified support systems shall have restricted access, for example, through the use of passwords and key locks.

1.8.3.2.4 Direct links from the safety system to equipment outside the plant shall be prohibited.

1.8.4 Recommended Practices

1.8.4.1 All changes of constants and parameters should be automatically recorded in a secure manner for subsequent auditing.

1.8.4.2 All authorised accesses should be recorded and all security attacks should be reported (automatically where possible) and investigated.

1.8.4.3 Education on the security policy should be imposed to all persons involved.

1.8.4.4 Constant vigilance is required. The security policy should address the security issue through appropriate training programmes and specifically targeted campaigns.

* * *

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, i.e. 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, dependability 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 [21].

1.9.2.3 Unfortunately, mathematical analysis has important 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 correctness of a system requirement specification, nor the intentions behind it.

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 properties of the system. These boundaries and properties shall be explicitly, unambiguously and accurately documented (see common position 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 1.1.3).

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 common position 2.1.3.1).

1.9.4 Recommended Practices

1.9.4.1 Criteria for the choice of formal methods

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

1.9.4.1.2 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.

1.9.4.1.3 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.

1.9.4.2 Verification when applying formal methods

1.9.4.2.1 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.

1.9.4.2.2 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.

1.9.4.2.3 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:

– Competent (commensurate to the nature and complexity of the system);

– Organisationally independent of the supplier(s) of the system (and of its safety demonstration), and
– 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 has achieved/maintained its required dependability.

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 properties or features of the system have not been adequately 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

1.11.1.1 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.

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

1.11.1.3 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.

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

1.11.1.5 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 common position 1.2.3.10, 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 common position 1.2.3.5, 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 recommended practice 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 common position 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 that are allowed by these sections shall be justified.

1.11.3.5 The common position 1.2.3.15 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 1.2.3.14
- 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 1.5.3.2, 1.5.3.3, 1.5.3.6

Chapter 1.6 on organisational requirements:

– Common positions 1.6.3.2.1, 1.6.3.2.2, 1.6.3.2.3
– Recommended practice 1.6.4.3

Chapter 1.8 on security:

– Common positions 1.8.3.1.6, 1.8.3.1.10 and all common positions in 1.8.3.2

Chapter 2.2 on computer system design:

– Common positions 2.2.3.6, 2.2.3.12

Chapter 2.4 on coding and programming directives:

– Recommended practices 2.4.3.1, 2.4.3.4.1, 2.4.3.4.2, 2.4.4.3.4

Chapter 2.5 on verification:

– Recommended practice 2.5.3.5.1

Chapter 2.6 on validation:

– Recommended practice 2.6.4.1

Chapter 2.7 on change control and configuration management:

– Common positions 2.7.3.1.2, 2.7.3.3.2, 2.7.3.3.3, 2.7.3.3.6, 2.7.3.4.3, 2.7.3.4.5
– Recommended practices 2.7.4.1, 2.7.4.2

Chapter 2.8 on operational requirements:

– Common positions 2.8.3.3.4, 2.8.3.4.2, 2.8.3.4.3, 2.8.3.5.1
– Recommended practices 2.8.4.2.1, 2.8.4.3

1.11.4 Recommended Practices

1.11.4.1 As mentioned in practice 1.2.4.4, it is recommended that the failure analysis required in common position 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, QA 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 paragraph 1.11.1.3. 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:

- IEC 62138 includes graded requirements for the qualification of software executing functions of the categories B and C defined in IEC 61226;
- 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	– QA plan (generic) covering the whole life cycle – Evidence of structured design approach – Configuration and software maintenance management after installation	– QA plan (specific to the project) covering the whole life cycle – 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 CCF; defence and mitigation countermeasures	– Analysis of impact on safety of the software component potential failures – Analysis of risk and consequences of software CCF; 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 (HW + SW) ∈ 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 (HW + SW) ∈ 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	– QA plan (generic) covering the whole life cycle	– QA plan (specific) covering the whole life cycle
		– Evidence of structured design approach	– Structured design approach (phases)
			– Reviews of ends of phases
		– Evidence of experience of staff	– Evidence of experience of staff
		– Configuration and software maintenance management after installation	– Configuration and software maintenance management after installation
Development	– Complete, clear, non-ambiguous and auditable specifications	– Complete, clear, non-ambiguous and auditable specifications	– Complete, clear, non-ambiguous and auditable specifications
		– Identifiability of safety related functions in the specifications	– Identifiability of safety and safety-related functions in the specifications
			– Traceability of specifications through the development
		– FMCA of hardware and software	– FMCA of hardware and software
		– Analysis of risk and consequences of software CCF; defence and mitigation countermeasures	– Analysis of risk and consequences of software CCF; defence and mitigation countermeasures
		– Autocontrol functions (HW + SW) $\in$ specifications	– Autocontrol functions (HW + SW) $\in$ specifications
			– Demonstration of complete coverage by autocontrol and periodic tests
		– Design/coding methods limiting faults	– Design/coding methods limiting faults (evidence of programming directives)
	– Standard tools	– Validated tools	– 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 Systems	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

