IAEA SAFETY STANDARDS SERIES

Software for Computer Based Systems Important to Safety in Nuclear Power Plants

SAFETY GUIDE

No. NS-G-1.1

INTERNATIONAL
ATOMIC ENERGY AGENCY
VIENNA

IAEA SAFETY RELATED PUBLICATIONS

IAEA SAFETY STANDARDS

Under the terms of Article III of its Statute, the IAEA is authorized to establish standards of safety for protection against ionizing radiation and to provide for the application of these standards to peaceful nuclear activities.

The regulatory related publications by means of which the IAEA establishes safety standards and measures are issued in the **IAEA Safety Standards Series**. This series covers nuclear safety, radiation safety, transport safety and waste safety, and also general safety (that is, of relevance in two or more of the four areas), and the categories within it are **Safety Fundamentals**, **Safety Requirements** and **Safety Guides.**

Safety Fundamentals (blue lettering) present basic objectives, concepts and principles of safety and protection in the development and application of nuclear energy for peaceful purposes.

Safety Requirements (red lettering) establish the requirements that must be met to ensure safety. These requirements, which are expressed as 'shall' statements, are governed by the objectives and principles presented in the Safety Fundamentals.

Safety Guides (green lettering) recommend actions, conditions or procedures for meeting safety requirements. Recommendations in Safety Guides are expressed as 'should' statements, with the implication that it is necessary to take the measures recommended or equivalent alternative measures to comply with the requirements.

The IAEA's safety standards are not legally binding on Member States but may be adopted by them, at their own discretion, for use in national regulations in respect of their own activities. The standards are binding on the IAEA in relation to its own operations and on States in relation to operations assisted by the IAEA.

Information on the IAEA's safety standards programme (including editions in languages other than English) is available at the IAEA Internet site

www.iaea.org/ns/coordinet

or on request to the Safety Co-ordination Section, IAEA, P.O. Box 100, A-1400 Vienna, Austria.

OTHER SAFETY RELATED PUBLICATIONS

Under the terms of Articles III and VIII.C of its Statute, the IAEA makes available and fosters the exchange of information relating to peaceful nuclear activities and serves as an intermediary among its Member States for this purpose.

Reports on safety and protection in nuclear activities are issued in other series, in particular the **IAEA Safety Reports Series**, as informational publications. Safety Reports may describe good practices and give practical examples and detailed methods that can be used to meet safety requirements. They do not establish requirements or make recommendations.

Other IAEA series that include safety related sales publications are the **Technical Reports Series,** the **Radiological Assessment Reports Series** and the **INSAG Series**. The IAEA also issues reports on radiological accidents and other special sales publications. Unpriced safety related publications are issued in the **TECDOC Series**, the **Provisional Safety Standards Series**, the **Training Course Series,** the **IAEA Services Series** and the **Computer Manual Series**, and as **Practical Radiation Safety Manuals** and **Practical Radiation Technical Manuals**.

SOFTWARE FOR COMPUTER BASED SYSTEMS IMPORTANT TO SAFETY IN NUCLEAR POWER PLANTS

The following States are Members of the International Atomic Energy Agency:

AFGHANISTAN
ALBANIA
ALGERIA
ANGOLA
ARGENTINA
ARMENIA
AUSTRALIA
AUSTRIA
BANGLADESH
BELARUS
BELGIUM
BENIN
BOLIVIA
BOSNIA AND HERZEGOVINA
BRAZIL
BULGARIA
BURKINA FASO
CAMBODIA
CAMEROON
CANADA
CHILE
CHINA
COLOMBIA
COSTA RICA
COTE D'IVOIRE
CROATIA
CUBA
CYPRUS
CZECH REPUBLIC
DEMOCRATIC REPUBLIC OF THE CONGO
DENMARK
DOMINICAN REPUBLIC
ECUADOR
EGYPT
EL SALVADOR
ESTONIA
ETHIOPIA
FINLAND
FRANCE
GABON
GEORGIA
GERMANY
GHANA
GREECE
GUATEMALA
HAITI
HOLY SEE
HUNGARY
ICELAND
INDIA
INDONESIA
IRAN, ISLAMIC REPUBLIC OF
IRAQ
IRELAND
ISRAEL
ITALY
JAMAICA
JAPAN
JORDAN
KAZAKHSTAN
KENYA
KOREA, REPUBLIC OF
KUWAIT
LATVIA
LEBANON
LIBERIA
LIBYAN ARAB JAMAHIRIYA
LIECHTENSTEIN
LITHUANIA
LUXEMBOURG
MADAGASCAR
MALAYSIA
MALI
MALTA
MARSHALL ISLANDS
MAURITIUS
MEXICO
MONACO
MONGOLIA
MOROCCO
MYANMAR
NAMIBIA
NETHERLANDS
NEW ZEALAND
NICARAGUA
NIGER
NIGERIA
NORWAY
PAKISTAN
PANAMA
PARAGUAY
PERU
PHILIPPINES
POLAND
PORTUGAL
QATAR
REPUBLIC OF MOLDOVA
ROMANIA
RUSSIAN FEDERATION
SAUDI ARABIA
SENEGAL
SIERRA LEONE
SINGAPORE
SLOVAKIA
SLOVENIA
SOUTH AFRICA
SPAIN
SRI LANKA
SUDAN
SWEDEN
SWITZERLAND
SYRIAN ARAB REPUBLIC
THAILAND
THE FORMER YUGOSLAV REPUBLIC OF MACEDONIA
TUNISIA
TURKEY
UGANDA
UKRAINE
UNITED ARAB EMIRATES
UNITED KINGDOM OF GREAT BRITAIN AND NORTHERN IRELAND
UNITED REPUBLIC OF TANZANIA
UNITED STATES OF AMERICA
URUGUAY
UZBEKISTAN
VENEZUELA
VIET NAM
YEMEN
YUGOSLAVIA
ZAMBIA
ZIMBABWE

The Agency's Statute was approved on 23 October 1956 by the Conference on the Statute of the IAEA held at United Nations Headquarters, New York; it entered into force on 29 July 1957. The Headquarters of the Agency are situated in Vienna. Its principal objective is "to accelerate and enlarge the contribution of atomic energy to peace, health and prosperity throughout the world''.

Printed by the IAEA in Austria
September 2000
STI/PUB/1095

SAFETY STANDARDS SERIES No. NS-G-1.1

SOFTWARE FOR COMPUTER BASED SYSTEMS IMPORTANT TO SAFETY IN NUCLEAR POWER PLANTS

SAFETY GUIDE

INTERNATIONAL ATOMIC ENERGY AGENCY
VIENNA, 2000

VIC Library Cataloguing in Publication Data

Software for computer based systems important to safety in nuclear power plants : safety guide. — Vienna : International Atomic Energy Agency, 2000.

p. ; 24 cm. — (Safety standards series, ISSN 1020–525X ; no. NS-G-1.1)

STI/PUB/1095

ISBN 92–0–101800–2

Includes bibliographical references.

1. Nuclear power plants—Safety measures—Data processing. I. International Atomic Energy Agency. II. Series.

VICL 00–00250

FOREWORD

by Mohamed ElBaradei
Director General

One of the statutory functions of the IAEA is to establish or adopt standards of safety for the protection of health, life and property in the development and application of nuclear energy for peaceful purposes, and to provide for the application of these standards to its own operations as well as to assisted operations and, at the request of the parties, to operations under any bilateral or multilateral arrangement, or, at the request of a State, to any of that State's activities in the field of nuclear energy.

The following advisory bodies oversee the development of safety standards: the Advisory Commission for Safety Standards (ACSS); the Nuclear Safety Standards Advisory Committee (NUSSAC); the Radiation Safety Standards Advisory Committee (RASSAC); the Transport Safety Standards Advisory Committee (TRANSSAC); and the Waste Safety Standards Advisory Committee (WASSAC). Member States are widely represented on these committees.

In order to ensure the broadest international consensus, safety standards are also submitted to all Member States for comment before approval by the IAEA Board of Governors (for Safety Fundamentals and Safety Requirements) or, on behalf of the Director General, by the Publications Committee (for Safety Guides).

The IAEA's safety standards are not legally binding on Member States but may be adopted by them, at their own discretion, for use in national regulations in respect of their own activities. The standards are binding on the IAEA in relation to its own operations and on States in relation to operations assisted by the IAEA. Any State wishing to enter into an agreement with the IAEA for its assistance in connection with the siting, design, construction, commissioning, operation or decommissioning of a nuclear facility or any other activities will be required to follow those parts of the safety standards that pertain to the activities to be covered by the agreement. However, it should be recalled that the final decisions and legal responsibilities in any licensing procedures rest with the States.

Although the safety standards establish an essential basis for safety, the incorporation of more detailed requirements, in accordance with national practice, may also be necessary. Moreover, there will generally be special aspects that need to be assessed by experts on a case by case basis.

The physical protection of fissile and radioactive materials and of nuclear power plants as a whole is mentioned where appropriate but is not treated in detail; obligations of States in this respect should be addressed on the basis of the relevant instruments and publications developed under the auspices of the IAEA.

Non-radiological aspects of industrial safety and environmental protection are also not explicitly considered; it is recognized that States should fulfil their international undertakings and obligations in relation to these.

The requirements and recommendations set forth in the IAEA safety standards might not be fully satisfied by some facilities built to earlier standards. Decisions on the way in which the safety standards are applied to such facilities will be taken by individual States.

The attention of States is drawn to the fact that the safety standards of the IAEA, while not legally binding, are developed with the aim of ensuring that the peaceful uses of nuclear energy and of radioactive materials are undertaken in a manner that enables States to meet their obligations under generally accepted principles of international law and rules such as those relating to environmental protection. According to one such general principle, the territory of a State must not be used in such a way as to cause damage in another State. States thus have an obligation of diligence and standard of care.

Civil nuclear activities conducted within the jurisdiction of States are, as any other activities, subject to obligations to which States may subscribe under international conventions, in addition to generally accepted principles of international law. States are expected to adopt within their national legal systems such legislation (including regulations) and other standards and measures as may be necessary to fulfil all of their international obligations effectively.

EDITORIAL NOTE

An appendix, when included, is considered to form an integral part of the standard and to have the same status as the main text. Annexes, footnotes and bibliographies, if included, are used to provide additional information or practical examples that might be helpful to the user.

The safety standards use the form 'shall' in making statements about requirements, responsibilities and obligations. Use of the form 'should' denotes recommendations of a desired option.

The English version of the text is the authoritative version.

CONTENTS

1. INTRODUCTION

BACKGROUND

1.1. Computer based systems are of increasing importance to safety in nuclear power plants as their use in both new and older plants is rapidly increasing. They are used both in safety related applications, such as some functions of the process control and monitoring systems, as well as in safety critical applications, such as reactor protection or actuation of safety features. The dependability of computer based systems important to safety is therefore of prime interest and should be ensured.

1.2. With current technology, it is possible in principle to develop computer based instrumentation and control systems for systems important to safety that have the potential for improving the level of safety and reliability with sufficient dependability. However, their dependability can be predicted and demonstrated only if a systematic, fully documented and reviewable engineering process is followed. Although a number of national and international standards dealing with quality assurance for computer based systems important to safety have been or are being prepared, internationally agreed criteria for demonstrating the safety of such systems are not generally available. It is recognized that there may be other ways of providing the necessary safety demonstration than those recommended here.

1.3. The basic requirements for the design of safety systems for nuclear power plants are provided in the Requirements for Design issued in the IAEA Safety Standards Series [1]. These requirements were extended and interpreted for the design of the protection system and related features in Ref. [2] and for the design of safety related instrumentation and control systems in Ref. [3]. The revision of Refs [2, 3] is under way to reflect the present state of technology, including the application of digital technology.

1.4. The IAEA has issued a Technical Report [4] to assist Member States in ensuring that computer based systems important to safety in nuclear power plants are safe and properly licensed. The report provides information on current software engineering practices and, together with relevant standards (such as Ref. [5]), forms a technical basis for this Safety Guide.

OBJECTIVE

1.5. The objective of this Safety Guide is to provide guidance on the collection of evidence and preparation of documentation to be used in the safety demonstration for the software for computer based systems important to safety in nuclear power plants, for all phases of the system life cycle.

SCOPE

1.6. The guidance is applicable to systems important to safety as defined in Refs [2, 3]. Since at present the reliability of a computer based system cannot be predicted on the sole basis of, or built in by, the design process, it is difficult to define and to agree systematically on any possible relaxation in the guidance to apply to software for safety related systems. Whenever possible, recommendations which apply only to safety systems and not to safety related systems are explicitly identified.

1.7. The guidance relates primarily to the software used in computer based systems important to safety. Guidance on the other aspects of computer based systems, such as those concerned with the design of the computer based system itself and its hardware, is limited to the issues raised by the development, verification and validation of software.

1.8. The main focus of this Safety Guide is on the preparation of documentation that is used for an adequate demonstration of the safety and reliability of computer based systems important to safety.

1.9. This Safety Guide applies to all types of software: pre-existing software or firmware (such as an operating system), software to be specifically developed for the project, or software to be developed from an existing predeveloped equipment family of hardware or software modules. The issue of the use for safety functions of pre-existing or commercial off-the-shelf software, on the development of which little information is available, is addressed in the Annex, where a section from Ref. [6] is reproduced (see also Section 6.3 of Ref. [7]). Information on specifying requirements in the course of upgrades of instrumentation and control systems can be found in Ref. [8].

1.10. This Safety Guide is intended for use by those involved in the production, assessment and licensing of computer based systems, including plant system designers, software designers and programmers, verifiers, validators, certifiers and

regulators, as well as plant operators. The various interfaces between those involved are considered.

STRUCTURE

1.11. Section 2 provides recommendations on the technical considerations for computer based systems, addressing advantages and disadvantages of such systems, safety and reliability issues, and some organizational preconditions of the system development project.

1.12. Section 3 provides recommendations on the application of requirements for management of safety to computer based systems important to safety.

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

1.14. Sections 5 to 15 are dedicated to individual phases of the development life cycle. The sections begin with a short introduction that describes the phase. Under the heading RECOMMENDATIONS, there is a set of recommendations for this phase. Under the heading DOCUMENTS, there is a list of documents to be produced as an output from the phase, and guidance is provided concerning the content of these documents. Also, some general recommendations are made concerning the attributes and presentation of the products of the phase. In all parts, the intent is not to provide an exhaustive description of all material that will be needed for development purposes, but rather to summarize the recommendations and the material and its attributes that are most important for the safety demonstration.

2. TECHNICAL CONSIDERATIONS FOR COMPUTER BASED SYSTEMS

CHARACTERISTICS OF COMPUTER BASED SYSTEMS

2.1. In relation to the assessment of safety and reliability, computer based systems have two basic properties. They are programmable and their hardware is based on

discrete digital logic. As with other systems, hardware faults may be due to errors in design or manufacturing, but typically they result from wearing out, degradation or environmental processes, and are of a random nature. Software, the programmable part of the computer, does not wear out, but it may be affected by changes in the operating environment. Software faults may result from either bad or unclear specification of requirements (which gives rise to errors in the logical design or implementation) or errors introduced during the implementation phase or maintenance phase.

2.2. The programmable nature of computer based systems coupled with the discrete logic means that they have a number of advantages over non-digital and non-programmable systems. They facilitate the achievement of complex functions; in particular, they can provide improved monitoring of plant variables, improved operator interfaces, improved testing, calibration, self-checking and fault diagnosis facilities. They can also provide greater accuracy and stability. The use of multiplexed 'bus' structures may lead to a reduced need for cabling. Software modifications require less physical disruption of equipment, which can be useful for maintenance.

2.3. These advantages are counterbalanced by a number of disadvantages. Software implementation tends to be more complex and therefore more prone to design errors than implementation of purely hard-wired systems. Moreover, software implementations are discrete logic models of the real world. This has two types of consequences. Software is more sensitive (i.e. less tolerant) to 'small' errors. It is also more difficult to test, because interpolation and extrapolation are much more difficult to apply to computer based systems than to traditional analog systems, and ultimately are not entirely valid.

