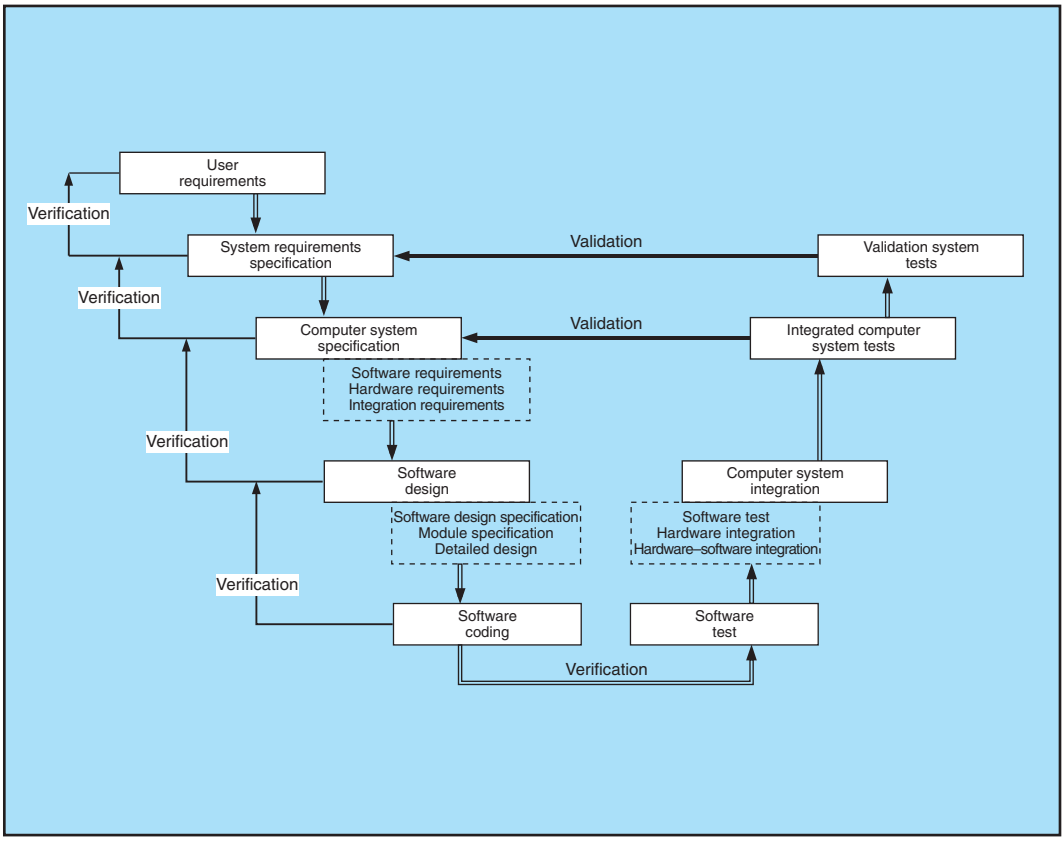

TECHNICAL REPORTS SERIES No. 384

Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control

INTERNATIONAL ATOMIC ENERGY AGENCY, VIENNA, 1999

VERIFICATION AND VALIDATION OF SOFTWARE RELATED TO NUCLEAR POWER PLANT INSTRUMENTATION AND CONTROL

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

AFGHANISTAN
ALBANIA
ALGERIA
ARGENTINA
ARMENIA
AUSTRALIA
AUSTRIA
BANGLADESH
BELARUS
BELGIUM
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
May 1999
STI/DOC/010/384

TECHNICAL REPORTS SERIES No. 384

VERIFICATION AND VALIDATION OF SOFTWARE RELATED TO NUCLEAR POWER PLANT INSTRUMENTATION AND CONTROL

INTERNATIONAL ATOMIC ENERGY AGENCY
VIENNA, 1999

VIC Library Cataloguing in Publication Data

Verification and validation of software related to nuclear power plant instrumentation and control. — Vienna : International Atomic Energy Agency, 1999.

p. ; 24 cm. — (Technical reports series, ISSN 0074–1914); no. 384)

STI/DOC/010/384

ISBN 92–0–100799–X

Includes bibliographical references.

1. Computer software—Verification. 2. Nuclear power plants—Instruments. I. International Atomic Energy Agency. II. Series: Technical reports series (International Atomic Energy Agency); 384.