In order to achieve high dependability, use is typically made of redundant systems and components. While identical redundancy is effective in guarding against random hardware failures, a common cause failure possibility arises from systematic failures, eg specification, design, implementation and maintenance errors etc. Diversity may be introduced to provide protection against common cause failures. Decisions are required as to the way in which the required diversity will be achieved. Attention has also to be given to the reliability that can be claimed following the introduction of diversity and the demonstration required to support such claims. 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 Systems' Architecture Software Considerations

When designing systems important to safety to provide plant functions such as control and shut down protection, important decisions have to be made as to what is an acceptable systems' architecture. This section addresses systems' architecture issues that are likely to influence the need for and nature of software design diversity.

One approach typically adopted during architecture design, to protect against the possibility of common cause failure, is to consider the use of multiple, possibly diverse, systems. Also consideration of the need for defence in depth such that a failure in one layer is compensated for in the overall systems' architecture may lead to the need for diverse, possibly software based, systems.

The number of systems, components or channels required, the degree of diversity between them, the selection of the technology for each of them, and possibly the apportionment of reliability targets, have to be addressed. One approach that may be adopted for one-out-of two systems where one of them is a computer based system is to employ a simple non-computer

based secondary system. Where multiple computer based systems, channels or components are employed, the issue of software diversity has to be considered and this is discussed further below.

1.12.2.2 Independence

The computer system architecture design should ensure that the required independence is maintained between systems, components or channels.

Many factors may affect and compromise independence; for example factors relating to the conceptual design, the programming language, practice, coding style, tools used for implementation, physical separation etc. Total independence, however, seldom exists; it is therefore extremely hard (usually impossible) to demonstrate, and so should not be required.

The objective is to prevent, as far as possible, defects that are potential common causes of unsafe failures. To this purpose, a discussion, common positions and recommendations on computer system architecture including use of physical separation and electrical isolation etc can be found in chapter 2.2 on computer system design. It is at the systems' 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 systems' architecture should employ the technique of defence in depth (eg to ensure overlapping 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 failures across all defence barriers utilising computer based systems should be analysed.

Decisions taken as to the devices used at the plant level (eg sensors and actuators) will also require careful consideration since modern instrumentation is increasingly utilising software components (eg 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 defence against coincident failures. 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 high levels of dependability expected of the implementation). Diversity is a second classical line of defence to improve dependability and to prevent coincident hardware or software failures from defeating safety functions.

The use of diverse channels, subsystems or components, therefore, neither compensates nor is an excuse for using hardware or software of inadequate quality. Diversity is intended, like defence in depth, to bring additional evidence on the absence of coincident failures, and not be a substitute for a lack of evidence that the hardware and software of the diverse components satisfy their own specifications.

Diverse unacceptable channels or components never make an acceptable system. By way of examples, a channel or component would be unacceptable if any the following were the case:

- it cannot be shown to satisfy its specification,
- there is insufficient evidence to justify its claimed reliability,
- there is insufficient evidence to justify adequate independence from all other components and/or channels claimed as diverse, with respect to potential causes of coincident 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 diverse 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 coincident software 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 expectation is that the faults introduced by the teams will be diverse and coincident failures across the computer based systems will be reduced. Unfortunately there is no precise way of determining the benefit that such diversity delivers. There are many different sources of potential coincident software failures and research in this field shows that statistical independence cannot always be assumed, see references [16], [18], [22]. 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 systems' architecture). As a result safety case arguments as to the reliability achieved by the use of software diversity are required.

1.12.2.5 Software Reliability Figure

For a discussion on the reliability figure to be used following introduction of software diversity, see chapter 1.13 on software reliability.

1.12.2.6 Diversity Design Options and Seeking Decisions

Options must be taken as to what are the most effective methods and techniques to ensure diversity of failure behaviour across diverse software programs. These choices are termed diversity seeking decisions. Current practice is founded on engineering judgment, which identifies what are considered to be the best means to force diversity with some limited support from research. 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 force 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 specifications, requirement methods and techniques such that coincident failures are unlikely. Another important but potentially limiting factor is the need to ensure the software has the greatest simplicity compatible with the design intent.

Amongst the various methods to force diversity the use of functional diversity is generally accepted as a particularly important measure (eg [1]; and [12], §13.4.1 which specifically recommends functional diversity implementation unless it is impossible or inappropriate; also see eg [22]). 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). The best means of implementing functional diversity has to be considered and this might include ensuring that at least one system utilises a simple function (eg temperature level trip) where the other(s) employ functions involving non trivial calculations. Functional diversity may also be used within a computer based system (i.e. across different channels or components utilising common hardware) but the means of implementing functional diversity and benefit that this approach delivers will need to be carefully considered (eg while the application software might be different the impact of identical components such as operating system software will have to be addressed).