DEVELOPMENT PROCESS

2.4. The development of systems important to safety should be a process controlled step by step. In this approach the development process is organized as an ordered collection of distinct phases. Each phase uses information developed in earlier phases and provides input information for later phases. The development of systems important to safety is by nature an iterative process. As the design progresses, faults and omissions made in the earlier stages become apparent and necessitate iterations on the earlier stages. An essential feature of this approach is that the products of each development phase are verified against the requirements of the previous phase. At certain phases of the development a validation process is carried out to confirm compliance of the product with all functional requirements and non-functional

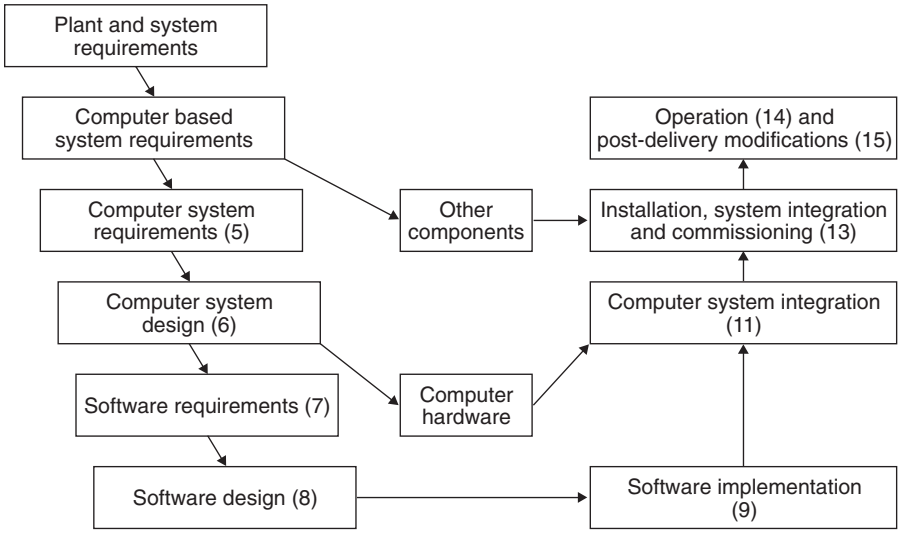

FIG. 1. Development of software for computer based systems important to safety (numbers refer to sections of this Safety Guide).

requirements and the absence of unintended behaviour. Important benefits of such a process controlled step by step are described in Section 3.2.2 of Ref. [4].

2.5. Typical phases of the development process and an outline of the process that may be applied are shown in Fig. 1. The boxes show the development activities to be performed and the arrows show the intended order and the primary information flow. The numbers in parentheses in Fig. 1 indicate the sections of this Safety Guide in which the development activities and products are described. Activities without references are not within the scope of this Safety Guide but are shown for context. Figure 2 illustrates the relationship of verification and validation to the requirements, design and implementation. The choice of the particular development activities and their order in these figures and in this Safety Guide are not intended to require a particular method of development; variations may also have the necessary attributes and be capable of meeting the requirements.

2.6. The development of the computer based system should start with an analysis of the plant and system requirements carried out by engineering specialists, including

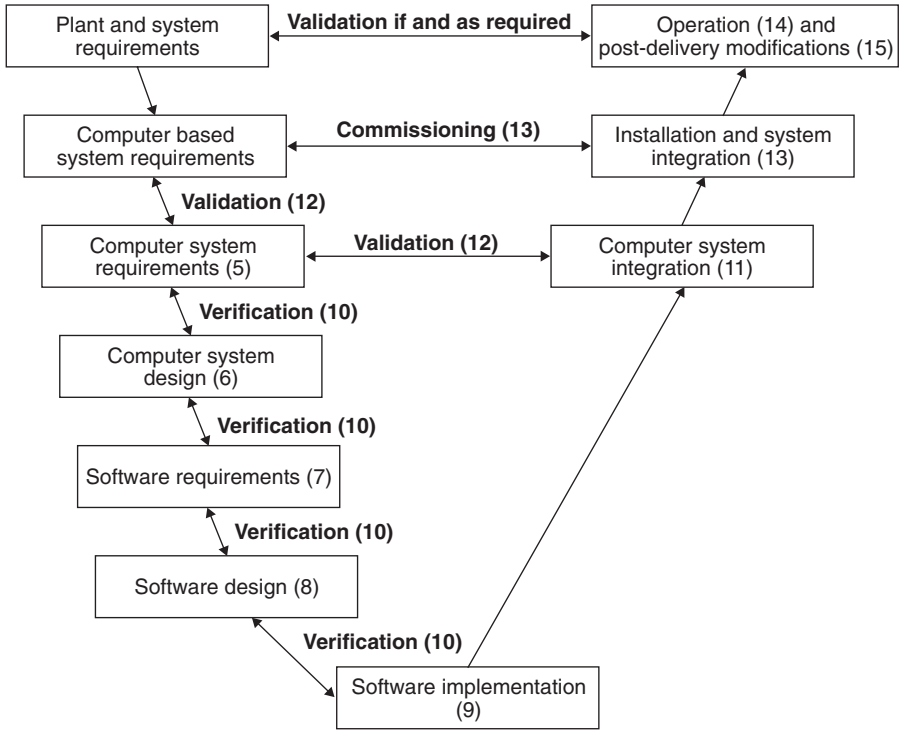

FIG. 2. Verification, validation and commissioning (numbers refer to sections of this Safety Guide).

safety engineers and software engineers. On the basis of the results of this analysis, the requirements for the computer based system are derived. Iterations are normally needed in this analysis before a final set of requirements is identified. There should be a clear demonstration of a systematic derivation since omissions at this stage may result in an incorrect specification of requirements for the safety provisions and hence may result in an unsafe system.

2.7. The first design decision should be to apportion the requirements for the computer based system between the computer system (computer system requirements) and conventional electrical and electronic equipment for measuring plant variables

and actuating the control devices (other components). The designer may have reasons to opt for analog equipment to implement certain functional requirements.[1]

2.8. The computer system requirements are apportioned between the software requirements and the computer hardware in computer system design. The software is then designed as a set of interacting modules which are implemented as code that will run on the computer hardware (software design and software implementation). Next, the software is integrated with the computer hardware to produce the computer system (computer system integration). Finally, the computer system is installed (installation) in the plant for commissioning and operation. One of the stages of the commissioning phase is the integration of the computer system with the other components (which is part of the system integration phase).

SAFETY AND RELIABILITY ISSUES

2.9. Because of the disadvantages mentioned earlier, the quantitative evaluation of the reliability of software based systems is more difficult than that for non-programmable systems and this may raise specific difficulties in demonstrating the expected safety of a computer based system. Claims of high software reliability are not demonstrable at the present time. Hence, designs requiring a single computer based system to achieve probabilities of failure on demand of lower than 10^{-4} for the software should be treated with caution (cf. para. 8.2.2 of Ref. [9]).

2.10. Since software faults are systematic rather than random in nature, common mode failure of computer based safety systems employing redundant subsystems using identical copies of the software is a critical issue. Countermeasures are not easy to implement. Designers assume independence and use diversity and a comprehensive qualification strategy to protect against common mode failures. However, it may be difficult to estimate the degree of success and the benefits of these strategies when software is involved.

2.11. With current technology, it is possible in principle to develop computer based instrumentation and control systems with the necessary dependability for systems important to safety and to demonstrate their safety sufficiently. However,

[1] This Safety Guide does not discuss the aspects of the assessment that relate to other equipment; it focuses on the computer system and particularly on the software that runs on the computer.

dependability can only be demonstrated if a careful and fully documented process is followed. This process may include the evaluation of operating experience with pre-existing software, following specific requirements (see also the Annex). Recommendations for achieving an adequate safety demonstration are given in this Safety Guide.

ORGANIZATIONAL AND LEGAL ISSUES

2.12. There are various organizational and legal aspects of the development project for a computer based system that should be carefully taken into consideration at a very early stage of the project in order to ensure its success. They include such factors as the availability of a suitable legal and administrative framework for licensing computer based systems important to safety and sufficient competence and resources within the organizations involved in the system development process. If these factors are not carefully considered at the conceptual stage of the project, its schedule and costs may be considerably affected. Some of these factors may influence the decision to use programmable digital technology, making this option impractical or even prohibitively expensive.

2.13. Quantification of software reliability is an unresolved issue. Software testing has its limits and the quantification of software reliability for computer based systems may be difficult or impossible to demonstrate. The regulatory position concerning the safety demonstration and reliability required of software should be clarified early in the project planning stage. The approach to be followed to deal with the safety and reliability issues should be determined, documented, made available to and if necessary agreed upon by the regulator. This may include specific regulatory hold points.

2.14. It should be ensured that sufficient competence and resources are available in the regulatory organization, the licensee's design team, the regulatory body's technical support staff and the supplier to fulfil the recommendations (for example those in this Safety Guide) for the software development process and the assessment of its safety demonstration. The licensee should also ensure that the supplier of the computer and/or software will be prepared to release all proprietary information that would be needed for the licensing process.

2.15. The licensee (i.e. the user of the computer system) should establish an appropriate structure to deal with operational and maintenance issues. Occasionally, the licensee may need its own team to perform post-delivery modifications on the software. This team should have the same level of competence and equipment as would have been provided by the original producer.

2.16. These factors contribute to a significant increase in the technical resources that are needed, and therefore have a bearing on costs. The financial commitments may be of an order of magnitude that makes the option of a computer based system prohibitively expensive.

3. APPLICATION OF REQUIREMENTS FOR MANAGEMENT OF SAFETY TO COMPUTER BASED SYSTEMS

3.1. The majority of requirements for management of safety developed over the years for non-computer-based systems are applicable to computer based systems. However, the application of these requirements for management of safety to computer based systems, in view of the differences between software and hardware, is not always straightforward and has, in many cases, new implications. Section 3 provides a brief overview of the most relevant requirements for management of safety, on issues related to requirements for design, design and development, management and quality assurance, and documentation. These requirements for management of safety form the basis for deriving the requirements of the computer based system, especially the non-functional requirements. Documentation should be of high quality, owing to its importance in the design of software and in the safety demonstration.

REQUIREMENTS FOR MANAGEMENT OF SAFETY

Simplicity in design

3.2. It should be demonstrated that all unnecessary complexity has been avoided both in the functionality of the system and in its implementation. This demonstration is important to safety and is not straightforward, as the use of digital programmable technology permits the achievement of more complex functionality. Evidence of obedience to a structured design, to a programming discipline and to coding rules should be part of this demonstration.

3.3. For safety systems, the functional requirements that are to be fulfilled by a computer system should all be essential to the achievement of safety functions; functions not essential to safety should be separated from and shown not to impact the safety functions.

3.4. For computer based system applications, top-down decomposition, levels of abstraction and modular structure are important concepts for coping with the problems of unavoidable complexity. They not only allow the system developer to tackle several smaller, more manageable problems, but also allow a more effective review by the verifier. The logic behind the system modularization and the definition of interfaces should be made as simple as possible (for example by applying 'information hiding' (see Section 3.3.4 of Ref. [4])).

3.5. In the design of system modules, simpler algorithms should be chosen over complex ones. Simplicity should not be sacrificed to achieve performance that is not required. The computer hardware used in safety systems should be specified with sufficient capacity and performance to prevent software from becoming too complex.

Safety culture

3.6. The personnel working on a software development project for a system with a high safety importance should include application specialists as well as computer software and hardware specialists. This combination of expertise helps to ensure that safety requirements, which are generally well known owing to the maturity of the industry, are effectively communicated to the computer specialists. It should be ensured that all personnel understand how their jobs are related to the fulfilment of the safety requirements and they should be encouraged to question activities or decisions that may compromise the safety of the system. This means that the software specialists should also have a good understanding of the application. An appropriate specification language (e.g. a graphical language) can be used for the description of safety functions.

Safety classification scheme

3.7. A safety classification scheme to define the safety significance of the instrumentation and control system functions [2, 3, 9] may be used for the system development process. It directs the appropriate degree of attention (see para. 3.8) by the plant designers, operators and regulatory authorities to the specifications, design, qualification, quality assurance, manufacturing, installation, maintenance and testing of the systems and equipment that ensure safety.

Balance between risk reduction and development effort

3.8. Trade-offs in the achievement of various conflicting design objectives should be carefully assessed at each design stage for the system and its associated software.

A top-down design and development process should be used to facilitate this assessment. Graded design and qualification requirements, when applied to computer system functions, may be derived from a safety classification scheme (para. 3.7). This gradation may be used in order to balance the design and qualification effort. The computer system should meet the criteria for the highest safety class of the functions it is implementing.

3.9. Appropriate measures to warrant the level of confidence that is necessary should be associated with each safety class. It should be noted that for the hardware part of the system the confidence level can be assessed by using quantitative techniques; however, for software only qualitative assessment is possible.

Defence in depth

3.10. Defence in depth as applied in the design of nuclear power plants [1, 10] should be used in the development of the computer based system and its associated software. If a software based system constitutes the main safety feature, defence in depth, for example by means of a diverse secondary protection system, should be provided.

Redundancy

3.11. Traditional design based on multiple redundant instrumentation channels with a voting arrangement, as commonly used in analog applications, has benefits for the hardware of computer systems. However, this type of redundancy does not prevent a failure of the system due to common cause faults in hardware and software design that could lead to impairment of all redundant channels.

Single failure criterion

3.12. The application of the single failure criterion to hardware random failures is straightforward [1, 11]: no single failure should result in loss of the safety functions. However, in software this criterion is difficult to meet, since a fault that causes a software failure is necessarily present in all the replicas of this software (see para. 3.13).

Diversity

3.13. The reliability of computer based systems can be enhanced by using diversity to reduce the potential for software common cause failures. The use of diverse functions and system components at different levels of the design should be

considered. Diversity of methods, languages, tools and personnel should also be taken into consideration. However, it should be noted that although diverse software may provide improved protection against common mode software errors, it does not guarantee the absence of coincident errors. The decision by the licensee to use diversity, the choice of type of diversity or the decision not to use diversity should be justified.

Fail-safe design, supervision and fault tolerance

3.14. System fail-safe features, supervision and fault tolerant mechanisms should be added into the software, but only to the extent that the additional complexity is justified by a demonstrable global increase in safety. When software is used to check the hardware it is running on, then its ability to respond correctly should also be demonstrated. The use of external devices, such as watchdog timers, makes the system response to fault detection more dependable. Defensive design, the use of appropriate languages, including safe subsets and coding techniques, should be used to ensure a safe response under all circumstances, as far as this is achievable. One goal of the computer system requirements phase should be to completely specify the desired and safe response to all combinations of inputs.

Security

3.15. It should be demonstrated that measures have been taken to protect the computer based system throughout its entire lifetime against physical attack, intentional and non-intentional intrusion, fraud, viruses and so on [12, 13]. Safety systems should not be connected to external networks when justification cannot be made that it is safe to do so.

Maintainability

3.16. The computer based system should be designed to facilitate the detection, location and diagnosis of failures so that the system can be repaired or replaced efficiently. Software that has a modular structure will be easier to repair and will also be easier to review and analyse, since the design can be easier to understand and easier to modify without introducing new errors. Software maintainability also includes the concept of making changes to the functionality. The design of a computer based system should ensure that changes are confined to a small part of the software.

Full representation of operating modes

3.17. The requirements for and design of the software for systems important to safety should explicitly define all relations between input and output for each of the operating modes. The software design should be simple enough to permit consideration of all input combinations that represent all operating modes.

Human–machine interfaces and anticipation of human limitations

3.18. The design of human–machine interfaces may have a significant impact on safety. The human–machine interfaces should be designed to provide the operator with a sufficient and structured but not an overwhelming amount of information, and also to provide sufficient time for reacting (see, for example, the thirty minute rule in Ref. [2]). When functions are allocated to the operator, the computer system should allow for the time needed to derive a manual action from the information provided. All operator inputs should be checked for validity as a defence against impairment by operator error. The possibility of the operator overriding these validity checks should be investigated carefully.

Demonstrable dependability

3.19. Not only should the system be dependable, it should also be possible to demonstrate to the regulator that it is dependable. This Safety Guide recommends to licensees how to achieve demonstrable dependability through design and qualification methods that improve traceability and through the production of adequate documents.

Testability

3.20. Each requirement and each design feature should be expressed in such a manner that a test can be done to determine whether that feature has been implemented correctly. Both functional and non-functional requirements should be testable. Test results should be traceable back to the associated requirements.

DESIGN AND DEVELOPMENT ACTIVITIES

3.21. In determining the necessary design and development activities, particular attention should be given to the following topics (paras 3.22–3.27).

Process controlled step by step

3.22. The design and development process should be controlled step by step. This development process can give evidence of correctness by its very construction. This can also ease the verification process and ensure that errors are detected early in the design process.

Reviewability

3.23. Accurate and easily reviewable documentation should be produced for all stages of the development process. The documentation used to demonstrate adequacy to the regulator should be the same as the documentation actually used in the design.

Comprehensive testing

3.24. Comprehensive testing should be applied. Testing is an important part of development, verification and validation. A demonstration of test coverage, including tracing of test cases to source documents, should be provided. The test results, demonstration of test coverage and other test records should be available for third party audit. A comprehensive test plan should be established and made available to the regulator at an early stage of the project.