VICL 99–00218

FOREWORD

The use of software based equipment and systems in nuclear power plants has been growing in recent years, both as a result of the need to replace obsolete analog equipment and as a means of improving and ensuring satisfactory levels of plant availability and safety. Software can now be found in safety applications (e.g. protection systems) and safety related applications (e.g. control systems), as well as in applications not important to safety (e.g. data logging). The use of computers in nuclear power plants is not new. Computerized data collection and display systems have been in operation for many years, and computers, often in the form of programmable logic controllers, have been used widely in control applications. Computer based reactor protection and safety system actuation systems are less common but have been introduced in many countries. These include Canada, France, the Republic of Korea, Sweden, the United Kingdom and the United States of America. This list is expected to lengthen in the near future to include countries such as the Czech Republic, Germany, Hungary and Japan. It is computer based protection systems and their safety significance that have caused interest and concern regarding the dependability of software and the justification of its integrity.

The nuclear industry does not face these issues alone. There have always been significant concerns about the introduction of software in industry and this applies particularly to safety applications. The nuclear industry is no different from other sectors except that its safety requirements are among the most demanding. The processes of verification and validation, which are part of the development process, provide much of the evidence that the systems are suitable for use.

This publication has been produced in response to a recommendation of the IAEA International Working Group on Nuclear Power Plant Control and Instrumentation. The report has the objectives of providing practical guidance on the methods available for verification of the software and validation of computer based systems, and on how and when these methods can be effectively applied. The intended readership consists of those who are in any way involved with the development, implementation, maintenance and use of software and computer based instrumentation and control systems in nuclear power plants. The report is intended to be of use to designers, software producers, reviewers, verification and validation teams, assessors, plant operators and licensers of computer based systems.

Major contributions to the report were made by O. Andersson (Sweden), A.R. Ets (USA), E. Leret (France) and D.N. Wall (UK), who compiled the document from contributions provided by members of the working team. The document was finalized by D.N. Wall and D. Welbourne (UK), together with A. Kossilov and V. Neboyan of the IAEA Division of Nuclear Power.

EDITORIAL NOTE

Although great care has been taken to maintain the accuracy of information contained in this publication, neither the IAEA nor its Member States assume any responsibility for consequences which may arise from its use.

The mention of names of specific companies or products (whether or not indicated as registered) does not imply any intention to infringe proprietary rights, nor should it be construed as an endorsement or recommendation on the part of the IAEA.

CONTENTS

1. INTRODUCTION

Computer equipment and information processing technology have been quickly taken up and exploited by instrumentation and control (I&C) engineers in the process industries to improve and ensure satisfactory levels of plant performance. Systems using computers and related technology were initially installed for functions which were needed for plant operation, but which were not important to safety. Computers were widely installed in systems for data collection, information display and data logging, and also in systems providing important control functions. They were often easily accepted at plants by the users. Their introduction for protection purposes rapidly followed their use in the aforementioned roles in most industrial sectors.

The introduction of software to perform critical functions in industry has always caused concern. The introduction of software based systems has led to well documented difficulties in several industrial sectors, for example the controversy associated with aircraft with multiplexed (fly by wire) controls. The nuclear industry is no different from any other sector except that the safety requirements are among the most demanding. The introduction of computer based systems in the nuclear sector has not been easy, since new methods and criteria have been needed to assess and judge their safety and integrity. For example, it has resulted in the introduction of a special procedure by the licensing body in the United Kingdom, the Nuclear Installations Inspectorate, and software was declared by the Nuclear Regulatory Commission in the United States of America as posing an "unresolved safety issue" (at the time of preparation of this publication).

Nuclear plants followed the trend of computerization of functions. This was initially driven by economic pressures such as the cost of the equipment required for data collection. During this time computer based systems were restricted to applications with low safety significance. This was done very deliberately, to avoid problems of demonstrating software integrity. This restriction was steadily eroded as pressure for enhanced safety, functionality and reliability increased. Computers started to appear with safety roles, for example the core calculator in some BWRs in the USA. This position has changed still further in the last few years with computer based systems being introduced with a primary role in nuclear safety. There are already systems for reactor protection in use in:

— Canada: the Point Lepreau, Darlington and Gentilly 2 CANDU plants (reactor protection system);
— France: the 1300 MW(e) series of PWR plants (SPIN 1 reactor protection system);
— The Republic of Korea: the Wolsong 1 CANDU power unit (reactor protection system);
— Sweden: the Forsmark BWR plant (neutron flux measurement and protection);

— The UK: the Dungeness B AGR (single channel temperature trip system) and Sizewell B PWR (reactor protection system);
— The USA: BWR plants (core calculator) and a PWR plant (Eagle 21 reactor protection system).