1.12.2.7 Common Cause Failure Analysis

While diversity design options intend to utilise diverse software components, confidence is required that this objective has been satisfied. A common cause failure analysis of the diverse software components and their development arrangements should be performed to provide the required confidence. This analysis should address the potential for and implications of the use of common modules, algorithms and code across (and within multi-channel) systems. Further guidance on common cause failure analysis can be found in [12].

1.12.3 Common Positions

1.12.3.1 The decision by the licensee to use diversity, the type of diversity or the decision not to use diversity shall be justified.

1.12.3.2 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.

1.12.3.3 The specification of any diverse component shall be demonstrated to be adequate in the context of the intended application.

1.12.3.4 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.

1.12.3.5 Software satisfying the requirements of part 2 of this document shall be used in each system employing software diversity.

1.12.3.6 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.

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

1.12.3.8 The design of computer based systems utilising software diversity shall be justified in the context of the particular application, against current best practice which includes but is not limited to:

- functional diversity,
- technology diversity,
- independent teams for specifying, decomposing and deriving requirements,
- independent development teams with no direct communication,

- independent teams for performing verification and validation,
- different allocation of requirements to software components,
- different allocation of software components to hardware components,
- different timing and different order of execution,
- simplicity of software design and implementation,
- different description/programming languages and notations,
- different development methods,
- different development platforms, tools and compilers,
- different operating systems,
- different hardware,
- diverse verification and validation.

1.12.3.9 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.

1.12.3.10 The systems and software architecture design shall have the minimum complexity commensurate with the design requirements.

1.12.3.11 When functional diversity is required, the diverse software based systems shall be associated with the same class, unless appropriate justification is provided (see eg [13]).

1.12.3.12 When functional diversity is required by deterministic arguments of the safety demonstration, the software of the diverse systems shall be subjected to the same graded requirements.

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 given full consideration to the factors that could lead to coincident failures.

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 ensure best use is made of emerging scientifically based advice, 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 coincident failures, eg with variations in names, in memory locations, in execution sequences or by other software diversity methods (see section 1.12.2.3 in this chapter).

* * *

1.13 Software Reliability

1.13.1 Rationale

1.13.1.1 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.

1.13.1.2 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 [18]. 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 [13] and [8] 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 (i.e. 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 modest.

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 [3] and the other Bayesian statistics [19]. 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 [16], [17], [18].

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 (i.e. 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 Positions – 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 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 (i.e. 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 common position 2.3.3.1.9.

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 [8] 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 (i.e. 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 (i.e. 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, i.e. 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

1.14.1.1 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.

1.14.1.2 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.

1.14.1.3 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 contribute to software dependability.

1.14.1.4 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 dependability (reliability, availability) based on statistical data,
- to provide qualitative confidence in the dependability – or 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 of its dependability.

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:

1.14.2.4.1 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.

1.14.2.4.2 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.

1.14.2.4.3 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.

1.14.2.4.4 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:

1.14.3.5.1 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.

1.14.3.5.2 The value of the credit as an acceptance or complementing factor is judged against the criteria stated in IEC 60880 Ed. 2 (2006) [12];

1.14.3.5.3 The operating experience data are demonstrably relevant, i.e. they are shown to satisfy the conditions of relevance of common positions 1.14.3.4, 1.14.3.6 and 1.14.3.7;

1.14.3.5.4 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;

1.14.3.5.5 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:

- meaningful information on the occurrences, the severity of the effects and the actions taken to correct or mitigate all detected faults, errors and failures,
- information on malfunctions detected in other applications,
- 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.

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.

1.14.3.7 When evaluating the relevance of operating experience, the impact of configuration management and maintenance activities shall be taken into account (see in particular paragraph 1.14.2.4.4 above and common position 2.7.3.3 addressing software maintenance, ie software changes after installation).

1.14.3.8 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 1.14.2.4.4 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 [8], 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

1.15.1.1 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.

1.15.1.2 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 RAM 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.

1.15.1.3 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.

1.15.1.4 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.

1.15.1.5 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 take into account 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 (i.e. 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 licensee shall define the safety class of the smart sensor/actuator. The class shall be determined in accordance with chapter 1.2 and will be the same as the class of the system in which the smart sensor/actuator is embedded unless justified otherwise.

1.15.3.5 The smart sensor/actuator dependability (safety, reliability, availability) 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.

Failure Analysis

1.15.3.6 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.

1.15.3.7 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.

Dependability Assessment and Qualification

1.15.3.8 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 1.4.3.9.4). If any of the safety standard clauses are not used then the omissions shall be evaluated and justified.

1.15.3.9 The licensee shall undertake a programme of independent (i.e. from the smart sensor/actuator supplier) confidence building activities appropriate to the safety class (which might include activities such as 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.

1.15.3.10 For safety systems, the smart sensors/actuators software implementation of the dependability requirements mentioned in common position 1.15.3.5 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 common position 1.15.3.11 below.