Use of automated tools

3.25. All tools used should be mutually compatible. The tools should be qualified to a level commensurate with their function in the development of the software and in the safety demonstration. The techniques used to gain confidence in the tools should be specified and documented.

Traceability

3.26. Requirements should be traceable to design; design should be traceable to code; requirements, design and code should be traceable to tests. Traceability should be maintained when changes are made. There should also be traceability in the reverse direction, to ensure that no unintended functions have been created.

Compliance with standards

3.27. The requirements for management of safety, safety requirements and technical standards to be used in the development should be identified. A compliance analysis

should be prepared for the key standard [5] used in the specification and implementation of the computer system design.

MANAGEMENT AND QUALITY ASSURANCE

Clearly defined functions and qualifications of personnel

3.28. The plant management should demonstrate that the level of staffing is adequate. Management should establish the functions and qualifications required of personnel involved in software design, production and maintenance programmes, and should ensure that only qualified and experienced personnel perform these functions. Personnel qualifications should be established for each task within the software development and maintenance process, including the execution of the quality assurance programme [14].

Acceptable practices

3.29. Well established methods, languages and tools for software development should be used. Methods, languages and tools that are still at the research stage should not be used.

Quality assurance programme

3.30. The organization's quality assurance programme should extend into the software development process and should also cover configuration management and change control after the software is delivered. At least for safety systems, independent verification, validation and testing, with independent supporting audits, should also be covered. The quality requirements for the software should be described in a software quality assurance plan which should be required by the quality assurance programme.

Assignment of responsibilities

3.31. Interfaces should be established between organizational entities within the design organization and between the design organization, its client and other organizations involved in the system development process. Controls pertaining to design interfacing should be established and should include the assignment of responsibilities and the issue of documents to and from the interfacing organizations.

Third party assessment

3.32. Third party assessment should be used for safety systems. Its scope and extent should be defined in the description of the quality assurance programme. The team performing the quality assurance and the verification and validation tasks should be independent of the development team. These issues are covered later in this Safety Guide (paras 3.33 and 4.17).

3.33. The objective of the third party assessment is to provide a view on the adequacy of the system and its software which is independent of both the supplier and the user (the licensee). Such an assessment may be undertaken by the regulator or by a body acceptable to the regulator. It should provide confidence in the production process of the licensee and suppliers. The assessment strategy, the competence and the knowledge of the project necessary for the third party assessment in order to provide the necessary level of confidence should be carefully considered. In addition, third party assessment should be agreed to by all parties (regulator, licensee, suppliers) so that the appropriate resources can be made available at the desired time. Some of the third party assessments should involve examination of the process (e.g. through quality assurance audits and technical inspections). Other third party assessments should include examination of the product (e.g. through static analysis, dynamic analysis, code and/or data inspection and test coverage analysis). The assessment of the final product should (as far as possible, in view of time constraints) be undertaken on the final version of the software. This can include third party assessment of intermediate products of the development process, such as software specifications.

DOCUMENTATION

3.34. For confidence in the reliability of a software product, evidence should be provided of the soundness of the development process. The documentation is crucial in providing the ‘transparency’ and ‘traceability’ necessitated by this approach. Documentation for the design and implementation of a trustworthy software product should be clear and precise.

3.35. The set of documents should ensure the traceability of design decisions [14, Q3]. Appropriate documents should be produced at each step of the development process. Documentation should be updated throughout the iterative development, including commissioning and ongoing maintenance processes. The documents available to the regulator should be identical to those used by the designers. The designer should be informed of this early in the project.

3.36. Documentation of the requirements, the design and the code should be clear and precise so that the designers, the programmers and the independent reviewers can comprehend fully every stage of the development and verify its completeness and correctness.

3.37. Good documentation is also essential to maintenance. The proper documentation format should be used to reduce the likelihood of inconsistencies and errors in future maintenance related changes. Documents should have the attributes of comprehensibility, preciseness, traceability and completeness, consistency, verifiability and modifiability, as described in paras 3.38–3.44.

Comprehensibility

3.38. Documentation should be understandable by people with a variety of backgrounds and expertise. The language used should be clear, and when it is formal (e.g. graphical forms), it should have well defined syntax and semantics.

Preciseness

3.39. Requirements and descriptions of designs should be stated formally (with well defined syntax and semantics) with explanations given in natural language. There should be only one possible interpretation of each description or requirement (see para. 5.1.2 of Ref. [4]).

Traceability and completeness

3.40. The purpose of tracing is to demonstrate that the implementation is complete with respect to the computer system requirements and design, and to facilitate the detection of unsafe features in the implementation. Tracing from higher level documents to software documents checks completeness, and tracing back from software documents to higher level documents checks for unspecified items which might be unsafe. Every requirement should be uniquely identified.

3.41. A traceability matrix (or matrices) should be implemented that clearly shows the linkage between computer based system requirements and items which implement each computer based system requirement in the computer system requirements, the computer system design, the software requirements, the software design and the software implementation. This matrix should demonstrate complete test coverage of the requirements for the computer based system during implementation, integration, installation and commissioning.

Consistency

3.42. The documents should not contain contradictory or inconsistent statements. Each piece of information should have a single, identifiable place in the document and information should not be repeated or divided between two or more places. Each requirement, design element or program module should have a unique identifier (this also aids traceability). The notation, terminology, comments and techniques should be used in a uniform way throughout the documentation.

Verifiability

3.43. Verifiability is improved when documents are understandable, unambiguous and traceable. Use of the same model or language for computer system requirements through software requirements and software design will also aid verifiability, but is not necessarily a solution to all problems. When formal languages are used to specify requirements or designs, theorem provers and model checkers may also aid in verifiability.

Modifiability

3.44. Documents should be modifiable, i.e. their structure and style should be such that any necessary changes can be made easily, completely and consistently, and will be easily identifiable.

4. PROJECT PLANNING

4.1. The development process should be carefully planned and clear evidence should be provided that the process has been followed in order to facilitate the licensing of systems important to safety. Project planning should be documented in a comprehensive and specific safety plan for the computer based system, or as a set of plans that cover all aspects of the project. Section 4 describes each aspect as a separate plan, but it is equally valid to place all these plans in a single document. Modifications to the system should not be excluded and provisions should be made for possible further iterations of the safety analysis.

4.2. A development plan should define a set of development activities and the essential characteristics of each activity. Other aspects of the project which should be

planned are quality assurance, verification and validation, configuration management, and commissioning and installation. Figure 1 shows an outline of the process of computer based system development that is presumed in this Safety Guide to apply (see Section 2 for information concerning the choice of the particular development model). Figure 2 illustrates the relationship of verification and validation to the requirements, design and implementation.

DEVELOPMENT PLAN

4.3. The development plan should identify and define the development process that will be used on the particular project. Paragraphs 4.4–4.10 indicate what the computer system development plan should cover.

Phases

4.4. All phases of the development process (Fig. 1) should be identified. Each phase, for example computer system design, consists of specification, design and implementation. The design activity of one phase sets the requirements for the next phase, and specifying the requirements for a phase is part of the design activity of the preceding phase. Implementation, for example, is the process of selecting pre-existing code, including library routines, and producing any additional code needed. Verification should be performed across each phase of the development and before starting the next phase (see Fig. 2 and paras 4.12–4.18 on the verification and validation plan).

Methods

4.5. The methods to be used in the development should be identified. This selection should be related to the quality assurance programme description, in which standards and procedures are established.

Tools

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

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

Documents

4.8. Documents to be produced during each phase should be identified and their contents specified. It should be indicated where all requirements, quality attributes and performance characteristics will be found and what acceptance criteria will be used for the whole project. The documents should provide evidence that the project has adhered to the development plan.

Schedule and milestones

4.9. The schedule for the documents should be established and times should be identified when project review is to be conducted. These are products of a managerial task that encompasses:

— Assessing the availability of resources;
— Estimating the duration of each phase, allowing for iterations;
— Assessing training needs;
— Assessing the adequacy of the facilities and tools available;
— Estimating the time necessary for regulatory review and approval;
— Estimating the time necessary for project review at key points.

Personnel

4.10. A plan should be prepared to ensure that those personnel involved in the development activities are competent in the application of the relevant standards, procedures and methods, in the use of design, programming and analysis tools and methods, and in configuration management and change control practices. Records of their competence should be maintained. If new members are added to the team, they should be closely supervised until they have demonstrated their competence to their line managers.

QUALITY ASSURANCE

4.11. The quality assurance programme description [14, 15] should be prepared and implemented by the licensee and should be available for regulatory review (and possibly approval) before the project begins. A software quality assurance plan should

be produced at the outset of the project. This plan should cover external suppliers and should include at least the following:

(1) Identification of the hardware and software items to which the quality assurance programme description applies and of the governing standards, procedures and tools to be used on the project.
(2) For each document to be produced, indication in the quality assurance programme description of who should review it and approve it for official release.
(3) A description of the project's organizational structure which should include assurance of the independence of quality assurance auditors.
(4) A description of the competence and training needs for personnel involved in the project.
(5) A mechanism for identifying, reporting and correcting non-conformance to standards and procedures.
(6) Identification of all necessary plans, such as the development plan, the verification plan, the configuration management plan and the commissioning and installation plan.
(7) Indication of the number and scope of quality assurance audits on the basis of the safety class of the system.
(8) The procedures for qualifying tools (see para. 4.6).
(9) A mechanism for checking the quality of components from external suppliers; if this mechanism relies on external certification procedures, such as type testing, the description of these procedures should also be included.

VERIFICATION AND VALIDATION PLAN

4.12. Demonstration of the correctness and safety of the system requires a variety of verification and validation activities [16]. In the context of the life cycle of a computer based system, verification is checking a product against the output of the preceding phase and validation is checking a product against higher level requirements and goals (Fig. 2).

4.13. Validation should be performed to demonstrate that the computer system achieves its overall safety and functional requirements. There are two distinct steps of validation. The first step is the validation of the computer system requirements against the plant and system requirements (see para. 2.6). The basis for this validation against these higher level requirements and safety analyses should be explicitly identified in the validation report. The second step is the validation of the computer system implementation against the computer system requirements. Techniques and

explicit validation procedures should be identified in the verification and validation plan.

4.14. Verification should be performed for products of the following development phases (as defined in this Safety Guide):

(1) Computer system design;
(2) Software requirements;
(3) Software design;
(4) Software implementation.

4.15. The techniques to be used to verify the software should be stated in the verification and validation plan, explicit procedures for the techniques should be identified and their suitability for the safety class of the system should be justified. It is expected that they will encompass a combination of techniques, including both static examination of documents and dynamic execution of the implementation.

4.16. It should be determined which records of the verification and validation results should be maintained for the lifetime of the system. The records should provide evidence that all planned activities have been performed, results recorded, and anomalies investigated and resolved. The records should be made available to third parties for audit and review. The records should clearly demonstrate the traceability of all verification and validation work to the relevant source documentation.

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

(1) Technical independence: The work should be done by different people, using different techniques and tools.
(2) Management independence: The work should be led and motivated by different people. The verification and validation team and the development team should have different management lines. Official communication between independent teams should be recorded.
(3) Financial independence: There should be a separate budget with restrictions on transfer of financial resources between development and verification and validation.

4.18. The plan should include a mechanism for recording all instances of non-compliance found during the analysis and ensuring that they are properly resolved by means of the change control process (paras 4.23–4.24).

CONFIGURATION MANAGEMENT PLAN

4.19. Configuration management is an extension of the development plan and quality assurance programme description, and is of sufficient importance that it is described separately here (see also Section 5.2 of Ref. [15] and existing standards pertaining to configuration management).

Version control

4.20. All items of software development, such as compilers, development tools, configuration files and operating systems, should be under configuration management control. All identifiable items, such as documents, components of the software or data structures, should be given a unique identification, including a version number. These items should include both developed items and existing items that are being reused or reapplied. There should be a library or storage area designated to hold all items that are under configuration control in order to make it possible to find and retrieve any identified item in any of its existing versions. There should be a procedure describing when and how a developed or acquired item is to be placed under configuration control. There should be a mechanism whereby at particular points on the schedule a set of configuration controlled items can be identified which represent the 'baseline' for subsequent work. Each item should have a record containing relevant information, such as when it was completed, what changes have been incorporated since the previous version, its current approval status and the persons responsible for creating, reviewing and approving it. There should be approval procedures associated with the quality assurance programme description.

4.21. Appropriate links are required between the procedures and databases for the configuration management of software and hardware. Changes in hardware configuration may affect software verification and validation activities.

4.22. At any time, a quality assurance auditor or regulatory auditor should be able to request and directly receive any items of the set that constitute the most complete and current description of the system and project (the current baseline). This set of items should be identical with those currently being used as a basis for ongoing development or analysis, with the exception of work in progress that has not yet been approved or released.

Change control

4.23. Once an item has been placed under configuration control, it should only be changed according to a well defined procedure that includes an impact analysis, and that procedure should result in a new version of the item rather than a replacement of the existing identified item. The change control procedure should maintain records of the problems that were identified which necessitated the changes, how the problems were analysed, which items were affected, which specific changes were made to correct the problem and which versions and baseline were produced to solve the problems. The change control procedure may also identify responsibilities for approving changes if these are in addition to, or different from, the approval mechanism for basic items. In general, a change should involve a repetition of all processes used to create the item originally, including all analyses, starting from the highest level item affected. A regression analysis should be used to identify the tests required to maintain the verification and validation record. The procedures of the regression analysis and its results should be documented.

4.24. After completion of the development (and delivery to the site), a different change process may be applied since the organization and people responsible for maintenance may be different from the original developers. This is discussed further in Section 15. Prior to the application of the maintenance change process, a change procedure should be documented, and the need to repeat the analyses undertaken during development should be considered. Any decisions not to repeat any of the analyses should be documented and justified.

INSTALLATION AND COMMISSIONING PLAN

4.25. After a system has been constructed and validated as a stand-alone system, it should be integrated with other plant systems and tested within the real plant environment. This process of installation and commissioning should be carefully planned to co-ordinate the proper transition from development to use and the handover from the developers and verifiers to the users and maintainers.

4.26. The installation and commissioning plan should cover the following:

(1) The sequence of steps for proper integration of the system into the plant and the corresponding plant states needed for safe introduction of the new or changed system;
(2) The required interactions with the regulator, including any approvals or hold points that should be respected before the system can be put into operation;

(3) The commissioning test cases and sequence and the corresponding plant states needed to confirm proper functioning of the system in the plant environment;
(4) The interfaces between the other components and the new or existing plant systems, and the tests needed to check the proper functioning of each interface;
(5) The duration of any probationary period;
(6) A description of the records and reports that will be generated to describe the results of commissioning;
(7) The initiation of the process to instruct and inform the users and maintainers;
(8) The transfer of the configuration management and change control process from developers to maintainers, including the mechanism to address any anomalies found during commissioning.

5. COMPUTER SYSTEM REQUIREMENTS

5.1. The computer system requirements define, as a minimum, the functional and non-functional properties of the computer system that are necessary and sufficient to meet the safety requirements for the plant that have been defined at a higher design level. The specification of computer system requirements is a representation of the necessary behaviour of the computer system. Precise definition of the interfaces to the operator, to the maintainer and to the external systems is an integral part of the product provided as the output of this phase. At this stage, the definition of interfaces is limited to the functional and non-functional properties of these interfaces; their design or implementation may be as yet undetermined.

5.2. The elaboration of the computer system requirements is based on the results of high level plant design (Fig. 1). Safety analyses, for example accident analyses, transient analyses or plant safety analyses (based on postulated initiating events and safety criteria to be met), are an essential part of this design. In addition to safety requirements, some additional requirements not directly associated with safety are added at this stage of the design, such as requirements for availability. Consequently, the definition of the safety system requirements is the result of a co-operative effort by experts from different disciplines. This part of the design is outside the scope of this Safety Guide, but it provides the specifications of protective actions and of performance criteria that should be covered by the system requirements. Conversely, the system requirements should also be validated against those specifications for completeness and consistency.

5.3. The derivation of these requirements is a critical step in the development process because errors and deficiencies at this stage will eventually challenge the validation process if not detected during development. Additionally, defects in the computer system requirements are a potential source of common cause failures in redundant subsystems containing identical software.

RECOMMENDATIONS

General

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

5.5. An accurate and clear description of these requirements should be written before the start of the next stages of the project. This description should be understandable to all regulator and licensee experts concerned: process engineers, safety engineers, computer system engineers and plant engineers.