This list is expected to increase in the near future to include other countries and reactor types as the use of computer technology develops. Countries that are expected to install computer systems in the near term include the Czech Republic, Germany, Hungary and Japan.

Pressure for change also arises from the need to replace old analog systems as they become obsolete and the desire of plant owners to improve the economic performance of their plants. A significant number of computer based systems important to safety are currently being developed. These systems are expected in new reactors and as backfits on old reactors as part of I&C upgrade programmes.

This publication examines software verification and validation as key elements of the demonstration that a particular software based system is fit for service. This section sets out the objective and scope of the publication and identifies the need for and roles of the verification and validation processes. This publication, like many others, will often refer to verification and validation as 'V&V', as if they were a single process. Despite this common usage, the reader is reminded that verification and validation are two quite different processes with two quite different objectives, as shown by their definitions in Section 1.5.

1.1. OBJECTIVE AND SCOPE

The objective of this publication is to provide information on how effective V&V of computer based I&C systems can be achieved and on methods available for V&V. This is restricted to systems important to the safety and operation of nuclear power plants (NPPs). The publication will focus on software. It will not consider V&V of hardware necessary to implement the system specification, although the need for this is recognized. The V&V of the hardware should be done following established practice and recognized standards such as IEC 987, issued by the International Electrotechnical Commission [1].

The information provided is intended to be consistent with, and to supplement, the requirements and recommendations of the IAEA's Code on Quality Assurance for Safety in Nuclear Power Plants and other Nuclear Installations [2] and its Manual on Quality Assurance for Computer Software Related to the Safety of Nuclear Power Plants [3]. The guidance is also consistent with the requirements of international and other key standards, including IEC 880 [4], IEC 987 [1] and the Institute of Electrical and Electronics Engineers standard IEEE Std 1012 [5].

This report uses the categories A, B and C, ranking importance to safety of functions, systems and equipment, as defined in IEC 1226 [6]. It assumes the existence of a system requirements specification that is complete, correct, consistent and unambiguous at the beginning of the development process. It assumes also that the software is developed following a conventional life cycle, consisting of a series of defined, planned steps. The means of verifying the inputs to and outputs from each of these steps are then discussed. The methods for performing validation of the product are discussed similarly. The relationship to licensing is discussed and conclusions are given.

Techniques for V&V are reviewed in Appendix I. The single failure criterion is discussed in Appendix II. The experience of several countries is summarized in Appendix III. Tables show the inputs, tasks and outputs of each V&V step, with recommendations on techniques, for each category of safety importance.

The publication is intended for use as a reference when establishing the means of verifying and validating software. The information is not prescriptive in nature, by contrast to the requirements of a standard. It is expected that this publication will be used to produce approaches to V&V processes. The approaches should be consistent with the production method being used and make the best use of staff and tools available, while taking due account of the prevailing regulatory conditions. It is not intended to identify a set of defined V&V processes which should be completed in order for the system to be licensable.

This publication was developed primarily to assist suppliers, users and regulators in the development, installation, acceptance and licensing of computer based systems important to safety. The techniques can and ought to be applied to plant control and other computer based systems. The user of the publication may select and apply those recommendations that are applicable to the development route chosen for a given project, in the context of the safety role of the product. Justification of the approach adopted would normally be required by the quality plan of the development process.

The key references which form the basis of this publication are Refs [3, 4, 6–8].

1.2. NEED FOR VERIFICATION AND VALIDATION

There is a possibility that the designers and implementers of a computer based system will make errors, and that the errors will result in undetected faults in the system. The system may then be installed with faults in it. This could clearly result in dangerous failures. The detection of errors is therefore important of itself. Owing partly to anecdotal stories and partly to some poor experience, the public perception of software systems is that they are likely to fail. Public confidence requires the safety

regulator to have very high assurance, supported by clear evidence, that errors have been detected suitably.