1.15.3.11 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 common position 1.15.3.8 above) shall be determined and justified by the licensee in relation to the:

- specific nature of the missing evidence;
- dependability properties identified under common position 1.15.3.5; and
- potential failure modes of the smart sensor/actuator identified under common position 1.15.3.6.

1.15.3.12 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 dependability properties. Sufficient evidence shall be available to justify:

- the quality of the production process (quality and verification and validation plans, pedigree and experience of designers and supplier, etc);
- the quality of the product (test and verification coverage, etc);
- 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);
- that specific risks (eg common cause failure, unused existing software, flooding (overloading), etc) are not a concern.

1.15.3.13 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.

1.15.3.14 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

1.15.3.15 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).

1.15.3.16 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).

1.15.3.17 Common positions 1.4.3.5, 1.4.3.7, 1.4.3.9.1, 1.4.3.9.2, 1.4.3.9.3, 1.4.3.9.5, 1.4.3.10, 1.4.3.11 shall apply without change to smart sensors/actuators.

1.15.3.18 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 1.14.3.8.

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 1.15.3.8 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.

* * *

PART 2: LIFE CYCLE 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 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 may have an important impact on all subsequent stages of the design, and can be difficult to correct at later stages.

2.1.2 Issues Involved

The system requirements and their documentation must be 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. At the same time the description must be unambiguous and accurate, and this implies a certain degree of formalism. A compromise which preserves safety must be made between these two requirements which are in conflict. [20]

2.1.3 Common Position

2.1.3.1 The system requirements shall be validated, i.e. their correctness, consistency, completeness 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 documentation shall at least cover the following items:

2.1.3.2.1 A precise definition of the system boundaries, that is of the system interface to the plant and of its man machine interface. 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.

2.1.3.2.2 A specification of system requirements, i.e. of the functions and of the non-functional properties – dependability, security, performance (eg accuracy, timing constraints, cycle and required response times in relation to the rates of change of the plant parameters) – required from the system.

2.1.3.2.3 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.

2.1.3.2.4 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.

2.1.3.2.5 The specification of special engineered features where there is a need to change specific system parameters such as calibration constants for operational reasons.

2.1.3.2.6 The results of the validation required in 2.1.3.1.

2.1.3.2.7 All modes of operation (eg power operations, refuelling, post accident monitoring) and human interactions (eg operation, maintenance, tests).

2.1.3.3 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.

2.1.3.4 The system should include the means for verifying that the final control elements have achieved the desired state in the required time. Correct completion of a safety function shall be monitored and any failure shall be notified to the plant operator.

2.1.4 Recommended Practices

2.1.4.1 When practical, the system requirements specifications should be animated or simulated.

2.1.4.2 The specifications of the system validation tests should be derived from the system requirements.

* * *

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. New designs are often driven by technological considerations (use of a new, more efficient or more reliable technology) or by commercial objectives (better configurability, easier maintenance).

For an adequate safety demonstration, the computer system architecture that is proposed should be traceably derived from the system requirements with documented arguments 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, man-machine interfaces, etc. These components do not contribute in the same way to the dependability 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.

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 ASICs (application specific integrated circuits),
- 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 (EQ); electromagnetic interference (EMI) and radio-frequency interference (RFI) immunity,
- Communication systems, multiplexers, network protocols,
- Fault detection and auto-supervision,
- HW/SW fault tolerance mechanisms,
- Testability, In service testing and maintenance, auto-testing
- Systems outside the computer based system boundaries which require design decisions (eg instrumentation requiring software compensation or filtering),
- Human interfaces,
- Security,
- Constraints on downstream 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 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 auto testing or by periodic testing, and are notified. 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:

- the possible failure modes of the architecture that may compromise the safety functions have been taken into account, and
- 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 so that verification and validation are made as easy as possible.

2.2.3.12 For safety systems, the common position 2.7.3.3.2 is applicable.

2.2.3.13 The evidence of satisfying the requirements of the above common positions shall be documented and made available to the licensor as early as possible in the project.

2.2.3.14 It shall be confirmed that the system architecture ensures that

- all system states are analysed to identify those that are hazardous,
- the safe state space of the system is defined,
- the system components are constrained to those whose safe state space is observable, and departure from it detectable.

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:

2.2.4.1.1 Fault tolerance is the capability to mitigate the effect of errors, so as to prevent them from turning into failures that jeopardise system safety. It requires detection, control and decision mechanisms, and interconnections. The efficiency of fault tolerant mechanisms depends on the knowledge of the faults to be protected against. Also, it works best when reliable components of design are employed, when the error rate is already low, and the fault mechanisms well identified. In addition it must be recognised that a fault tolerant mechanism can become a source of failures in its own right. These potential failures should be identified, and their impact on the fault tolerance capabilities should be assessed.

2.2.4.1.2 Fault avoidance through the use of highly reliable design elements is therefore 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.

2.2.4.2 The use of features not specified by the manufacturer, for example undeclared instructions or clock rates, should be prohibited unless their use is fully justified.