5.6. The description of the computer system requirements should be easy to use in order to verify the adequacy of the computer system and the software specifications and to define the specifications of the tests that will validate the system when the design is complete.

5.7. The description of the requirements should be based on a precise model of the system and of its interface to the plant environment. An example of such a model in terms of monitored and controlled variables is given in Section 5 of Ref. [4].

Validation

5.8. The system requirements should be validated, i.e. their soundness, consistency and completeness should be established in relation to the specifications which result from the plant safety analyses.

5.9. Tools used to analyse the consistency and completeness of the computer system requirements should be validated to a level of confidence proven to be equivalent to the level required from the design process described in paras 5.10–5.23.

System interfaces

5.10. The system interfaces should be designed so as to facilitate involvement of the operator in protective actions such as manual backup, interventions and manual resetting following a reactor trip. Input data and user commands should be reconfirmed by operators and validated by the system before being eligible for processing.

5.11. When the operator interface involves a visual display unit, attention should be paid in the design to adequate response times, navigation and help facilities [2, 3]. Every display field should be used for one purpose only in a given mode and context. For instance, the use of a single field to display the values of the same parameter in different engineering units should be avoided. All interfaces should be consistent and should use the same names and identifiers. A task analysis should be performed to demonstrate the adequacy of the manual safety provisions and any associated displayed information.

5.12. The system interfaces should also be designed to facilitate in-service inspections without causing spurious protective actions.

5.13. The system interfaces, such as the interface to a plant network, should be designed so as not to violate other protective functions performed by external systems. Malfunctions and failures of external systems or support systems should cause the system to be brought into a safe state.

5.14. There should be a precise definition of the system boundaries, i.e. of the interface between the computer based system and the plant. In particular, the interfaces of the system with the sensors and actuators, the operator, the maintainer and any other external system should be specified.

5.15. The format of input, output and display data and the organization of alarms should be carefully specified.

5.16. Automatically loaded data, for example from a floppy disk, should be machine verified against the original data. A read-back facility should be provided for calibration data that are loaded manually. The automatic generation of default values by the system without warning should be prohibited.

5.17. The physical environment and, for replacement systems, previously installed systems place constraints on the values of environmental variables and parameters. These restrictions should be identified and documented. In particular, compatibility with existing systems in the plant and protection against the propagation of failures of non-safety systems should be addressed.

Functional safety requirements

5.18. For safety systems, the functions of the system that are necessary and sufficient to meet the safety requirements for the plant should be specified. These functional safety requirements should be expressed in terms of the additional constraints that should be maintained by the system on the plant parameters, which should be kept within specified limits, and in terms of effects the system should have on the plant.

5.19. A safety analysis should also be made for safety related systems to determine functional safety requirements.

Non-functional requirements

5.20. Non-functional requirements should specify the following:

— The properties of the system behaviour, i.e. the constraints on environment, accuracy, time and performance imposed on that behaviour (the time and performance constraints should be within the allowable time limits determined by plant safety analyses).
— The relevant dependability attributes, such as reliability, availability and security, required of the system behaviour. In particular, consideration should be given to limiting the reliability claimed for the computer based system. The security requirements should be derived from the safety policy that has been defined for the computer based system environment and should take into account the security procedures that should be implemented (see para. 14.5).
— Robustness, i.e. how the computer based system will react to potential failures at the interface with the plant, such as sensor failures, or inputs outside the expected range.
— Whether and where physical separation is needed (for example between safety and control functions).
— When necessary to meet the reliability requirement, the form of diversity to be introduced into the system's design. Diverse features can be specified in terms of diverse detection, measurement, voting and actuation methods; in terms of different functions to react to the same postulated initiating event; and in terms of independent and diverse system components. The appropriate types of

diversity required should be evaluated against the reliability requirement of each function.
— How the possible need to replace some parts of the system is taken into account.

Analysis

5.21. For safety systems, all the safety functional and non-functional requirements should be shown to be the result of the plant safety analysis and to be determined by it.

5.22. As part of the system safety demonstration, the computer system requirements should be subjected to a failure analysis. Potentially incorrect system outputs should be identified, their impact on the plant and the environment should be evaluated, and adequacy of defence in depth should be confirmed.

5.23. The sources of potential common cause failures should be identified. The operating modes in which similar signal trajectories could exist in more than one channel or subsystem should be identified. In this respect, particular attention should be paid to parts of the system or functions that will be implemented in software.

DOCUMENTS

Contents

5.24. The documentation produced in this phase should cover the following items:

— Specifications of functional requirements;
— Specifications of non-functional requirements;
— Specifications of interfaces with operator, maintainer and external systems;
— Need for archival records and data for post-accident analysis;
— Specifications of validation tests and of their coverage;
— Validation report for the computer system requirements.

Specifications of functional requirements

5.25. The functional requirements should be specified in a form which is independent of the means of implementation, for instance in terms of the functional relations that should be maintained by the system. These functional relations should specify the following:

— The relations between the state environmental variables that result from physical, natural or other constraints imposed by the environment and by other systems and which the system may not violate;
— The additional relations that should be established and maintained by the computer based system between monitored variables (process variables, operator signals) and controlled variables (output to actuators and indicators) when it operates in this environment.

5.26. These relations should be arranged in a way that reflects the functional organization of the system, in order to induce the necessary mappings to this organization of the structure of the computer system and of the modular structure of the software that will be defined at the later stages.

5.27. Variables should be denoted by standard mathematical engineering notation. They should be carefully defined, for example by means of diagrams, co-ordinate systems, signs or scales.

5.28. The different modes (groups of states) and classes of modes in which the system may have to operate should also be clearly identified, especially if the interfaces between the system, the plant and the operator are different for these modes.

5.29. A well defined formalism should be used to describe functional relations. Decision tables containing the functional relationships between inputs and outputs may be used for that purpose, in particular for verification and for the definition of validation tests.

Specifications of non-functional requirements

5.30. For safety systems, these specifications should be demonstrably derived from the plant safety analysis and should document the time constraints, the performance constraints and the dependability requirements.

5.31. The dependability requirements, in addition to specifying the required reliability and availability, should document and justify the following:

— The required component qualifications;
— The application of safety requirements such as the single failure criterion;
— The requirement for physical and functional separation (the detection of a postulated event by more than one plant parameter and the independence of parameter processing is an example of such a requirement).

Specifications of interfaces with operator, maintainer and external systems

5.32. The functional and non-functional specifications for the system interfaces should be clearly expressed for all possible distinct modes of behaviour of the system. Special models and specification methods such as those used for the description of communication protocols may be used.

Need for archival records and data for post-incident analysis

5.33. The frequency and accuracy necessary for keeping archival records and data for post-incident analysis, and whether such data need to be generated continuously or only in response to specified plant conditions, should be specified in a document.

Specifications of validation tests and their coverage

5.34. The test specifications should cover the entire functionality of the system and its interface with the plant and should be shown to be derived from the above functional and non-functional requirements.

5.35. For safety systems, the test specifications should also specify tests which are statistically valid at system level and commensurate with the required reliability (see para. 2.9).

5.36. The test specifications should state the expected results.

5.37. For safety systems, these test specification documents should be verified and approved independently by people who did not take part in the elaboration of the specifications for system requirements. These test specification documents should also be reviewable by regulators.

Validation report for computer system requirements

5.38. A validation report should include the following:

— The specifications derived from plant safety analyses against which the system requirements have been validated;
— The steps carried out to perform this validation;
— The conclusions of this validation.

5.39. The validation report should, in particular, be reviewable and made available as early as possible to the licensing authority.

Documentation style

5.40. General recommendations on documentation are provided in paras 3.34–3.44.

6. COMPUTER SYSTEM DESIGN

6.1. The initial version of the computer system design results from a mapping (by systematic and structured or formal means) of the system requirements to be satisfied by the computer system into those to be satisfied by an appropriate combination of the following:

— Pre-existing or predeveloped software or firmware (e.g. the operating system);
— Hardware or application specific integrated circuits;
— Software to be developed or to be produced by configuring predeveloped software.

Subsequent refinement of this computer system design will result in additions owing to the activities discussed in paras 6.3–6.6.

6.2. Section 7 describes how the parts of the specification for the computer system design which pertain to software are refined to produce the software requirements. In a similar manner, those aspects of the specification for the computer system design which pertain to the hardware part should be refined and applicable standards should be selected (e.g. Ref. [17]), but this activity is beyond the scope of this Safety Guide.

RECOMMENDATIONS

6.3. The computer system architecture selected should provide the system interfaces specified by the computer system requirements and should implement the non-functional computer system requirements, such as those pertaining to performance and reliability.

6.4. For design, verification and validation, and maintenance purposes, there is a need to incorporate ‘added’ requirements into the relevant specification document at the appropriate level of abstraction. These could include the following:

(1) Requirements resulting from lower level decisions on computer system design or software design;
(2) Requirements resulting from lower level decisions on design made for parts of the system outside the computer system, but which may affect the computer system (such as selection of instrumentation which needs software compensation or hardware filtering).

6.5. Whether these requirements are reflected in changes to the specification of system requirements or to lower level specification documents, a means should be available for maintaining configuration control on these added requirements so that an assessment can be made of whether validation coverage of them is necessary.

6.6. The means by which the computer system should be demonstrated to be adequately safe should be addressed in the document for computer system design. Consideration should be given to the design issues set out in paras 6.7–6.26.

Separation of safety aspects from non-safety aspects

6.7. The safety system should be dedicated to the task of performing its safety functions. Where it is necessary and justified for non-safety functions to be part of the safety system, an analysis should be performed to determine whether the whole system should be classified as a safety system, and the safety function should not be jeopardized by the other functions.

6.8. Since simplicity facilitates the achievement of reliability, it should be considered whether to encapsulate the functions and components that are related to safety and to isolate them from the other systems. This can be done by removing the non-safety functions and components from the computer system, by use of a distributed computer system or through the use of appropriate ‘firewalls’ within a centralized computer system.

6.9. After separation, the non-safety functions and components may be implemented with less restrictive or more resource intensive methods. In addition, the separation of safety functions and components can also be beneficial in satisfying the single failure criterion, because failures in the non-safety functions and components would have no impact on the safety functions and components. However, care should be taken to ensure that the separation is in fact achieved.

6.10. Separation of safety functions and components may lead to an increase in overall system complexity. This extra complexity, which is added at the interface to achieve the separation, should not overwhelm the intended reduction in complexity.

The more complex the computer system architecture, the more difficult it is to provide a safety demonstration.

6.11. In the event of common access to a hardware or software component, such as a data link, by safety functions and non-safety functions, failure of the component should not prevent the execution of the safety functions. This type of common access should be avoided in the computer system design if it has the potential to impair the safety functions.

Redundancy, channelization and voting logic

6.12. Redundancy in the computer system to meet the requirements for reliability and on-line testing should usually be provided. Redundancy enables the system to cope with random hardware failures but it is not proof against common cause failures.

6.13. To reduce the chance of cross-link failures of redundant equipment, the redundant subsystems should be separated electrically and physically. The correct overall result may be obtained by voting the results obtained by each subsystem even when a component has failed or a subsystem is taken out of service for maintenance of the failed component. Because the voting logic itself necessitates common usage by all the subsystems, adequate and sufficient buffering should be used to ensure that the separation of the redundant equipment in the subsystems is maintained. Also, redundant subsystems should run asynchronously.

6.14. The need for overrides for commissioning the integrated computer system should be considered since it may have an effect at the hardware–software interface.

Diversity

6.15. To reduce the potential for common cause failures, diversity should be incorporated into the computer system architecture to the extent necessary to achieve the safety and reliability requirements for the overall computer system. Equipment diversity means that different types of equipment are used to perform redundant functions. Functional diversity means that different methods are used to carry out a particular function.

6.16. The extent to which software can be made diverse and free from common cause failures is not fully resolved. The use of diverse software in redundant systems (discussed in more detail in Section 9) can reduce the probability of common cause failure in comparison with the use of identical software. However, there is no way to

predict accurately enough the effect of diversity. Experimental results have shown that in diverse versions the number of identical software defects varies between 10% and 100% of the total number of software defects [18]. This aspect, together with the limits of the reliability that can be claimed for the software, should be taken into consideration in determining the allocation of functions between hardware and software. An additional aspect that should be considered is that the voting logic is a complex issue, especially for software.

Fault detection and fail-safe capability

6.17. The computer system should be designed to detect faults internal or external to the system which may prevent it from carrying out its safety function. If such a situation occurs, the computer system should default to a 'safe state', even in the event of a loss of electrical power to the computer system and its output actuators. The determination of a safe state is not always straightforward and it is likely to vary with the situation in the plant (in some cases the safest reaction of the computer system will be no reaction except to alert the plant operating staff).

6.18. If a particular type of fault is insufficiently severe to necessitate a fail-safe reaction, the computer system should still provide notification to plant operating and maintenance staff so that they can effect a repair of the fault before it can contribute to a multifault failure which does have safety implications.

Fault tolerance

6.19. The computer system's hardware and software architecture should be such that the system operates in a predefined safe manner after the occurrence of a limited number of hardware and software faults. Techniques such as the use of alternate data sources and redundant architectures should be considered. The limitations of these techniques and the associated procedures should be fully described in the computer system design and should be reflected in the manuals of plant operating procedures.

System process monitoring and equipment monitoring

6.20. The computer system's architecture should be designed with account taken of the need to monitor plant system processes and plant equipment, as well as the operational status of components of the computer system itself. For example, in control systems for which there is a need to compare like signals from several channels in order to distinguish failed plant process signals, the means for achieving this without compromising channel separation should be determined.

Testability in operation

6.21. A computer system used in systems important to safety should be designed so that it is periodically testable during plant operation in order to verify the continued operability and reliability of the system. This may require adjustments to the computer system's architecture.

6.22. The frequency of in-service testing should be determined on the basis of the safety analysis in which the necessary test interval has been determined. The scope and coverage of in-service testing should be determined in the computer system design and should be shown to be adequate for its purpose.

Accuracy and response times

6.23. The combination of hardware and software and other elements which make up the computer system should achieve the required accuracy and response times to satisfy the overall system performance requirements. The analysis of accuracy and timing should take into account all possible sources of delay or signal distortion which could be introduced by all elements in the chain from the sensing devices for the physical condition of the plant to the final control actuating mechanisms. This includes inaccuracies or delays introduced by the software and at the computer system's hardware–software boundary. In particular, it should be taken into account that the use of a computer system's hardware or software to condition plant signals (such as by filtering out noise and removing relay contact instability) may result in reduced response times.

6.24. To assist in the identification and analysis of system response issues, deterministic communication protocols and scheduling techniques for computer resources should be used. The analysis should include consideration of the worst case demands placed on the system (such as accident conditions that simultaneously cause changes in many input signals and require many output actuations). The more complex the communications architecture of the intracomputer system, the more difficult it is to provide a convincing safety demonstration. Techniques which may be of assistance in providing such a demonstration are those which address the analysis of communicating processes (such as calculus of communicating systems, Petri nets or state transition diagrams [4]).

Non-functional requirements

6.25. The computer system design should also cover non-functional aspects of the computer system requirements such as the ability to cope with extreme temperatures,

radiation exposure or static electricity, as applicable to the particular installation (i.e. equipment qualification), immunity to radiofrequency interference and immunity to electromagnetic interference such as noise or electrical surges conducted via power, signal, communication or ground lines. It should be noted that immunity to radiofrequency interference and electromagnetic interference can only be demonstrated on the installed system.

6.26. The future need to replace some parts of the system should be considered in the computer system design.

DOCUMENTS

Computer system architecture

6.27. The documentation on the computer system design should contain a clear description of the computer system architecture selected. In addition, it should include a justification of the computer system design in terms of the required dependability of the system and the ability to implement all functional requirements. Such a justification can be provided, in part, by evidence of the use of a structured methodology. Also, the selected architecture should demonstrate a balance between simplicity in concept and the capacity to satisfy performance requirements. The justification of the computer system design can be partially achieved with the possible use of modelling and analysis, formal method iterations or prototyping. The results of such analyses should be described in the documentation on the computer system.

6.28. In addition to refining the functional and performance requirements of the computer system which are contained in the specification of system requirements, the documentation on computer system design should cover the additional requirements derived from the following (paras 6.29–6.40).

In-service system testing

6.29. Decisions about how in-service system testing will be done (for example on-line or off-line, automated or manual) and about how the maintenance needs will be met (for example by calibration, isolation and repair of failed components, use of spare installed devices, means of manual channel tripping) could necessitate additional hardware interfaces as well as additional requirements for software functionality.

6.30. The selection of the medium to be used to store the software logic (for example RAM versus ROM) will influence the extent of the on-line or periodic off-line checks that should be carried out to ensure continued integrity of the computer system.