The failures that can result from these faults make it necessary to apply active measures in a systematic manner to find and correct errors. Current practice for computer based systems places a heavy dependence on V&V to detect errors. The V&V processes provide assurance that the specified functional and reliability requirements of the system are met.

Hardware development and V&V processes are generally considered to be sufficiently mature to detect systematic errors effectively. If the systematic error rate is very low, the failure rate of the final system can be bounded by the random failure rate of the hardware. This must be questioned if the complexity of the hardware increases or when novel technology is used. For example, application specific integrated circuits, and similar devices, involve many thousands of interconnected logic gates on a single chip. The reliability of such devices could be vulnerable to unrevealed errors in the design or production processes. The reliability could then include a significant systematic fault probability, and would not be limited by the random failure probability alone.

The present approach to software reliability is similar to that described above. Software faults are due to human error, and the resulting failures are systematic in character. The approach concentrates on methods of demonstrating that no errors were made in implementing the requirements. The aim is to show with high confidence that very few software faults exist, such that the systematic failure rate is acceptably low. If this is shown, then the assumption is made that the system reliability is limited by the hardware random failure rate. The justification of this assumption places a major dependence on the integrity of the development and V&V processes. These processes provide the evidence to enable the assumption to be made and justified. They therefore provide a route to qualification of the system for an application important to safety.

To be effective, V&V should be carried out in conjunction with a structured software development process. The V&V must consider the context of the system application. This publication considers the production of new software following the software life cycle given in IEC 880, but recognizes that the new software approach may not always be applicable. Existing, predeveloped software may be reused to implement new systems [4]. Indeed, software reuse is expected to become a frequent feature of new systems, and to bring reliability and cost advantages when it is correctly adopted.

When developing new software, faults can be introduced at every stage of the development process, from specification to system integration. The first element of verification is to find and correct errors as early as possible in the life cycle. This is done by performing verification when the development team considers that it has successfully completed each phase. The second element of verification is to provide

assurance that all software faults which could degrade system performance have been removed. In addition, for safety systems, each verification step should identify that the system requirements are satisfied.

Validation provides demonstration and assurance that the software and the system meet all specified requirements. Validation thus provides checks of system functionality against system requirements. It aims to show that all faults which could degrade system performance have been removed before the system is operated.

To be successful, the V&V processes must fulfil these functions. This requires an orderly, planned approach to V&V. Successful V&V will result in the reduction of the number of software faults to a tolerable minimum. Therefore, the number of failures in service should be acceptably low. Particular attention must be given to the output products of the V&V processes to ensure that they can be used by others, such as a reviewer or a licenser. There must be clear documentary evidence to demonstrate to an independent body that there is high assurance that the in-service performance will be acceptable.

1.3. OUTLINE OF THE VERIFICATION AND VALIDATION PROCESS IN THE CONTEXT OF THIS PUBLICATION

Verification and validation are part of the development process and are contrasted in this report with the quality assurance (QA) process. Verification and validation are means by which the product is checked, and by which its performance is demonstrated and assured to be a correct interpretation of the requirements. A continuous process of V&V must be actively applied throughout the software development cycle. V&V includes a strong element of checking, and leads to remedial action. By contrast, QA ensures that a suitable management process has been specified, that the development process conforms to the requirements of basic standards and that the process specified is followed. Applicable standards may be ISO 9000-3, issued by the International Organization for Standardization, and IEEE Std 730 [9, 10]. Quality control (QC) is performed to check that the specified software development methods and process in the QA plan have been correctly followed.

1.3.1. Verification of original requirements

A correct form of the system requirements specification is needed to start the development and V&V. Experience indicates that some requirement errors will exist. All system design documentation should be consistent with the stated requirements. If the requirements specification is incorrect, subsequent parts of the development may contain faults that verification alone may be unable to identify.

A safety system requirements specification will have been produced by reference to the reactor safety analysis during the plant design and by consideration of plant and reactor performance limits. The preparation of the requirements for closed loop control systems may involve extensive modelling of the reactor and plant dynamics to derive the algorithms for control and to demonstrate their stability. The preparation of the requirements for open loop control of switchgear, valves and motors will involve identification and grouping of the logic functions of each control switch or control function. These specifications should provide sufficient detail to allow the development process to take place without any risk of misinterpretation.