* * *

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 (i.e. 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 reference [9]):

- 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, or 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 demonstration of tool dependability (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, the architecture does not localise their impact and 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

2.3.3.1 Software Functional and Non-Functional Requirements

2.3.3.1.1 The software requirements shall be demonstrated to satisfy the system and architecture requirements, both functional and non-functional. In particular, these requirements include reliability, availability and security requirements. 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.

2.3.3.1.2 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.

2.3.3.1.3 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 plan, at the start of the project.

2.3.3.1.4 The software requirements shall address the results of the computer system failure and hazard analyses (see common position 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 common position 2.8.3.1.2, the adequacy of automatic self supervision in combination with periodic testing during operation shall be demonstrated.

2.3.3.1.5 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 error shall be notified (see also 2.1.3.2.3, 2.1.3.2.4, 2.2.3.7, 2.2.3.8 and 2.2.3.9).

2.3.3.1.6 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.

2.3.3.1.7 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.

2.3.3.1.8 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.

2.3.3.1.9 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.

2.3.3.2 Software Architecture and Design

2.3.3.2.1 The software design shall satisfy the design requirements.

2.3.3.2.2 The software 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.

2.3.3.2.3 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.

2.3.3.2.4 Documented evidence shall be given that the supervision features mentioned in common position 2.3.3.1.4 maintain the system in a safe state.

2.3.3.2.5 The design shall constrain the implementation so that the execution of the software is verifiable and predictable.

2.3.3.2.6 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.

2.3.3.2.7 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 that were identified for the software it makes use of: the operating system services and the device drivers.

2.3.3.2.8 The software shall be designed so as to facilitate static analysis and testing as much as possible, for instance by minimising state memorisation between system execution cycles and by maximising independence between software input-output paths (see common positions 2.4.3.3.1 and 2.4.3.3.6 on coding).

2.3.3.2.9 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

2.3.4.1 Functional and non-Functional Software Requirements

2.3.4.1.1 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 [10]).

2.3.4.1.2 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.

2.3.4.1.3 For each relevant software output, a safe state should be indicated that can be used in case of a detectable but not correctable error.

2.3.4.1.4 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.

2.3.4.2 Software Design

2.3.4.2.1 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.

2.3.4.2.2 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.

2.3.4.2.3 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 recommendations 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).

2.4.3.3 Coding

2.4.3.3.1 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.

2.4.3.3.2 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.

2.4.3.3.3 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.

2.4.3.3.4 The code shall be written in a style and with language features and program constructs that make it as easy as possible to test, in particular by reducing the number of dynamic test cases required.

2.4.3.3.5 The code shall be designed so as to facilitate static analysis, testing and readability.

2.4.3.3.6 In particular, the code shall – as much as possible – run in a direct and fixed sequence pattern, i.e.:

- Computed branching shall be prohibited. Branching into loops, modules or subroutines shall be prohibited.
- Dynamic instruction changes shall be prohibited.
- 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.
- Recursion – not amenable to static analysis – and re-entrancy shall be avoided unless they are shown to produce simpler code than by other means.

– 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.
– All alternatives in switch or case statements shall be explicitly covered by the code. Appropriate actions shall be implemented to deal with default conditions.
– Indirect addressing shall be used carefully so that computed addresses shall be easy to identify.

2.4.3.3.7 Dynamic storage allocation shall be prohibited.

2.4.3.3.8 Module size shall not exceed a limit specified for the system without justification.

2.4.3.3.9 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.

2.4.3.3.10 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.

2.4.3.4 Subroutines

2.4.3.4.1 For safety system software, a maximum shall be specified for the depth of nested subroutines (eg to avoid stack overflows).

2.4.3.4.2 To reduce complexity, avoid potential coding errors, and facilitate static analysis, failure mode analysis and testing, subroutines shall

– communicate with their environment exclusively via their parameters;
– for safety system software, have only one entry point and return from only one point.

2.4.3.5 Data structures and addressing

2.4.3.5.1 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.

2.4.3.5.2 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.

2.4.3.5.3 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.

2.4.3.5.4 In agreement with 2.4.3.3, 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.

2.4.3.5.5 Arithmetic expressions shall reflect the equations they represent.

2.4.3.5.6 Arithmetic expressions shall be executed in binary fixed-point arithmetic, unless the use of floating-point arithmetic can be demonstrated to contribute to safety.

2.4.3.5.7 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.

2.4.3.5.8 If execution in floating-point arithmetic is necessary, the hardware and software used to implement floating point calculations and functions shall be suitably qualified.

2.4.3.6 Defensive Programming

2.4.3.6.1 If complicated calculations are necessary, the program shall be written so that – where feasible – a simple function or assertion will provide an error check and back-up action.

2.4.3.6.2 In addition to requirements 2.2.3.8 and 2.3.3.1.5, defensive coding shall be applied by programmers with the objective of maintaining program behaviour within its specifications, including the following:

- known states and program properties, and parameter ranges shall be verified;
- variable values that can cause faulty operations shall be detected, (for example, division by 0).