Assessment of needs for annunciation and operator interfacing

6.31. Human factor engineers should determine (for example by human factor task analysis) the nature of annunciation and the equipment to be used to provide it (for example messages that are alarm window based or visual display unit based). They should also determine how best to provide plant operators and maintainers with the information they need to fully understand the plant status at all times and with the means of making operational adjustments when necessary. These decisions could necessitate additional hardware interfaces and software functionality.

6.32. Although all aspects of the computer system should be maintained under an appropriate level of configuration control, it should be considered whether some part of the system (for example set points) should be more easily changeable than other parts. If so, requirements for a suitable means of achieving this should be specified (see Section 15).

Analysis of the hardware and software interfaces

6.33. An analysis of all internal interfaces (such as those for drivers and communications protocols) should be carried out and documented to demonstrate that all interface properties are specified. This should include items such as the definition of the information to be transmitted across the interfaces, consideration of the characteristics of synchronous and asynchronous protocols, and selection of transfer rates.

6.34. The safety properties that should be demonstrated should be identified (for example ‘liveness’ or resolution of contention and absence of deadlock) and confirmed as having been satisfied by the analysis of the interface properties.

Input–output signal processing and data handling and storage

6.35. Signal processing can be done in computer hardware or software, or external to the computer system, in order to make the plant process data available to the computer system. Related design decisions (with the justification for the choice) and

the characteristics of the physical (for example memory or disk) and logical (for example databases) means for storing and subsequently accessing the data should appear in the documentation on computer system design after careful consideration of the fitness for purpose of their properties.

Hazard analysis for the computer system

6.36. A hazard analysis should be carried out for the computer system's architecture and the functionality within it to identify any specific risks that might compromise the safety function and thus to indicate any need for changes to the architecture or additions of functions (such as self-checks) to mitigate the effects of hazards. Such an analysis should be included in the safety demonstration.

Safety constructs for the computer system

6.37. Complementary to the hazard mitigating functions that are added in response to specific risks identified through the hazard analysis for the computer system, features such as watchdog timers, program sequence checking, reinitialization of variables and other fault detection mechanisms constitute good practice. Because of the need for simplicity of the system, these features are added only to the extent that they do not make the software unnecessarily more complex.

Security considerations

6.38. As part of the need to maintain strict configuration control of the computer system, the computer system design should determine how intentional or inadvertent corruption of the computer system's functionality (for example by unauthorized access, unauthorized code or a virus) is to be prevented [12, 13]. This should include details of procedural or other controls on how changes to the system are to be made and verified and how unauthorized changes are to be prevented. There should be an analysis of the threats to security together with a justification of the level of security to be implemented.

Coverage of fail-safe mechanisms

6.39. The documentation on the computer system design should specify the coverage achieved by the fail-safe mechanisms that are specified in order to satisfy the

non-functional computer system requirements. It should also state any limitation associated with reliability claims.

System integration requirements

6.40. The documentation on the computer system design should contain the system integration requirements for the following:

— The pre-existing software or firmware (e.g. the operating system);
— The hardware or the application specific integrated circuits;
— The software to be developed or to be produced by configuring predeveloped software.

These system integration requirements are derived from the interface analysis for the hardware and software (paras 6.33 and 6.34) and would include, for example, a definition of hardware–software interfaces and handling of associated exceptions. This information is necessary for the production of complete test specifications for verification of the integrated system.

Documentation style

6.41. The documents for the computer system design should be part of a document continuum which starts with the specification of the system requirements. The parts of the system requirements which pertain to the computer system are brought forward into computer system requirements. In some cases, the functional and performance requirements contained in the system requirements are brought forward without change, but more often they are refined to a lower level of abstraction to reflect details or decisions made which are pertinent to the computer system.

6.42. Within the documents for the computer system design, the presentation of the requirements as a whole and of each constituent element therein should be unambiguous and complete and should be traceable back to the rationale for its inclusion. This should be consistent with the means used to document the system functional requirements, but at a lower level of abstraction.

7. SOFTWARE REQUIREMENTS

7.1. Software requirements are the subset of the computer system requirements that will ultimately be implemented as computer programs. All computer system

requirements should be carried completely and correctly into either the software requirements or the hardware requirements. Software requirements will include the requirements which follow from the choice of the particular computer system design. The verification of the software requirements against the upper level requirements is an important step in the licensing process. Therefore it should be possible both for developers and for those who did not establish the requirements (reviewers and regulators) to trace the origin of the software requirements and verify them.

7.2. Preparing the software requirements is a process that is closely tied to designing the computer system. The software requirements describe what the software component must do in order that, when that software is executed on the chosen set of computers with associated peripherals, the overall system requirements are met. In the design of the computer system, functionality may be allocated in different ways between software and hardware to achieve an appropriate balance between such concerns as performance, cost, size, simplicity and compatibility. This will determine what the software requirements or hardware requirements will be. Specifiers should also be aware of the capabilities of the computers, tools and similar existing systems so that it is feasible to fulfil the software requirements in the software design and the software implementation.

7.3. The software requirements provide a link between the system level and the computer capabilities. As such, they should be understandable both in terms of processes in the external environment and as operations provided by the computer code.

7.4. If the computer system requirements are sufficiently detailed and their documentation is sufficiently formal, and if parts of the computer system design and of the code are generated by tools, then a separate software requirement document may be unnecessary for those parts. However, those parts of such computer system requirements from which code is generated or reused should be regarded as a statement of software requirements against which subsequent code should be verified. Also, any separately compiled modules that are included by the code generator should be supported by separate documents for the software requirements.

RECOMMENDATIONS

General

7.5. The software requirements should include the description of the allocation of system requirements to software, with attention to safety requirements and potential

failure conditions, functional and operational requirements under each operation mode, performance criteria, timing and constraints, failure detection, self-supervision, safety monitoring requirements and security requirements.

7.6. The software requirements should be properly documented and understandable in order to facilitate their use by software designers, computer system engineers, regulators and, if necessary, process engineers and safety engineers. They should be written in a form that is independent of the particular design and implementation that will be applied. Any division or structuring of requirements should be done in order to aid understanding of the requirements and to facilitate traceability to higher level documents.

7.7. The software requirements should be analysed for ambiguities, inconsistencies and undefined conditions; deficiencies should be identified, clarified and corrected. The software requirements should be verifiable and consistent. The functional requirements should be stated in mathematical terms where possible. Non-functional software requirements, such as minimum precision and accuracy, time behaviour, independence of execution threads and fail-safe behaviour, should also be stated explicitly and, whenever required, in quantitative terms.

Functions to be performed

7.8. The software has a major function in implementing the functional requirements of a computer based system. The software requirements should be the translation of those functional requirements into particular transformations of digitally coded inputs into digitally coded outputs. The required transformations from inputs to outputs may be in the form of logical expressions, mathematical functions or relations, or algorithms (sequences of operations) if the particular steps of the algorithm are part of the requirements. For better traceability and comprehensibility, the naming and encoding of the input, output and internal variables should, as far as possible, be meaningful in the system environment.

Safety

7.9. Functions that are important to the safety of the system should be indicated as such in the document on the software requirements. For each relevant output, the safe state which can be used in the event of a detectable but unrecoverable failure should be indicated. The computer system design may also have introduced additional relationships between software inputs and outputs, or between various outputs, that are relevant to safety. These should be described as additional safety requirements.

Reliability and availability

7.10. The computer system requirements include reliability or availability requirements, and these requirements should be carried forward into the software and hardware. Given the reliability required of the computer based system and the reliability that can be expected from the hardware, it may be that certain software requirements should be included so that the system will be able to meet its reliability targets (see Section 6).

7.11. An overall software reliability target may be stated, but it should be understood that the achievement of such a target will be less demonstrable than the fulfilment of other types of requirement. It is extremely difficult to demonstrate that quantitative reliability requirements for software have been met. Currently available methods do not provide results in which confidence can be placed at the level required for systems of the highest importance to safety, and therefore this Safety Guide does not provide guidance on the use of software reliability models. If applicants propose the use of software reliability models for certification or commissioning, a rationale for the model should be included in the certification or commissioning plan and agreed with the regulatory authority.

Interfaces

7.12. A description of the interfaces between the software and the operator, instruments (sensors and actuators), computer hardware, other systems and other software that may be present on the same hardware should be provided. This defines the boundary of the software.

Design constraints

7.13. By the time the software requirements are being written, choices will have been made that will constrain the software design and the software implementation. These choices should be documented or referenced as part of the software requirements where necessary. For example, the choice of computer hardware will constrain the choice of compiler and operating system, and the need for diversity will put constraints on the software architecture. Such design constraints should be justified and traceable.

Model

7.14. Software requirements may be based on a model of the system to be implemented (Section 5, and Section 5 of Ref. [4]). In this case the model and its

application should be well defined and documented with specification of the requirements. For example, control software is sometimes described using a finite state machine model. The use of the finite state machine model should be described so that the requirements for state transitions and functions specific to particular states can be properly understood.

Timing performances

7.15. A description of the time limits that are required for the transformations performed by the software should be provided. These may include minimum and maximum times between inputs and the corresponding outputs, or between successive, related outputs; or expected timing characteristics and sampling rates of the inputs that the software is required to handle (such as behaviour that depends on the frequency of arrival of certain inputs). The software as a whole may be required to detect and recover from faults or failures within a specified time.

Computational accuracy

7.16. Computational accuracy with regard to the numerical information to be processed by the software should be specified. Given that the processing will take place on a digital computer with limited word size, it should be recognized that some inaccuracies will occur during calculations. The level of inaccuracy that is tolerable should be stated explicitly to guide the software designers in their choice of the software representation of numerical data (i.e. the precision) and the particular method of computing required functions.

Security

7.17. Some security needs for the computer system will be translated into requirements on the software (such as validity checks on inputs, stored data or even the program itself). Security needs may also indicate that some information manipulated by the software, such as safety related set points, should have privileged access only.

DOCUMENTS

Contents

7.18. The main purpose of the documents on the software requirements is to form the basis for software development and licensing. Therefore these documents

may contain aspects related to software design, to licensing (for example risk considerations, recommendations for functions or engineered safety features) and to other items that provide the background for specific requirements.

Documentation style

7.19. The documentation on the software requirements should be prepared according to standards whose formality should not preclude readability. The documentation on the software requirements should be verifiable and maintainable. The use of a formal specification language may help to show the coherence and completeness of the software requirements.

7.20. Additional detailed guidance on the documentation and style of the documentation on the software requirements can be found in international standards for high reliability software used in safety systems in nuclear power plants [5].

7.21. General recommendations on documentation are provided in paras 3.34–3.44.

8. SOFTWARE DESIGN

8.1. Software design is the division of the software into a set of interacting modules and the detailed description of those modules. It is important that the design be well structured and understandable to the implementers, maintainers, testers and regulators. The design should demonstrably include all software requirements and should not contain any unsafe item. The design will normally address the architecture of the software and the detailed design within that architecture.

8.2. The architecture of the software refers to its organization into modules. These modules include processes, abstract data types, communication paths, data structures and display templates. There are numerous methods of dividing the software into modules with differing emphasis on particular kinds of module. The choice of architecture is very important in determining the simplicity of the modules and of the interactions between them, which in turn determines the simplicity of their specifications and of the verification and validation tasks.

8.3. The way the modules fit together (the software architecture) can be described in more than one way, even within a single design. This Safety Guide does not recommend any particular way of representing the software architecture.

RECOMMENDATIONS

8.4. At least two levels of design are recommended: the architecture of the software and its detailed design (for more detailed recommendations, see Section 5.1.1 of Ref. [4], and standards and technical books on software engineering). The required attributes of software design are discussed in paras 8.5–8.12.

Avoidance of complexity

8.5. In systems important to safety, unnecessary complexity should be avoided at all levels of design. The simpler the design, the easier it is to achieve and to demonstrate all other attributes. It also gives greater confidence that the software is fully understood.

Safety

8.6. The software design phase should address the hazards identified in previous analyses (see the analysis in paras 5.21–5.23 and the hazard analysis for computer systems in para. 6.36) and the requirements that have been identified as important to safety. These requirements should include the necessary self-supervision features to detect hardware faults that may occur at execution time. Software should also check its own control flow and data. The property of liveness of a hardware–software module can be monitored by fault detection mechanisms such as watchdog techniques. On failure detection, appropriate action should be taken in terms of recovery, halting procedures and error messages. All errors, persistent or transient, should be recorded. Supervision and integrity of code access and data access in particular should be ensured by appropriate software and coding techniques if they are not ensured by hardware, for example by holding codes and data in erasable programmable memories.

Understandable and modifiable structure

8.7. In order to facilitate reviewing, the organization of the software architecture should form a hierarchical structure to provide levels of abstraction. Information hiding (see Section 3.3.4 of Ref. [4]) should be used to make piecewise review and verification possible and to facilitate modifications. The use of graphical techniques can aid understanding. It is preferred that the design should be described formally (with well defined syntax and semantics), with explanations given in natural language (see Sections 5.1.3.7, 5.2.3 and 5.2.4 of Ref. [4], but it should be noted that other formalisms exist).

8.8. Interfaces should be simple and expected changes should be isolated within a single module or a small number of modules.

Traceability

8.9. To facilitate the tracing of requirements, each design element, such as a module, a procedure, a subroutine or a file, should have a unique identifier.

Predictability

8.10. The architecture chosen should be deterministic. A design should be selected that makes the operation of the software predictable in terms of response to inputs and the time to produce a response. A fixed, repeated sequence of operations (such as polling) may generally be used rather than interrupts. Communication protocols should be deterministic and should not depend on the correct operation of external systems.

Consistency and completeness

8.11. The design should contain no contradictions and no ambiguities. The description of the interfaces between modules should be complete. Both sides of each interface between modules should match and there should be, as far as possible, a consistent partial ordering and use of variable names between module input and output interfaces.

Verifiability and testability

8.12. The design and its description should be such that it is possible to demonstrate that each software requirement has been met and to verify that the implementation is correct with respect to the detailed design.

DOCUMENTS

8.13. The documentation on software design should provide technical information on the overall architecture of the software and on the detailed design of all software modules. Relevant implementation constraints should also be specified.

8.14. A documented demonstration should be provided that the software design addresses the hazards identified in previous analyses and the requirements that have been identified as important to safety.

Software architecture

8.15. The decomposition into modules and the mapping of this decomposition onto the requirements should be described with proper justification. The architecture should also take into account the unavoidable changes that may occur during the lifetime of the system, so that software maintenance and upgrades can be performed conveniently.

8.16. The existence of interfaces between the various software modules and between the software and the external environment (documented in the software requirements) should be identified and specified in the documentation.

8.17. If the system includes multiple processors and the software is distributed amongst them, then the description of the software architecture should specify which process runs on which processor, where files and displays are located, and so on.

8.18. The architecture should take into account the constraints on modules and interfaces that may result from the decision to use diversity.

Implementation constraints

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

8.20. Documented evidence should be provided that, on detection of a failure, the actions taken for recovery, halting procedures and error messages maintain the system in a safe state.

Detailed design

8.21. Each software module identified in the software architecture should be described in the detailed design. The particular content of the description will depend on the module type. In general, a module description should completely define its interface with other modules and should completely define the module function and its function in the overall software.

Documentation style

8.22. General recommendations on documentation are provided in paras 3.34–3.44.

8.23. Diagrams and flow charts should be used as long as the meaning of the elements of the diagrams is well defined. Other common techniques for describing design include data flow diagrams, structure diagrams or graphical methods (see, for example, Section 5.1.3.7 of Ref. [4]).

9. SOFTWARE IMPLEMENTATION

9.1. The input to this phase consists of the specifications for the internal design and for the interfaces of the software modules. The outputs are the listings containing the source code and the executable code and the results of unit tests and module interface tests.

9.2. Code can be produced from the system specifications in various ways which are essentially combinations of two distinct approaches: the classical development process through the stages of specifications and design which are detailed in Sections 5–8, or, as mentioned in Section 5, the use of code generating tools which receive as input an application oriented description of the system with a high level language. The choice between these two approaches depends on the tools and resources available to the parties engaged in the project and should take into account, in particular, trade-offs between design and the demonstration of the dependability of tools. The recommendations in this section apply to all possible combinations of the two approaches.

RECOMMENDATIONS

Verifiability and reviewability

9.3. The production of code that can be demonstrated to be a correct implementation of the software specifications is a major issue. The code should be verifiable against these specifications. If verification is made by human inspection, the code should be readable, adequately annotated and understandable. Validated tools should be used to facilitate the code verification process (see Section 10).