The computer based implementation of these functional requirements will therefore have involved system design processes, which are normally subject to review and design control methods. This implies that the requirements specification has been verified against the safety and performance requirements of the plant. These processes are very important to the system integrity, but outside the scope of this publication.

1.3.2. Verification of requirements specification

The requirements specification may have two types of error: requirements incorrect of themselves but correctly implemented and errors of implementation of correct requirements. A well designed system acceptance test should find faults arising from errors in the functional requirements, i.e. the validation tests for acceptance should not be based solely on the written requirements specification but should attempt to test from first principles.

If the final product has inconsistencies with or omissions from the system requirements specification, the position is quite different, and verification can detect the errors. The use of requirement trace methods in development and particularly in verification should, if correctly established and followed, identify inconsistencies and omissions. Verification should also find and account for necessary additional functions, needed for engineering and maintenance reasons, which are not defined in the system requirements specification.

The verified requirements specification can then serve as the initial input to the V&V processes discussed here. It is recognized that in practice this is not always the case, and the requirements may be under some development themselves. Consequently, there is a need for a procedure to ensure the resolution of any problems identified with the system requirements specification, particularly as experience shows that this is the source of many errors.

1.3.3. V&V in development life cycle

For new software this publication specifically refers to the V&V of the development process phases of IEC 880 [4], described in detail in Section 3.1. These

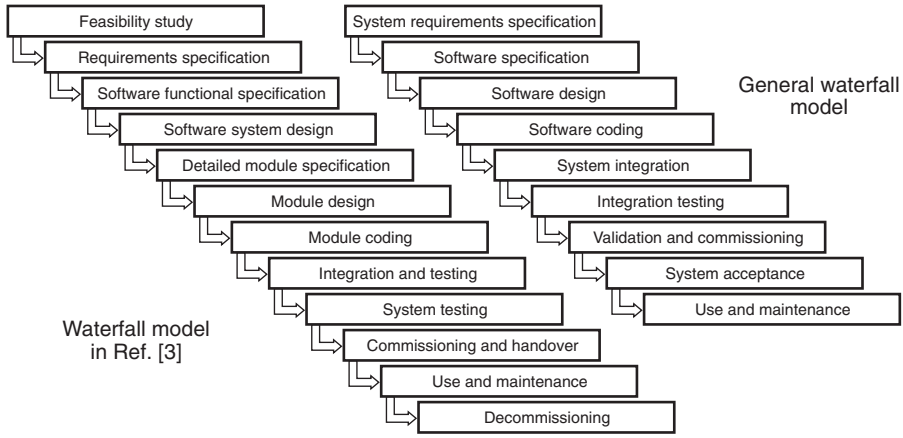

FIG. 1. Waterfall software development model.

phases are those of the traditional waterfall model, two versions of which are shown in Fig. 1. This provides a framework for the descriptions generated in this publication. Other development life cycles may be used. The reference model phases include:

— System requirements specification verification;
— Computer system specification verification;
— Software design verification;
— Code verification;
— Computer system integration verification;
— Integrated computer system test verification;
— Validation and commissioning test verification;
— System handover and acceptance verification;
— Use and maintenance verification.

However, the use of the waterfall model and these phases should not be interpreted in any way as implying a limit on the use of the information and techniques described here. Where a different software life cycle is used, the appropriate information can be identified by establishing the equivalent mapping onto the phases of the development model and applying the information relating to that phase.

The V&V steps listed above are product based, referring to the inputs to and outputs from phases. The information in this publication is also directed to the preparation of:

— V&V plans;
— Test plans.

These plans are important components of the assurance process. Unless the plans are formulated correctly and completely, the effectiveness of the V&V activities is reduced.

The framework developed for new software has also been used as a basis for describing how existing software could be verified and validated. This includes accessible, proprietary and configurable software. These types are discussed later.

The input to, subject of and product of V&V are primarily documentation. The input includes the source code. The importance to effective V&V processes of the completeness and correctness of documentation cannot be overstressed. Clearly, completeness does not mean sheer volume. The information elements identified in this publication for the V&V of the various phases must be present. The planner of the V&V programme must determine the presentation form of this information so that it aids the development and V&V activities. When it is not present in the documents, the planner should require the information from the developers in a suitable document, to allow the V&V to be achieved.