2.4.3.6.3 Dynamic checking for overflow and underflow shall be performed, eg for array bounds, lists, queues, buffers.

2.4.3.7 Language and compiler requirements

2.4.3.7.1 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.

2.4.3.7.2 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.

2.4.3.7.3 The compiler of the programming language shall satisfy the applicable common positions of section 1.5.3. In particular, the error detection capabilities of the compiler shall be clearly defined and verified to the extent that the safety demonstration depends on these capabilities.

2.4.3.7.4 The same compiler version shall be used for all compiled parts of the software running on the same processor.

2.4.3.7.5 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.

2.4.3.7.6 Any compiler code optimisation shall be provably correct. The same optimisation options shall be used on all the software compiled by the same compiler.

2.4.3.8 Operating Systems

2.4.3.8.1 In support of common position 2.3.3.2.5, the behaviour of the operating system shall be predictable, analysable and verifiable, for example by ensuring that:

- the organisation of the operating system into tasks, including the allocation of services to tasks, is as simple as possible, clearly identified and documented;
- the task generation should take place statically in a linear sequence in the initialising phase (no dynamic task creation and deletion);
- the task priorities are statically assigned;
- no dynamic memory allocation is used;
- the system includes all the self diagnosis features necessary to detect its own failures;
- use of interrupts is restricted to cases that result in design simplification;
- the absence of stack overflow is demonstrated.

2.4.3.9 Support Software and Tools

2.4.3.9.1 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.

2.4.3.10 Documentation

2.4.3.10.1 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.

2.4.3.10.2 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.

2.4.3.10.3 Commentaries shall also include any specific information required for maintenance of the code, in particular, its history and the designers' names.

2.4.3.10.4 If a tool has generated the code, the common positions in this section 2.4.3.10 apply to the tool inputs.

2.4.4 Recommended Practices

2.4.4.1 Coding

2.4.4.1.1 If a requirement specification language is used, it should have a well defined syntax and semantics.

2.4.4.1.2 The coding phase should start only after the verification team has approved the design phase.

2.4.4.1.3 Evidence should be provided that structured programming techniques have been followed.

2.4.4.1.4 Branching out of loops should be avoided, except in the case of error exit.

2.4.4.2 Data structures and addressing

2.4.4.2.1 The use of global variables within subroutines should be avoided.

2.4.4.2.2 The symbolic names and types of variables, constants, modules, procedures and parameters should be meaningful at the application level.

2.4.4.3 Language and compiler requirements

2.4.4.3.1 For application software, it is strongly recommended that a problem-oriented language rather than machine-oriented language be used.

2.4.4.3.2 Assembler and machine code insertions into high level language code should not be allowed without justification.

2.4.4.3.3 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.

2.4.4.3.4 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.

* * *

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 life cycle process.

Most of the commonly used software development life-cycle 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 [12] lists the most common software testing methods. (see eg also [21]).

2.5.2 Issues Involved

2.5.2.1 Demonstration of software correctness and of its contribution to reliability:

2.5.2.1.1 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, man-machine interfaces, etc. These components do not contribute in the same way to the dependability of the whole system. Specific evidence and methods of validation and verification are necessary to show that these components comply with the dependability requirements of the system.

2.5.2.1.2 Software behaviour can be highly sensitive to "small" programming, coding or even typing errors. 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.

2.5.2.1.3 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.

2.5.2.1.4 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.

2.5.2.1.5 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

2.5.3.1 Selection of Verification Methods and Tools

2.5.3.1.1 At the end of each phase of the life cycle (requirements, design, coding, hardware-software integration), the output of the phase shall be verified against its input specifications.

2.5.3.1.2 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.

2.5.3.1.3 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.

2.5.3.1.4 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. This justification will be accessible to the regulator.

2.5.3.1.5 Every software module shall be verified by executing the code to a degree of coverage that complies with the common positions in section 2.5.3.3.

2.5.3.1.6 The verification of the system integration shall include testing by executing the code.

2.5.3.2 Verification Planning

2.5.3.2.1 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.

2.5.3.2.2 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).

2.5.3.2.3 The verification plan shall address the criteria, strategy and equipment that are necessary for each verification step. Furthermore the verification plan shall clearly define the intended test coverage.

2.5.3.3 Coverage

2.5.3.3.1 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.

2.5.3.3.2 All modes of system operations shall be considered during the design of test cases.

2.5.3.3.3 Each distinct test at system or module level shall monitor all outputs of the system or module.

2.5.3.3.4 All interfaces (between modules, and between programs), and all communication protocols shall be tested.

2.5.3.3.5 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.

2.5.3.3.6 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.

2.5.3.3.7 Data structures and declared constant values shall be verified for correctness.

2.5.3.3.8 Operations on – and access to – all data items shall be exercised, and the coverage of these tests shall be justified.

2.5.3.3.9 All time critical sections of code shall be tested under realistic conditions, and the coverage of these tests shall be justified.