Change and version control

9.4. Code cannot usually be produced correctly in a first version. The requests for changes and the modifications made in implementation should be carefully controlled and coherence between the successive versions that are produced should be maintained.

9.5. The implementation may reveal omissions or inconsistencies in the software design specifications or the computer system requirements. A system for requesting formal change and controlling modifications should therefore be in place in the implementation phase to deal with these omissions and inconsistencies. Records of these changes should be kept and should be available to the regulator. A system should also be in place to maintain coherence between the various versions of the modules that will be produced (see Sections 4 and 15) and to guarantee full coverage of the testing of the changes.

Programming languages

9.6. For safety systems, the programming language (or the subset which is used) should have a rigorously defined and documented syntax and semantics.

9.7. The programming language (or the subset which is used) should have adequate expressive power. The ease with which module specifications can be refined towards an implementation depends very much on the expressive power of the programming language.

9.8. Application oriented rather than machine oriented languages should be used for the application software. Assembly language should only be used for modules of limited size and functionality, and only if this is fully justified by real time performance or compatibility constraints. Otherwise, the language should have sufficient facilities for supporting modular programming, and for structuring the code in terms of procedures, subroutines or subprograms with specifications and calling sequences distinct from their code bodies.

9.9. The programming language and its translator should not, by their design, prevent the use of error limiting constructs, translation time type checking, run time type and array bound checking, and parameter checking. These run time checks, however, may be dispensed with if the implementation is mature and thoroughly trusted or if they would require excessive processor power at run time.

Use of tools

9.10. Validated tools should be used since they relieve programmers from tasks of the code production process that are prone to manual error, such as programming and verification.

9.11. A suitable set of tools should be selected for the implementation. These may include translators and code generators, debuggers, linkers, and testing and other verification tools.

9.12. Thoroughly tested translators that are available and maintainable throughout the system lifetime should be used. If no thoroughly tested translator is used, additional analysis and verification (see Section 10), manual or by the use of other tools, should demonstrate that the translation is correct.

9.13. These tools should be compatible with other tools used in other phases of the development.

Diversity

9.14. 'Software diversity' is achieved by a development technique by which two or more functionally identical versions of a program (variants) are developed from the same specifications by different programmers or programming teams with the intent of providing error detection, improved reliability or reduced probability that programming or translator errors will influence the output results. Although the use of software diversity can result in a lower probability of common cause failure, it should not be considered a remedy for all errors. Residual coincident errors may remain and the probability of residual common cause failure remains difficult to assess. The use of 'functional diversity' should be considered, since it may provide a significant reduction in residual error. With functional diversity, different means are proposed to achieve the same safety objective. Where two or more implementations are performed in software, the techniques recommended for achieving an acceptable level of software diversity should still be used (paras 6.15 and 6.16).

9.15. Software diversity can be achieved by different techniques, the use of recovery blocks and N-variant techniques being the most common. Software diversity can also be applied to the different aspects of software implementation, including tests. For example, independent programming teams, diverse solution spaces, and different working environments, tools and run time support systems (operating systems,

compilers) can be used. All these aspects and techniques should be considered before a choice is made, if software diversity is to be used. The choice and its justification should be properly documented, with account taken in particular of the extra complexity introduced by these techniques.

9.16. The voting logic may raise complex design issues that should be considered. However, as with redundant systems, the voting should be performed as far down the decision chain of each separate subsystem as possible in order to maximize subsystem diversity. For example, where two or more measurements are to be used to indicate the need for a reactor trip, then the voting should be applied after comparison of each signal with its trip setting, i.e. voting should be on the binary state signal and not on the measurement. Voting strategies such as 2 out of 3 voting should also be considered to reduce the likelihood of spurious action. Other aspects of voting to be considered are the hardware tolerances in the measurement process (including the granularity of the data), the bandwidth (skewness) allowed for the comparisons and the distance between voting points.

9.17. A capability for on-line error detection is a positive aspect of software diversity that should be considered. Diverse variants are more likely to have different outputs in the case of untested input vectors or unanticipated changes in the environment, so that the erroneous behaviour is more likely to be detected.

9.18. Any type of diversity implies a degree of independence. If equipment, software or functional diversity is used, then the claimed level of independence should be demonstrated.

Module programming

9.19. It is essential that, before the programming of a module begins, the specifications of its design and the specifications of its interfaces be complete and available.

Code simplicity

9.20. The code of each program of a module should be kept simple and easy to understand, both in its general structure and in its details. Recursive structures, code compactions, optimizations and tricks which hide the functionality of the code and are used for the programmer’s convenience to the detriment of the program readability should be avoided.

Coherent data structures

9.21. Data structures and their naming conventions should be used uniformly throughout the whole system. Each data structure identifier should consistently reflect its type (array, variable, constant, etc.), its scope (local, shared, etc.) and its kind (input, output, internal, etc.).

Avoidance of language insecurities

9.22. Most programming languages suffer from insecurities, which make it difficult or even impossible to detect violations of the language rules either by the compiler or by analysis of the program text. If these insecurities cannot be eliminated by discarding some language features or by adding additional static–semantic rules, their use should be avoided, or at least restricted, identified and thoroughly verified.

Coding rules

9.23. Recommended programming techniques should be prescribed in an approved detailed set of coding rules and deviations from those rules should be identified during the verification of the software modules. Guidance on coding rules can be found in international standards for highly reliable software used in plant safety systems (e.g. [5]).

Self-supervision and self-checking

9.24. The software design includes the necessary self-supervision features to detect hardware faults that may occur at the time of execution. Software should also supervise its own control flow and data. These supervision features may not all have been anticipated in the software design requirements (see Section 8). In this case, a request for modification of these requirements should be made.

Operating systems

9.25. Only thoroughly and satisfactorily tested operating systems should be used. For safety systems, only operating systems that comply with the recommendations of this Safety Guide should be used. The use of the operating system should be restricted to the indispensable functions. Those functions should be identified and should have

well defined interfaces. Each particular function should always be called in the same way. Relevant and documented operating experience of the use of these functions should be available.

9.26. Precautions should be taken to ensure that the separation of safety functions and non-safety functions is not jeopardized by the use of a common operating system, other running support software or network communication software.

Testability

9.27. It should be possible to have access to all parts of the executable code (including access to ensure the correct behaviour of the run time support code and of the hardware fault supervision mechanisms) for testing by one or a combination of the techniques mentioned in this Safety Guide, of either the 'black box' or the 'white box' type (see, for example, Section 9.2.2 of Ref. [4]).

DOCUMENTS

9.28. The code of each program of a module should be displayed in a specific section of a document together with the contextual information necessary to verify the correctness of this program in relation to its specification.

9.29. The contextual information provided should be sufficient to serve as a basis for understanding and testing a program of a module in isolation from the other programs of the module and from other modules. This contextual information on the module should contain either a replicated description of, or a reference to, the relevant fragment of the software design specifications, the logical assertions, or the preconditions and postconditions that should be satisfied by every program of the module. It should also identify the other programs and modules that call upon, and are called upon, and the program input and output variables and parameters, together with their range of validity. To facilitate the reviewing of the code, this document should be no longer than one or two pages.

9.30. The choice of the programming languages used should be justified and documented. The description of the language syntax and semantics should be complete and available.

9.31. The choice of all the tools used should be justified and documented. The process by which the tools have been validated should also be documented.

9.32. Additional detailed guidance on the documentation and style of programs can be found in international standards for highly reliable software used in plant safety systems (e.g. [5]).

10. VERIFICATION AND ANALYSIS

10.1. Verification is the process of ensuring that the products of each development phase (including any pre-existing software) satisfy the requirements of the previous phase (see Section 2 for a definition of development phases). Analysis is the process of checking that the products of each development phase are internally consistent and properly apply the chosen techniques. The objective of the verification team should be to provide a full and searching examination which is commensurate with the required reliability.

RECOMMENDATIONS

General

10.2. A variety of techniques for verification and analysis are available (see, for example, Section 8 of Ref. [4], and Ref. [16]). Each technique provides some assurance of product quality, but no single technique provides complete assurance. A variety of complementary techniques should therefore be used. Manual review and inspection can generally be performed on all documents. Additional forms of verification and analysis are considered desirable or even essential to the achievement of assurance. For example, although static code analysis may eliminate the need for costly repetition of a test programme, it should not be considered a substitute for testing. All forms of static analysis are based on models of program behaviour; testing is essential to explore aspects of program behaviour that these models do not capture.

10.3. The scope and depth of each technique applied for verification and analysis should be fully justified.

10.4. Verification results will provide an insight into the design process. Records of the numbers and types of anomalies should be maintained. These records should be reviewed to determine whether or not any lessons can be learned, and appropriate process improvements should be made. It may be difficult or unwise to implement

such process improvements within the scope of the development of a particular system; the process improvements will benefit future system development projects. Documents describing processes and techniques should be subject to configuration management that should be distinct from the configuration management of the computer system documents.

10.5. Manual techniques such as reviews, walk-throughs, inspections or audits should be applied to the verification of all life cycle phases and they may be the only applicable technique for verification of those phases which precede source code generation. Since these are manual methods it is important that the means by which the results of such reviews are to be recorded is considered. Use can be made of checklists. However, care should be taken in the construction of checklists to optimize their usefulness (for example, the required response to a checklist item should be clear and not open to interpretation). The means by which the verifiers are to record the results of their reviews should be stated in the verification plan together with a justification of the chosen method.

10.6. Inspection of the documentation on software design and software implementation should be undertaken prior to the design of the software test cases. In this way the structure of the software can be revealed and the design of the test cases can be informed by the inspection process. The test case specifications should be fully documented and reviewed. The test coverage should be justified. The reviews should also check that the design and coding constraints have been observed in the production of the software.

Static analysis

10.7. The following techniques of static analysis may be used to build confidence in the correctness of the source code:

— Verification of compliance with design, coding and standards constraints;
— Control flow analysis;
— Data usage and information flow analysis;
— Symbolic execution;
— Formal code verification.

10.8. When software requirements have been formally specified, it is possible to undertake formal code verification. However, formal verification generally requires considerable expertise, and therefore consulting competent analysts should be considered. Further guidance on formal verification and program proofs can be found in Section 5.2.4 of Ref. [4].

10.9. Static analysis should be performed on the final version of the software.

Testing strategy and coverage

10.10. Testing is an analysis of the software that involves execution of the object code (preferably on the target processor or on a simulator). If this is done, the correctness of all the software, translation tools and hardware elements may be challenged during testing. Challenges to the software under examination can be constructed by designing suitable test cases. In addition, a test strategy should be designed which allows an incremental buildup of the full software (such as bottom-up or top-down). A typical test program may involve initial testing of the module's programs at the lowest level in the software hierarchy followed by testing the module's programs at progressively higher levels. In this way the already tested lower level programs can be integrated into the test environment and can be used to test the higher level module's programs.

10.11. The means by which full functional coverage of the tests is to be demonstrated should be defined in the verification plan. It should be possible to trace each of the test cases using a traceability matrix as described in para. 3.41. All non-functional requirements implemented in software, such as numerical precision and performance, should be tested.

10.12. The means by which it is to be demonstrated that non-functional requirements are met should be documented in the verification plan. Performance testing should cover, for example, all timing requirements relating to speed of response to inputs, time to detect and recover from faults, and capability to accept all specified input rates.

10.13. Targets for the structural coverage metrics of the testing (such as 100% statement coverage and branch coverage) should be stated and justified in the verification plan. Any deviation from the targets stated in the plan should be justified and documented.

10.14. The test programme should pay particular attention to the testing of interfaces (such as module–module, module's program–module's program, software–hardware, internal–external at the system boundary). It should be ensured that all data passing mechanisms and the interface protocol are performing satisfactorily.

10.15. Consideration should be given to the means by which the testing of exception conditions (such as divide by zero, out of range) is to be performed. This might

require the use of specialized test tools (such as fault injection tools, in-circuit emulators).

10.16. All ranges of input variables should be tested. Since exhaustive testing is impractical, consideration should be given to techniques such as equivalence class partitioning and boundary value analysis which can help to reduce the number of test cases required to provide sufficient testing coverage to the software under test.

10.17. All modes of system operation should be considered during the design of the test cases.

Preparation and conduct of testing

10.18. The means by which full coverage of the tests is to be demonstrated should be defined in the verification plan (see Section 4). Any deviation from the targets stated in the plan should be justified and documented.

10.19. Test plans should be designed so as to facilitate regression testing, by ensuring that tests are repeatable and require minimal human intervention.

10.20. Personnel who are planning and conducting the tests should be given appropriate training in the use of test tools, procedures and techniques. These personnel should be independent from the development team.

10.21. Test equipment should be calibrated against traceable standards. The equipment used to conduct the tests should be uniquely identified and recorded in the test procedures to ensure that tests are repeatable and to assist with the investigation of anomalies.

10.22. All outputs from the system under test should be monitored. Any variation from the expected results should be investigated. The results of the investigation should be documented.

10.23. Test records should be maintained. The records should be auditable by a third party. The coverage of the tests should be readily evident, including the traceability of each test to the appropriate functional requirements.

10.24. Any anomalies in test performance should be reviewed and, if it is determined that there is a need for a modification to the test procedure, an appropriate procedure for change control should be applied, for example by reference back to the originator of the test procedures.

10.25. If it is determined that errors exist in either the hardware or the software, then any necessary changes should be made under control of agreed modification procedures (see Section 15). Errors should be analysed for cause, the lack of early detection in the development process should be investigated and appropriate regression testing should be undertaken (see para. 10.19).

10.26. Further guidance on testing can be found in Refs [4, 5].

Hazard analysis

10.27. It has already been pointed out that the computer based system and its interfaces with the plant should be evaluated at various phases of the development for their potential contribution to hazards at plant level (possible techniques are outlined in Section 8.3.9 of Ref. [4]). When such potential critical behaviours are identified, they should be traced into the computer system design, the software design and the code in order to identify parts of the design and of the software that necessitate special design features. In addition, these hazards should be traced back into the requirements and should be incorporated into the plant safety analysis as appropriate.

10.28. It should subsequently be verified whether the plant safety criteria have been satisfied in the software design phase and whether the software implementation phase has affected safety or introduced new hazards. Because of the difficulty of these verifications and the impact that they may have on the design in terms of proposed changes or additional features, they should be carried out as the design proceeds rather than only after the implementation is complete.

Tool assessment

10.29. The tools employed in producing software belong to the two following broad categories:

(1) Software development tools, whose output (possibly transformed) becomes part of the program implementation and which can therefore introduce errors. Code generators, compilers and linkers are examples.
(2) Software verification tools, which cannot introduce errors (but may fail to detect them). Tools for static analysis and test coverage monitors are examples.

10.30. For a tool of either category to be employed, there should be a precise definition of the tool's functionality. For a software development tool, the domain of applicability should be precisely known, and for a software verification tool, the analyses or checks it performs should be well defined.

10.31. In all cases, a tool should have sufficient safety dependability to ensure that it does not jeopardize the safety of the end product. Therefore a software development tool whose output is used without further review should be of the highest dependability level. This may be relaxed if its output is subjected to verification as described in the preceding paragraphs (10.1–10.28) or if reverse engineering is used as described in para. 10.32. For a software verification tool, the requirements may also be relaxed somewhat, on the grounds that its output will be closely scrutinized.

Reverse engineering

10.32. In some circumstances it is possible to use reverse engineering techniques [19] to provide confidence in a translation, for instance from a design description to a source code, or from a source code to a machine code. This reverse engineering involves applying in reverse (or inverting) the translation process. Its feasibility depends on whether the original translation process is well defined, traceable, straightforward and (uniquely) invertible. This inversion has been performed satisfactorily, and even mechanized, for output of generators of code from designs. The technique has also been used successfully in reconstructing source code from machine code where the source has been fairly low level. The technique cannot be rigorously applied to reconstruct high level language programs, for which the mapping to executable code is extremely complex. However, the technique of translating both source and machine code into a common formal language and then employing machine assistance to compare the two versions has potential and therefore should be considered.

Evaluation of operational history

10.33. Feedback on the in-service behaviour of software may provide confidence in its ability to perform its desired functions in addition to other verifications. If such information is to be used, then an analysis of the relevance of its past usage, especially its compatibility with the proposed new environment and the relevance to the proposed new application, should be provided.

DOCUMENTS

10.34. Verification and analysis should be documented. The documents should provide a coherent set of evidence that the products of the development are complete, correct and consistent.

10.35. It has already been recommended (Section 4) that a verification and validation plan be produced at an early stage of the project. This plan should define the set of verification and analysis techniques to be used, should provide justification that this set will give sufficient evidence and should define the particular records, reports and documents that will be prepared. The plan should also cover the management issues of who will perform each process and how these processes will be co-ordinated with each other and with the development processes.