V&V plans should be an integral part of the project development and production plans, and should be produced at the same time. The verification procedures to be used at the end of each phase must be adapted to the software development model being used. They must consider the working environment of the development organization. Important planning concerns include:

— Identifying the teams involved in the development, verification, validation and other qualification processes;
— Determining the interfaces between the V&V team, the development team and the QA team;
— Defining the forms of communication between the teams;
— Organizing the V&V team;
— Declaring the components to be supplied and their expected content and format;
— Preserving independence of the teams;
— Identifying the standards to be applied;
— Defining the acceptance or success criteria;
— Establishing the time schedule.

1.4. ORGANIZATION AND MANAGEMENT OF THE VERIFICATION AND VALIDATION PROCESS

V&V activities should be planned. The management plan should include the software V&V plan and its subsequent updates. V&V management planning should include the listing and collection of applicable standards, procedures and conventions that guide the verification process.

It is common practice in the development of category A systems (generally equivalent to safety systems; see the discussion in Section 2 and IEC 1226 [6]) to establish a single independent team to conduct both the verification and validation activities. This publication assumes that there is such a team. However, this should not prevent the separation of the two activities and their conduct by separate teams. For category B and C systems (generally the control and information display systems; see Section 2 and Ref. [6]), the recommendations provided in this publication do not exclude the possibility of staff belonging to the development team also carrying out V&V activities. Self-verification is not advised.

Effective V&V requires close co-ordination of the activities of the V&V team and the development organization, and therefore good management of the V&V effort is an essential part of the development of reliable software. Elements of V&V management, and activities which must be considered, include the following:

— For category A systems, a V&V team should be established which is independent of the development team. Independence is usually ensured by having different line management for the V&V and development teams.
— The V&V team prepares a formal software V&V plan. The plan guides the application of V&V to the software products, to achieve the required quality standards, and is the basis for each stage of V&V work. The plan should conform to the national standards of the developer or end user. There should be agreement between the developer and user about the content and format of the plan. The level of detail of the plan should be consistent with the category of the software.
— The software V&V plan should clearly define the responsibility, authority and interrelation of all staff who manage and perform V&V activities. These are often clear for category A software. Care is recommended in the case of category B and C software, since a lack of independence of the teams can lead to loss of integrity of the process.
— The software V&V plan should identify for each phase of the development process the input documents that are expected from the development team for verification. Fulfilling these document requirements will indicate good V&V practice and confirm that the software development plan is being followed.
— The V&V team should select methods and use them singly or in combination to give the most appropriate verification programme for each individual project.
— The V&V administrative procedures must be clearly set down so that any problems arising during the process can be resolved in a consistent manner to meet the stated requirements. In particular, the procedures used for anomaly reporting and resolution must be clearly identified. These procedures can be based on existing QA/QC standards, conventions and practices used by the development team. Policy statements with regard to control, deviation and

arbitration of disputes between the V&V team and the development team must be clearly defined.

1.5. TERMINOLOGY

Definitions of some terms used in this publication are given below, to establish a common reference framework. The definitions may be different from those used in standards and other documents referenced in this publication. Owing to the special difficulties associated with definition of the terms 'error', 'fault' and 'failure', the following is quoted from the draft of the first supplement to IEC 880 [11]:

> "If a person or process makes an *error* in producing something, this will result in a *fault* in the product. When the product is used, it may be satisfactory, or it may *fail*, if the fault is not corrected. If the use challenges the fault, the product will fail if no other defence prevents the failure. A *failure* is due to both a fault and a challenge, with no other defence operating. For software, a challenge to a fault is provided by a *signal trajectory*.

error. The result of an incorrect human action or process that produces an unintended result.

failure. A software failure is the result of a signal trajectory which causes code or data with a software fault to be used. A system failure occurs when the system output is not that required for a given signal trajectory. A system failure may be the result of a hardware design error, a hardware fault or a software fault and the associated signal trajectory that results in failure.

fault. A software fault is the result of an error in the code or data in a computer such that the performance would not be as specified if the software used that section of code or data. A hardware fault is the result of an error of design or of a failure of a component or a service to the system.