2.5.3.3.10 The result of calculations performed by software shall be verified against pre-calculated values.

2.5.3.3.11 The adequacy of commentaries and code documentation shall be verified.

2.5.3.3.12 Compliance with the appropriate project design and coding directives and standards shall be verified.

2.5.3.4 Traceability and Documentation

2.5.3.4.1 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.

2.5.3.4.2 Test results shall clearly indicate which tests met expectations and which did not.

2.5.3.4.3 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.

2.5.3.4.4 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.

2.5.3.5 Independent Verification

2.5.3.5.1 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.

2.5.3.5.2 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.

2.5.3.5.3 All verification activities shall be comprehensively documented to enable auditing.

2.5.3.5.4 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 life-cycle 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 the 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 50-SG-D4 [7], "Commissioning Procedures 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 dependability of the whole system. Specific evidence and methods of validation and verification are necessary to show that these components comply with the dependability requirements of the system.

2.6.2.3 During the production of the software, 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 problematical.

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

2.6.3.1 Validation

2.6.3.1.1 The computer system validation shall be conducted in accordance with a formal validation plan. The plan shall identify acceptance criteria, techniques, tools, test cases and expected results.

2.6.3.1.2 Validation tests shall be performed on target hardware and under representative environmental conditions.

2.6.3.1.3 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.

2.6.3.1.4 Testing shall be conducted in a disciplined and orderly manner and in accordance with IEC 60880 [12]. 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.

2.6.3.1.5 Timing conflicts and bus contention problems are of importance for validation because they cannot be tested comprehensively in earlier phases. These aspects shall be thoroughly tested. Adequacy of spare CPU time, scan times and data transfer rates shall be demonstrated.

2.6.3.1.6 It shall be ensured that the requirements specifying the fault, exception and failure mitigating mechanisms (see common positions 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 common position 2.1.3.2.4), are correctly implemented.

2.6.3.1.7 The system shall also be subjected to appropriate validation tests to check that the set of system requirements is complete, consistent and correct.

2.6.3.1.8 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.

2.6.3.1.9 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.

2.6.3.1.10 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.

2.6.3.1.11 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.

2.6.3.1.12 All changes to either software or test specifications shall be subjected to change control procedures that comply with the recommendations of chapter 2.7.

2.6.3.1.13 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.

2.6.3.1.14 The validity of simulated inputs shall be justified and tests shall demonstrate the correct operation of simulated items.

2.6.3.1.15 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 dependability.

2.6.3.1.16 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.

2.6.3.1.17 All validation activities shall be comprehensively documented to enable auditing. The validation team shall indicate anomalies only; they shall not recommend design changes.

2.6.3.1.18 The system operation and maintenance manuals shall be validated, as far as possible, during the validation phase.

2.6.3.2 Commissioning

2.6.3.2.1 Commissioning tests shall ensure that both the system and the plant safety requirements are complete and correct.

2.6.3.2.2 The number and type of commissioning tests shall be justified. The omission of any validation test from the commissioning tests shall be justified.

2.6.3.2.3 The validation of the system operational and maintenance manuals shall, if necessary, be completed during the commissioning phase.

2.6.3.2.4 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

2.7.1.1 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 an error 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.

2.7.1.2 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 [12] and IEC 61513 [15] 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 common position 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 errors 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 errors 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

2.7.3.1 General

2.7.3.1.1 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 errors.

2.7.3.1.2 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 section 2.7.3.4) 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.

2.7.3.1.3 Once a software or documentation item has been approved, i.e. 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 the following paragraphs 2.7.3.1.4, 2.7.3.1.5 and 2.7.3.1.6.

2.7.3.1.4 The software change procedure and the configuration management system shall include an adequate problem reporting and tracking system.

2.7.3.1.5 The software change procedure shall contain the following basic elements:

- the identification and documentation of the need for the change;
- 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;
- 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 effect on non-functional system properties as, eg dependability, response time, system accuracy, hardware performance. 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.
- 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.
- The requirements of IEC 60880 [11] sections 9.1, 9.2 and 9.3 shall apply.

2.7.3.1.6 Faults shall be analysed for cause and lack of earlier detection. Any generic concern shall be rectified and a report produced.

2.7.3.2 Software Modification

2.7.3.2.1 A correction to a wrongly implemented software change, if found at the stage of site testing (i.e. following installation of the software on the plant), shall be processed as though it were a new software change proposal.

2.7.3.2.2 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.

2.7.3.3 Software Maintenance

2.7.3.3.1 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 this section 2.7.3.3.

2.7.3.3.2 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 error.

2.7.3.3.3 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.

2.7.3.3.4 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.

2.7.3.3.5 Only software approved for release shall be installed at site.

2.7.3.3.6 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.

2.7.3.3.7 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.

2.7.3.3.8 The independent reviews shall be documented.

2.7.3.4 Configuration Management

2.7.3.4.1 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.

2.7.3.4.2 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.

2.7.3.4.3 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.

2.7.3.4.4 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.