10.36. Since verification compares the results of one phase with the results of a previous phase, preparation for a verification can begin as soon as that previous phase is complete. Such preparation should be documented as the first part of the verification results. This documentation should consist of the following:

— Checklists and desired responses;
— Obligations for proof;
— Test cases and expected results (test plans);
— Test coverage targets and justifications;
— Tool assessment criteria.

10.37. During a verification process, verifiers should record sufficient information about the process that it could be repeated with confidence of achieving the same results. If any verification tools are used, they should be completely identified (including version and configuration) and any input or set-up parameters used should be recorded.

10.38. The results of a verification should be recorded to demonstrate that all prepared aspects have been covered. It should be clearly indicated which results met expectations and which did not (anomalies). All anomalies revealed by the various verification activities and analyses should be recorded and investigated. Records of all such anomalies should be maintained, including evidence of the means by which they were resolved. Where it is determined that a change is required to the documentation and source code, the procedures for changes described in Section 4 should be applied.

10.39. The test plan should be as comprehensive and as complete as is reasonably practicable. Some of the recommendations (such as functional and structural coverage) will produce more test cases than others. However, they all provide opportunities to identify test cases which should be included in the test plan. Further guidance on the generation of test cases can be found in Section 9 of Ref. [4].

10.40. The test plans, procedures, expected results and reports on the tests should be maintained and should be available for quality assurance audits and third party assessments. The test procedures should present the rationale for each test case and provide for tracing of the test cases to the relevant source documents. The expected test results should be stated (with their method of derivation) in the test documentation prior to execution of the tests.

10.41. The results of hazard analysis should be reported and the completeness of the coverage should be assessed.

11. COMPUTER SYSTEM INTEGRATION

11.1. This activity consists of three parts:

— Loading of all software into the hardware system;
— Testing that the software–hardware interface requirements are satisfied;
— Testing that all the software can operate in the integrated software–hardware environment.

11.2. The verification of the system integration is performed after the designer's system integration and testing. In this phase, the objective should be to generate evidence which will demonstrate that the system integration has been properly controlled.

RECOMMENDATIONS

11.3. The hardware integration activity should precede the computer system integration phase (guidance can be found in Ref. [17]).

11.4. A system integration plan should be derived from the system integration requirements, which form a part of the documentation on the computer system design (see para. 6.40).

11.5. A documented traceability analysis should be performed as part of the verification activity to demonstrate that the system integration requirements are complete with respect to the computer system design specification.

11.6. It should be ensured that only verified hardware and software modules are submitted to the system integration phase. This implies the need for strict configuration control of the integrated system, including the following:

— A controlled build list for all hardware modules;
— The retrieval of correct module versions from the software library.

11.7. The software and hardware module versions used for the system integration phases should be currently verified versions. Unverified versions may be used provided that there is adequate documented justification, i.e. either no subsequent changes are required after verification or the whole integration phase is restarted if changes are needed.

11.8. The verification of the system integration should be designed to check that all integration interfaces (such as hardware–software or software module to module) have been challenged.

11.9. For safety systems, the team designing and undertaking the system integration verification tests should be independent of the designers and developers. Project related communication between the test team and the designers should be recorded. All changes to approved test procedures should be recorded and subjected to reapproval.

11.10. The general testing requirements established in Section 10 should be followed.

11.11. As far as possible, testing carried out in this phase should be based on white box principles.

11.12. The set of tests should be justified. In constructing test cases, consideration should be given to the following:

— Coverage of all integration requirements (both stated and implied, for example robustness tests which demonstrate that the system responds safely to all possible interface conditions);
— All hardware–software interfaces;
— Coverage of full ranges (including out-of-range values for interface signals);
— Exceptions handling (for example demonstration of acceptable behaviour when hardware failure occurs);
— Equivalence class partitioning and boundary conditions;
— Timing related requirements (such as response time, input signal scanning, synchronization);
— Accuracy.

Further guidance on construction of test cases can be found in Section 12.

11.13. Use of verification methods other than testing and tools which can support the verification of system integration should be considered.

DOCUMENTS

11.14. The documentation produced within this phase should include the following:

— A plan for verification of system integration;
— The results of traceability analysis;
— A justification of methods used for verification of system integration;
— A justification of the level of intervention in the normal operation of the system to observe its behaviour during the testing.

11.15. The report on verification of system integration should cover, among other things, the following:

— Review of test coverage;
— Review of module integration and testing;
— Review of results of hardware–software integration tests;
— Assessment of internal timing performance;
— Review of traceability.

12. VALIDATION OF COMPUTER SYSTEMS

12.1. Validation is the process that demonstrates that the computer system will perform its intended function, as defined by the functional and non-functional requirements (see Section 5). As mentioned in Section 2.6 of Ref. [6], this process of the safety demonstration is regarded as essential since current analysis techniques do not enable the correctness of a system to be fully demonstrated. More precisely, validation is regarded as the test and evaluation of whether the integrated computer based system hardware and software demonstrate compliance with their functional and non-functional requirements. Validation is usually performed off the site.

12.2. Because of the digital nature of computer based systems, no opportunity exists to take advantage of extrapolation and interpolation in deciding the number of tests

to perform in order to demonstrate compliance with the requirements. This limitation obliges the validator to justify the amount of testing performed in relation to the required system reliability. Use of equivalence class partitioning and boundary conditions may be considered. There is also the problem of guaranteeing that certain modes of operation and interactions between the plant and the computer based system (those not readily testable at the validation stage) will be tested satisfactorily at the commissioning stage. Finally, the level of input simulation for an acceptable demonstration should be shown to be adequate.

RECOMMENDATIONS

12.3. The general testing requirements established in Section 10 should be followed. Testing should be conducted in a disciplined and orderly manner, according to a quality assurance programme description and procedures that are controlled within the quality assurance regime and in accordance with Ref. [5]. Additional guidance can be found in Section 2.6 of Ref. [6] and in Ref. [7].

12.4. The team designing the validation tests should be independent of the designers and developers. Project related communication between the test team and the designers should be recorded. All changes to approved test procedures should be recorded and subjected to reapproval.

12.5. The testing carried out at this stage is essentially black box testing based on the computer system requirements (see Section 5). However, an analysis of features added during the various design stages should be carried out to identify those features which it is desirable to test at the system level (i.e. by injection of appropriate test signals at the system boundary), for example the validation of correct fail-safe actions following the detection of bad quality or out-of-range input signals.

12.6. The set of tests should be justified. In constructing test cases, consideration should be given to the following:

— Coverage of all requirements (including robustness tests and security features);
— Coverage of full ranges (including out-of-range values for input signals);
— Exceptions handling (for example demonstration of acceptable behaviour when input failure occurs);
— Equivalence class partitioning and boundary conditions;
— Timing related requirements (such as response time, input signal scanning, synchronization);

— Accuracy;
— All interfaces (such as the hardware–software interface in system integration and external interfaces during validation);
— Stress and load testing (for example to reveal cliff edge effects);
— All modes of operation of the computer system, including transition between modes and recovery after power failure.

12.7. A traceability analysis should be performed to demonstrate that the validation requirements (for test or evaluation) are complete with respect to the computer system requirements.

12.8. The system hardware subjected to validation testing should be fully representative of the final configuration of the computer based system at the site installation and the software of the system should be identical.

12.9. The system should be subjected to a wide range of static input and dynamic input tests. It is important to exercise all parts of the system using realistic scenarios involving all inputs (dynamic testing). However, because it is neither reasonably practicable nor safe to test the behaviour of a safety system using real accident scenarios on the plant, the system should be tested using a simulation of the accident scenarios. A test harness could be used to record the results.

12.10. The dynamic tests should be based on an analysis of the plant transients induced by postulated initiating events. The test profiles should be representative of the expected plant parameter variations that would place demands on the computer system. The number of tests executed should be sufficient to provide confidence in the system's dependability.

12.11. Consideration should be given to the possibility of subjecting the computer system to statistical testing. The goal is to produce an estimate of the computer system's reliability by subjecting it to a series of statistically valid tests (see Section 9.3.7 of Ref. [4]). These tests should be randomly selected from the operational input space of the computer system.

12.12. Purely random testing, which implies that these tests are selected from all possible combinations of inputs, can also be used since it provides a convenient means of generating a large number of test cases which may reveal unexpected side effects. When the combination of inputs is not included in the operational input space of the computer system, the corresponding test is defined as a robustness test.

12.13. For dynamic and statistical testing, the number of tests applied should be related to the reliability requirements for the computer system. Evidence should be provided to demonstrate that the method of error detection is adequate.

12.14. Tests should be performed to confirm set point accuracies and hysteresis.

12.15. The system operation and maintenance manuals should be validated, as far as possible, in this phase.

DOCUMENTS

12.16. The documentation of the validation phase should include the following:

— Possible updates to the validation plan;
— An analysis of those features added during the various design stages;
— A traceability analysis demonstrating that the validation requirements (for test or evaluation) are complete with respect to the computer system requirements;
— The validation results;
— The plan of statistical tests, schedule and results;
— An analysis of the plant transients induced by postulated initiating events for use in the statistical tests;
— The plan of random tests, their schedule and their results.

13. INSTALLATION AND COMMISSIONING

13.1. Following delivery to the site, the computer system needs to be installed in the plant. This is a progressive process involving the securing of the various items of equipment in their allocated locations, the connecting of power and the repeating of some of the tests performed at the manufacturer's premises in order to demonstrate that the computer system has not suffered damage during its transportation and installation. This phase is followed by the connection of the plant cables to the computer system: these may be both communication cables and plant signal cables. After their connection, it should be demonstrated that each cable has been connected to its correct terminals.

13.2. Commissioning is the process during which nuclear power plant components and systems, having been constructed, are made operational and verified to be in

accordance with design assumptions and to have met the performance criteria. Commissioning includes both non-nuclear and nuclear tests. In this phase, the computer based system is progressively integrated with the other components and other plant items (system integration). Testing within the plant environment is an important part of the commissioning of computer based systems. In particular, modes of operation and interaction between the computer based system and the plant which could not be readily tested at the validation stage should be tested at the commissioning stage.

13.3. Installation and commissioning activities can take place either on new plant or during a plant modification (such as retrofitting and upgrading). The recommendations given in paras 13.5–13.10 apply to both cases of new plants and of modifications to existing plants.

13.4. The validation of the computer based system against the plant safety requirements is the final step in the verification and validation process [2, 3].

RECOMMENDATIONS

13.5. The complete system should not be installed in the plant until the computer system integration and validation phase has been completed. However, if this is not possible, justification for the adequacy of the tests performed after installation should be provided. The safety demonstration of the installed system should be performed to the same level as it would be at the manufacturer's premises (see Sections 8 and 9 of Ref. [4] and Sections 7 and 8 of Ref. [5]).

13.6. Consideration should be given to repeating some or all of the tests outlined in Sections 10–12, since equipment, and thereby software, may be damaged during installation. Particular attention should be given to the testing of external system interfaces and to the confirmation of correct performance with the interfacing equipment. Testing carried out in the commissioning phase should confirm, wherever practical, the previous tests of the correctness of the normal and exceptional responses of interfaces.

13.7. The computer based system should be subjected to an extended on-site probationary period during which the operation, testing and maintenance of the system should be as representative of the in-service conditions as possible. Where appropriate, the new system should be run in parallel with the old system for the probationary period, i.e. until sufficient confidence has been gained in the adequacy of the new system. The validation of operation and maintenance manuals should be

completed in this phase. During this period the system should be subjected to routine test and maintenance activities. A log should be kept of any revealed faults and corrective actions undertaken.

13.8. It should be recognized that as the computer system is progressively integrated into the plant, the point will be reached at which the computer system will be required to perform its safety function. All practicable testing of the computer system necessary to support continued commissioning and operation should be completed before this point is reached. Also, all necessary hardware and software for maintenance of the system should be in place before the system needs to support its safety function.

13.9. The team producing the commissioning tests should be independent of the producers of the computer based safety system. It should be ensured that those involved in the commissioning programme are competent to perform the tasks allocated to them.

13.10. Strict configuration control of the computer system (hardware and software) should be maintained during the commissioning programme. Any changes required in this phase should be subjected to a formally documented change process (see Section 15 on post-delivery modifications).

DOCUMENTS

13.11. Sufficient documentation should be produced to demonstrate the adequacy of the commissioning programme to satisfy comprehensively the installed computer based safety system. A documented demonstration of the adequacy of the test coverage, including tracing of tests to source requirements (safety system requirements), should be provided by the commissioning team. The documentation justifying the adequacy of the commissioning programme should be maintained by the plant operator and should be available for third party assessment. The documents produced in this phase should include justifications for the following:

— The adequacy of any system integration and validation testing performed after installation;
— The number of repeats of the pre-installation testing (sometimes termed factory testing or acceptance testing) to be performed in this phase;
— The commissioning test coverage, including the tracing of tests to source requirements (computer system requirements), all commissioning records and

a completion report plus a record of anomalies and test failures, including an explanation of their resolution;

— The capability of the computer system to perform the appropriate safety function in all phases of commissioning; in addition, the justification should demonstrate that all practicable testing of the computer system necessary to support continued commissioning and operation has been completed before the system needs to perform the appropriate safety functions.

14. OPERATION

14.1. The operation phase of a computer based system follows installation, commissioning and any regulatory approval for use. The system becomes part of the plant and is operated by the licensee. The operation phase of a particular system continues until it is removed or replaced (possibly by a modified version as described in Section 15).

14.2. Operation of a computer based system will involve some maintenance activities to keep the equipment in good condition and to repair any failed components. The processes that support the safety systems of nuclear installations in their operational use have the potential to jeopardize that safety system's fitness for purpose if they are not of a sufficiently high integrity.

RECOMMENDATIONS

Probationary period

14.3. As is recommended in para. 13.7 for new systems, following a modification a probationary period of operation should be imposed during which an increased frequency of in-service testing of the computer based system should be undertaken.

Corrective actions

14.4. After failure of a hardware component, corrective actions should be limited to one-for-one replacements of hardware and to the reloading of the existing software modules; these actions should not include any modifications (see Section 15 on post-delivery modifications).

Security

14.5. On the basis of the security policy that has been defined for the computer based system environment, appropriate security procedures — for instance password management — should be implemented (for example to guard against unauthorized access and viruses). See para. 5.20 for security requirements and para. 6.38 regarding measures for security in the computer system design.

14.6. Secure storage arrangements and procedural controls should ensure that only authorized software versions are loaded into the plant equipment. The correct performance of the computer based system should be demonstrated before it is returned to service.

Calibration data

14.7. Calibration data should be of a sufficiently high accuracy not to degrade the computer based system's reliability. For safety systems, such data should be generated automatically by a system that has been developed to the same standards as the computer based system. Where the data generation system has not been developed to this standard, the calibration data produced should be checked by diverse methods performed by an independent group.

14.8. The operator should be required to validate the data entered according to an agreed procedure before proceeding to the next item. All entered data should be separately archived and checked by an independent party before the safety system is reinstated.

14.9. A suitable test of the subsystem concerned should be performed following the change of calibration data so as to demonstrate its correct operation.

DOCUMENTS

14.10. Full and accurate records of operation of the system should be maintained by the licensee and should be available for third party assessment. These records should include information on all maintenance activities (including preventive and corrective actions), in-service testing and anomalies observed during operation of the system.

14.11. Incidents and anomalies associated with the computer based system (including difficulties in operation and with the use of maintenance manuals) should be

recorded according to procedures which should be in place for this purpose. The results of the investigations of incidents and the corrective actions undertaken should be stated. Corrective actions that involve modifications of software are subject to a strict change control process as described in Section 15.

14.12. For safety systems, if the system for generation of calibration data has not been developed to the same standard as the computer based system (see para. 14.7), the process for the generation of calibration data should be described and demonstrated to be diverse. The groups performing the required calculations should be identified and procedures should be in place that ensure the maximum diversity. All calculations should be documented and retained for future audit.

15. POST-DELIVERY MODIFICATIONS

15.1. During the operational phase, it is necessary to ensure that the functional safety of the computer based system is maintained both during and after modifications. As mentioned in para. 4.24, adequate change control procedures which deal specifically with plant safety issues should therefore be in place. It is also important that the safety impact of the modifications be analysed by personnel from appropriate disciplines. In this respect, modifications to computer based systems are no different from other plant modifications. However, there are certain specific issues that apply uniquely to computer based systems and their software, and these are addressed in this section.

RECOMMENDATIONS

15.2. The plant designers and operators should ensure that adequate change control procedures are in place, including appropriate procedures and organizational structures for the review and approval of the safety aspects of the modification. These procedures should be in place from the start of the project.