2.7.3.4.5 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 1.3) 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 common position 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, errors in these data could adversely affect the integrity of the safety system. For instance, an error 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 errors 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 errors, misreading and incorrect addressing). Equally, when data is entered electronically, there is the possibility of data corruption eg due to programming errors 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 error 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 whilst 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

2.8.3.1 Periodic Testing

2.8.3.1.1 A program 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.

2.8.3.1.2 It shall be demonstrated and documented that the combination of the provisions of automatic self-supervision and periodic testing during operations (cf. common positions 2.2.3.7 and 2.2.3.10) cover all postulated faults.

2.8.3.1.3 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 (cf. also common position 2.1.3.4).

2.8.3.2 Generation of Calibration Data

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.

2.8.3.3 Loading of Calibration Data

2.8.3.3.1 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.

2.8.3.3.2 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.

2.8.3.3.3 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.

2.8.3.3.4 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.

2.8.3.3.5 A full test of the guardline shall be performed following a data change so as to demonstrate its correct operation.

2.8.3.4 Equipment for Periodic Testing

2.8.3.4.1 The test equipment software shall be qualified to a level in accordance with its intended use.

2.8.3.4.2 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.

2.8.3.4.3 On termination of the testing operations, the test equipment shall restore the safety system to its original status.

2.8.3.4.4 The test equipment shall be subjected to periodic calibration checks.

2.8.3.5 Equipment for Plant Component Testing and Corrective Maintenance

2.8.3.5.1 The means of testing plant items shall be engineered as part of the safety system. Such 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.

2.8.3.5.2 It shall not be possible to connect this type of test equipment to more than one guardline at any one time.

2.8.3.5.3 Bypass of actuation signals for maintenance of output devices and components shall be indicated by alarm and be recorded as unavailable.

2.8.3.6 Operator Training

2.8.3.6.1 Instrumentation and control specialists shall follow a training program that addresses safety system performances, operation and maintenance during normal and abnormal reactor operation.

2.8.3.6.2 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.

2.8.3.7 Operational Experience

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 common positions 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 common positions of sections 2.7.3.1, 2.7.3.2, 2.7.3.3 are applicable.

2.8.3.8 Requirements to support inspection

2.8.3.8.1 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 program. This program 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.

2.8.3.8.2 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.

Inspection topic	**Common position**
Safety 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 dependability	1.5.3.2
Operating experience	1.14.3.1 to 1.14.3.8
Software change procedure	2.7.3.1.5
Version control procedure	2.7.3.4.2
Periodic testing	2.8.3.1.1

2.8.4 Recommended Practices

2.8.4.1 Periodic Testing

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 common position 1.5.3.2).

2.8.4.2 Generation of Calibration Data

2.8.4.2.1 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.

2.8.4.2.2 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.

2.8.4.3 Loading of Calibration Data

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 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 1 (draft regulatory guide DG-1130, December 2004) of proposed revision 2 of US NRC Regulatory Guide 1.152: “Criteria for programmable digital computer system software in safety related systems of NPPs”

2 Courtois, P.J. 2008. Justifying the dependability of computer-based system. With applications in nuclear engineering. Springer series in reliability engineering. 323pp.

3 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

4 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.

5 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.

6 HSE 1997. Four Party Regulatory Consensus Report on The Safety case for Computer-based Systems in Nuclear Power Plants. November 1997.

7 IAEA Safety Guide, Commissioning for Nuclear Plants, Safety Standards Series N° NS-G-2.9, 2003.

8 IAEA Safety Guide, Software for Computer Based Systems Important to Safety in Nuclear Power Plants, Safety Standards Series N° NS-G-1.1, 2000.

9 IAEA, Technical Reports Series. Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control, TRS N°384, 1999.

10 IAEA Technical Reports Series. Software Important to Safety in Nuclear Power Plants, TRS N°367, 1994.

11 IEC 60880. 1986. Software for computers in the safety systems of nuclear power stations.

12 IEC 60880. 2nd edition, 2006. Nuclear power plants. Instrumentation and control systems important to safety. Software aspects for computer-based systems performing category A functions.

13 IEC 61226. Second edition. 2005. Nuclear power plants – Instrumentation and control systems important to safety – Classification of instrumentation and control functions.

14 IEC 61508. 1998. Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General Requirements, Part 3: Software Requirements.

15 IEC 61513. 2001. Nuclear power plants. Instrumentation and control for systems important to safety. General Requirements for systems.

16 Knight and Leveson 1986 – An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming, IEEE transactions on Software Engineering, SE-12 (1), pp.96-109, 1986.

17 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).

18 Littlewood, B., Strigini, L. Validation of Ultra-High Dependability for Software-based Systems, City University. In Communications of the ACM, November 1993, pp. 69-80.

19 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.

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 Rushby, J. 1993. *Formal Methods and the Certification of Critical Systems*, Technical Report CSL-93-7, SRI International.

22 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.

* * *

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, PO Box, 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.

Perry, W. 1995. Effective methods for Software Testing. John Wiley.

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)