15.3. All modifications should be considered and classified according to their significance for safety. Those of the highest safety significance may need to be submitted to the regulatory authority, in particular those modifications that would alter previously approved operational limits and conditions.

15.4. A procedure should be in place by which experts independent of the designers and developers of the modification should assess the adequacy of the proposed

modification and of its implementation. The changes that should be covered by the change procedures include software changes, hardware changes and tool changes.

15.5. The change procedures should ensure that a safety justification is provided for each modification (see para. 15.3).

15.6. The modification of the computer system during on-line operation, and in particular of its software, should only be allowed if supported by detailed justification, and no equipment should be provided for this purpose. Modification to those parameters that might require variation during the operation of the plant (such as trip settings and calibration constants) should be undertaken using engineered facilities that have been shown to be fit for the purpose. The degree of variability provided by the engineered facility should be limited to the range justified in the plant safety analysis.

15.7. Strict configuration control should be maintained throughout the change process, in particular to resolve any conflicts resulting from modifications being carried out simultaneously. Only those items that have been through the complete change process should be installed in the plant equipment. The change procedures should make provision for those cases where troubleshooting or testing would require temporary changes while the computer system is not on-line.

15.8. The installation of the modified software into any redundant subsystem of diverse subsystems should be planned and made to reduce the effects of the temporary system degradation. The installation of the modified software into the safety system should be phased by subsystem to reduce common cause effects (i.e. one subsystem at a time).

DOCUMENTS

15.9. For each of the proposed changes, the following information should be provided, as appropriate:

— The reason for the modification;
— A functional description of the modification;
— A safety evaluation of the modification which demonstrates that plant safety is not adversely affected by the changes;
— A detailed description of the design modification, with an impact analysis that covers the full extent of the proposed change, including all plant items which may be affected;

— The reports on verification and validation and third party assessment, including a justification of their scope and of the regression analysis;
— A justification of the proposed installation method (see para. 15.8);
— The report on tests performed on the site.

15.10. All modification documents should be dated, numbered and filed in the listing of software modification controls.

15.11. A chronological record should be established and maintained. It should provide details of all modifications and changes, including references to the modification and change request, the impact analysis, the verification and validation of data and results, and all documents affected by modification and change activities.

15.12. The documentation of changes should be available to the regulator, who may wish to approve the change process and the implementation of the changes.

REFERENCES

[1] INTERNATIONAL ATOMIC ENERGY AGENCY, Safety of Nuclear Power Plants: Design, Safety Standards Series No. NS-R-1, IAEA, Vienna (2000).
[2] INTERNATIONAL ATOMIC ENERGY AGENCY, Protection System and Related Features in Nuclear Power Plants, Safety Series No. 50-SG-D3, IAEA, Vienna (1980).
[3] INTERNATIONAL ATOMIC ENERGY AGENCY, Safety Related Instrumentation and Control Systems for Nuclear Power Plants, Safety Series No. 50-SG-D8, IAEA, Vienna (1984).
[4] INTERNATIONAL ATOMIC ENERGY AGENCY, Software Important to Safety in Nuclear Power Plants, Technical Reports Series No. 367, IAEA, Vienna (1994).
[5] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Software for Computers in Safety Systems of Nuclear Power Plants, Standard No. 880, IEC, Geneva (1986).
[6] EUROPEAN COMMISSION, European Nuclear Regulators' Current Requirements and Practices for the Licensing of Safety Critical Software for Nuclear Regulators, Rep. EUR 18158 EN, Office for Official Publications of the European Communities, Luxembourg (1998).
[7] ATOMIC ENERGY CONTROL BOARD, CANADA; DIRECTION DE LA SÛRETÉ DES INSTALLATIONS NUCLÉAIRES, INSTITUT DE PROTECTION ET DE SÛRETÉ NUCLÉAIRE, FRANCE; NUCLEAR INSTALLATIONS INSPECTORATE, UNITED KINGDOM; NUCLEAR REGULATORY COMMISSION, UNITED STATES OF AMERICA, Four Party Regulatory Consensus Report on the Safety Case for Computer-Based Systems in Nuclear Power Plants, HMSO, Norwich (1997).

[8] INTERNATIONAL ATOMIC ENERGY AGENCY, Specification of Requirements for Upgrades Using Digital Instrument and Control Systems, IAEA-TECDOC-1066, Vienna (1999).

[9] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Instrumentation and Control Systems Important for Safety: Classification, Standard No. 61226, IEC, Geneva (1993).

[10] INTERNATIONAL NUCLEAR SAFETY ADVISORY GROUP, Basic Safety Principles for Nuclear Power Plants, Safety Series No. 75-INSAG-3 Rev. 1, INSAG-12, IAEA, Vienna (1999).

[11] INTERNATIONAL ATOMIC ENERGY AGENCY, Application of the Single Failure Criterion, Safety Series No. 50-P-1, IAEA, Vienna (1990).

[12] BRITISH COMPUTER SOCIETY, Guidelines on Good Security Practice, BCS, Swindon (1990).

[13] BRITISH STANDARDS INSTITUTION, Code of Practice for Information Security Management, BS 7799, BSI, London (1995).

[14] INTERNATIONAL ATOMIC ENERGY AGENCY, Quality Assurance for Safety in Nuclear Power Plants and other Nuclear Installations, Code and Safety Guides Q1–Q14, Safety Series No. 50-C/SG-Q, IAEA, Vienna (1996).

[15] INTERNATIONAL ATOMIC ENERGY AGENCY, Manual on Quality Assurance for Computer Software Related to the Safety of Nuclear Power Plants, Technical Reports Series No. 282, IAEA, Vienna (1988).

[16] INTERNATIONAL ATOMIC ENERGY AGENCY, Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control, Technical Reports Series No. 384, IAEA, Vienna (1999).

[17] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Programmed Digital Computers Important to Safety Systems of Nuclear Power Plants, Standard No. 987, IEC, Geneva (1989).

[18] VOGES, U., “Software diversity”, Proc. 9th Annual Conf. on Software Safety, Luxembourg, 1992, Centre for Software Reliability, City Univ., London (1992).

[19] PAVEY, D.J., WINSBORROW, L.A., Demonstrating equivalence of source code and PROM contents, Computer J. **36** (1993) 654.

Annex

USE AND VALIDATION OF PRE-EXISTING SOFTWARE[1]

This annex reproduces in its entirety (with minor modifications) Section 1.3 of Ref. [A–1].

RATIONALE

A–1. The inclusion of pre-existing software (PSW) components into the application software may not only be beneficial for productivity but may also increase the safety of a software system if introduced in a proper way. The benefit stems from the fact that PSW components often have been used in many applications, and their operating experience, when assessable and representative, can be taken into account. Reusable software components may have been developed to suitably high standards in other industries for use in safety critical applications and, therefore, may be reusable in the nuclear industry. Licensees may wish to make use of such software given that the appropriate assessment has been undertaken.

ISSUES INVOLVED

A–2. The functional and non-functional (dependability, performance,...) behaviour of the PSW is often not clearly defined and documented.

A–3. The documentation and the 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.

A–4. As a result of the issues in A–2 and A–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]

A–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.

COMMON POSITION[2]

A–6. The functions which have to be performed by the PSW components shall be clearly identified, and the impact on safety of these functions shall be evaluated.

A–7. The PSW components to be used shall be clearly identified, including their code version(s).

A–8. The interfaces through which the user or other software invokes PSW modules shall be clearly identified and thoroughly validated. Evidence shall be given that no other calling sequence can be exercised, even inadvertently.

A–9. The PSW shall have been developed and shall be maintained according to good software engineering practice and QA standards appropriate to its intended use.

A–10. 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.

A–11. If modifications of PSW components are necessary, the design documentation and the source code of the PSW shall be available.

A–12. 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.

A–13. For acceptance the following actions shall be taken:

A–14. 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;

[2] In the context of this Safety Guide, this term means recommendations.

A–15. 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;

A–16. Perform a compliance analysis of the PSW design against the applicable standards requirements (e.g. [A–2]);

A–17. The PSW functions intended for use shall be validated by testing. The tests may include tests performed by the vendor;

A–18. Ensure that the PSW functions cannot be used by the safety system, by other software or by the users in ways that are different from those which have been specified and tested (if necessary through the implementation of preconditions, locking mechanisms or other protections).

A–19. 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.

A–20. If the available information of the type required by recommendation A–19 above is not sufficient, then an analysis (risk assessment) of the impact on safety of a failure of the PSW shall be performed. Special attention shall be paid to possible side-effects and to failures that may occur at the levels of interfaces between the PSW and the user and/or other software components.

A–21. Errors that are found during the validation of the PSW shall be analysed and taken into account in the acceptance procedure.

RECOMMENDED PRACTICES

A–22. 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 [A–2, E3, Statistical Approaches]).

A–23. Data for the evaluation of the credit which 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 startup;
— 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:

— Date of error, severity;
— Fixes.

Release history should include:

— Date and identification of releases;
— Faults fixed, functional modifications or extensions;
— Pending problems.

These data should be recorded with the identification of the release of the PSW and of the associated configuration.

REFERENCES TO THE ANNEX

[A–1] EUROPEAN COMMISSION, European Nuclear Regulators' Current Requirements and Practices for the Licensing of Safety Critical Software for Nuclear Regulators,

Rep. EUR 18158 EN, Office for Official Publications of the European Communities, Luxembourg (1998).

[A–2] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Software for Computers in Safety Systems of Nuclear Power Plants, Standard No. 880, IEC, Geneva (1986).

GLOSSARY

The following definitions apply for the purposes of the present publication.

computer based system important to safety. Plant system important to safety in which the safety functions of the system are achieved through an embedded computer system.

computer system architecture. The hardware components (processors, memories, input–output devices) of the computer system, their connections, the communication systems and the mapping of the software functions on these components.

computer system integration. The process of integrating the software with the hardware to produce the computer system.

computer system requirements. The functional and non-functional properties of the computer system that are necessary and sufficient to meet the safety requirements for the plant, which have been defined at a higher design level.

dependability. Trustworthiness of the delivered service such that reliance can justifiably be placed on this service. Reliability, availability and safety are attributes of dependability.

firmware. Computer programs and data loaded into a class of memory (such as read-only memory (ROM)) that cannot be dynamically modified by the computer during processing.

functional safety requirements. Safety service or safety function required to be delivered.

implementation. (1) The process of translating a design into hardware components or software components. (2) The result of the process in (1).

non-functional requirements. Property or performance required to be guaranteed.

predeveloped software or pre-existing software. Software used in a computer based system that was not developed under the control of those responsible for

the system development (off-the-shelf software is one type of pre-existing software).

redundancy. Provision of alternative (identical or diverse) structures, systems or components, so that any one can perform the required function regardless of the state of operation or failure of any other.

reverse engineering. The regeneration of a specification from a completed design in order to make a comparison with the original specification to ensure that the two are compatible.

reviewability. Quality of allowing discrepancies with respect to planned results in evaluating software elements or project status to be easily identified.

safety related system. A system important to safety which is not part of a safety system.

safety system. A system important to safety, provided to ensure the safe shutdown of the reactor or the residual heat removal from the core, or to limit the consequences of anticipated operational occurrences and design basis accidents.

software requirements. Statement of what the software component is required to do in order that the system requirements are met when the software is executed on the chosen set of computers with associated peripherals.

system important to safety. A system that is part of a safety group and/or whose malfunction or failure could lead to radiation exposure of the site personnel or members of the public.

system integration. The process of integrating a computer system with other components of a plant system.

system life cycle. All stages through which a system passes from conception to final disposal.

timing. The time limits required for the transformations performed by a software.

traceability. The degree to which a relationship can be established between two products of the development process, especially products having a predecessor–successor relationship to one another.

validation[3]**.** The process of testing and evaluation of the integrated computer system (hardware and software) to ensure compliance with the functional, performance and interface requirements.

verification[3]**.** The process of ensuring that a phase in the system life cycle meets the requirements imposed on it by the previous phase.

voting. Strategy for reducing the likelihood of a system spurious action by voting on output signals (such as two out of three output signals).

[3] For the purposes of this Safety Guide, these definitions of validation and verification apply in the context of the life cycle of a computer based system**.**

CONTRIBUTORS TO DRAFTING AND REVIEW

Asmis, G.J.K.	Atomic Energy Control Board, Canada
Bafurík, J.	Nuclear Regulatory Authority, Slovakia
Beltracchi, L.	Nuclear Regulatory Commission, United States of America
Bouard, J.-P.	Electricité de France, France
Carre, B.	Praxis Critical Systems plc, United Kingdom
Chandra, U.	Bhabha Atomic Research Centre, India
Courtois, P.-J.	AV Nuclear, Belgium
Duong, M.	International Atomic Energy Agency
Fandrich, J.	Siemens A.G., Germany
Faya, A.	Nuclear Safety Commission, Canada
Ficheux, F.	Electricité de France, France
Geerinck, P.	TRACTEBEL S.A., Belgium
Greenberg, R.	Atomic Energy Commission, Israel
Hamar, K.	Atomic Energy Commission, Hungary
Henry, J.Y.	Institut de protection et de sûreté nucléaire, France
Hirose, M.	Nuclear Power Engineering Corporation, Japan
Hohendorf, R.J.	Ontario Hydro, Canada
Hughes, P.	Nuclear Installations Inspectorate, United Kingdom

Karpeta, C.	State Office for Nuclear Safety, Czech Republic
Kersken, M.	Institut für Sicherheitstechnologie GmbH, Germany
Kulig, M.J.	International Atomic Energy Agency
Lawrence, D.	Lawrence Livermore National Laboratory, United States of America
Lee, J.-S.	Atomic Energy Research Institute, Republic of Korea
Mandij, D.	Krško Nuclear Power Plant, Slovenia
Nechanický, M.	Temelin Nuclear Power Plant, Czech Republic
Pachner, J.	International Atomic Energy Agency
Regnier, P.	Institut de protection et de sûreté nucléaire, France
Roca, J.L.	Ente Nacional Regulador Nuclear, Argentina
Saidel, F.	Federal Office for Radiation Protection, Germany
Tanaka, T.	Tokyo Electric Power Company, Japan
Taylor, R.P.	Nuclear Safety Commission, Canada
Vojtech, J.	Nuclear Power Plants Research Institute, Slovakia
Voumard, A.	Federal Nuclear Safety Inspectorate, Switzerland
Wainwright, N.	Nuclear Installations Inspectorate, United Kingdom
Yates, R.L.	Nuclear Installations Inspectorate, United Kingdom
Zambardi, F.	National Agency for Environmental Protection, Italy

ADVISORY BODIES FOR THE ENDORSEMENT OF SAFETY STANDARDS

Nuclear Safety Standards Advisory Committee

Belgium: Govaerts, P. (Chair); *Brazil*: da Silva, A.J.C.; *Canada*: Wigfull, P.; *China*: Lei, Y., Zhao, Y.; *Czech Republic*: Stuller, J.; *Finland*: Salminen, P.; *France*: Saint Raymond, P.; *Germany*: Wendling, R.D., Sengewein, H., Krüger, W.; *India*: Venkat Raj, V.; *Japan*: Tobioka, T.; *Republic of Korea*: Moon, P.S.H.; *Netherlands*: de Munk, P., Versteeg, J.; *Russian Federation*: Baklushin, R.P.; *Sweden*: Viktorsson, C., Jende, E.; *United Kingdom*: Willby, C., Pape, R.P.; *United States of America*: Morris, B.M.; *IAEA*: Lacey, D.J. (Co-ordinator); *OECD Nuclear Energy Agency*: Frescura, G., Royen, J.

Advisory Commission for Safety Standards

Argentina: Beninson, D.; *Australia*: Lokan, K., Burns, P., *Canada*: Bishop, A. (Chair), Duncan, R.M.; *China*: Huang, Q., Zhao, C.; *France*: Lacoste, A.-C., Asty, M.; *Germany*: Hennenhöfer, G., Wendling, R.D.; *Japan*: Sumita, K., Sato, K.; *Republic of Korea*: Lim, Y.K.; *Slovak Republic*: Lipár, M., Misák, J.; *Spain*: Alonso, A., Trueba, P.; *Sweden*: Holm, L.-E.; *Switzerland*: Prêtre, S.; *United Kingdom*: Williams, L.G., Harbison, S.A.; *United States of America*: Travers, W.D., Callan, L.J., Taylor, J.M.; *IAEA*: Karbassioun, A. (Co-ordinator); *International Commission on Radiological Protection*: Valentin, J.; *OECD Nuclear Energy Agency*: Frescura, G.