(Note: A system failure does not necessarily result from a fault, if the conditions experienced by the system do not challenge the fault. Fault tolerant systems are designed to operate with faults present, to give high reliability.)

quality assurance. A programme that identifies to all concerned a basis for the control of all activities affecting quality, monitors the performance of these activities in accordance with the defined and documented procedures, and ensures that the specified quality is achieved.

review. A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers or other interested parties for comment or approval [12].

signal trajectory. A set of time histories of a group of signals, or equipment conditions, which result in a particular output state of the system.

validation. The testing and evaluation of the integrated computer system (hardware and software) to ensure compliance with the functional, performance and interface requirements [4].

verification. The process of determining whether or not the product of each phase of the digital computer system development process fulfils all the requirements imposed by the previous phase [4, 7].

2. SAFETY CLASSIFICATION AND TYPES OF SOFTWARE

This section discusses ways to classify computer based systems according to their safety duties. A classification of software types is proposed to structure the discussion of the different verification requirements that arise when producing a new system. The section also considers the issues associated with software reuse, the treatment of information and documentation from existing software products, and tool qualification.

2.1. CLASSIFICATION OF SYSTEMS

The IAEA Code 50-C-D establishes the concept of classification of NPP systems according to their importance to safety, and gives examples of the classification of the major systems of several types of NPP [13]. Safety Guides 50-SG-D3 and 50-SG-D8 apply classification to I&C systems [14, 15]. These guides establish the distinction between the safety system and safety related systems, which together form the systems important to safety. The safety system consists of the systems provided to ensure the safe shutdown of the reactor and heat removal from the core, and to limit the consequences of anticipated operational occurrences and accident conditions. Safety related systems are systems important to safety that are not part of the safety system.

The principles of IAEA classification have been interpreted by the IEC in IEC 1226, which defines a method of safety categorization. The standard identifies

categories A, B and C for functions, systems and equipment of I&C systems that are important to safety [6]. The definitions of these categories are simplified as follows:

— Category A is assigned to functions, systems and equipment (FSE) which have a principal role in the achievement or maintenance of NPP safety (IEC 1226, section 5.2.1);
— Category B is assigned to FSE which have a supporting safety role to systems of category A (IEC 1226, section 5.2.2);
— Category C is assigned to FSE which have an auxiliary or indirect role in the achievement or maintenance of NPP safety (IEC 1226, section 5.2.3).

The remaining FSE are assigned to be 'unclassified' (U). Annex A of IEC 1226 gives examples of typical functions and systems in each category.

This publication uses the categories of IEC 1226 in grading the recommendations for V&V according to the importance to safety of the function and system concerned. Individual countries have other methods of identification and categorization of functions, systems, equipment and items important to safety. These are defined through national and other standards. This publication must be interpreted in such cases in terms of the classification criteria of the standard or the method of classification concerned.

Safety does not drive the requirement for software integrity alone. The power plant may place a higher requirement on the reliability of the computer system than that imposed solely by safety considerations. This may be due to availability requirements, the importance of the system to plant operation or the novelty of the design of the system. The utility may then require the software to be treated as if it belonged to a high safety category, and to be subject to the more rigorous V&V needed for a high level of assurance of performance and reliability. A utility may therefore decide to treat a computer system of low importance to safety as if it belonged to a higher safety category.

2.2. TYPES OF SOFTWARE

A number of different types of software components are identified in order to discuss the development life cycle and associate the different means of V&V with them. The basis of this classification by type is the convenience of grouping software for which similar processes are followed.

The two main different types are new and existing software, which may be further described as:

(a) *New software:* all software written specifically for the application;

(b) *Existing accessible software:* typically software from a similar application that is to be reused and for which all the documentation is available;
(c) *Existing proprietary software:* typically a commercial product or software from another application that meets all or some of the current application requirements but for which little documentation is available;
(d) *Configurable software:* typically software that already exists but is configured for the specific application using data or an application specific input language.

New software is produced for a specific application and is unique to that application, when first created. The development will use standard products such as editors, compilers and support tools. Existing software is already in use and consists of the necessary basic system software plus application software and configuration data. The application software in turn might be additional software or a modified version of that used in another existing application.

There are advantages and disadvantages associated with both of these types of software. New software can be developed in a manner that is fully compliant with current nuclear standards and appropriate standards specific to software. The development process should follow IEC 880 and will generally require a larger amount of qualified resources and demand more V&V activity than is needed for use of existing software [4]. Some cost reduction may be possible; for example, the coding standard adopted could be selected to allow the maximum amount of tool support to be employed. The reuse of existing software described in (b), (c) and (d) above has the advantage that much of the software should be immediately available. Cost will only be incurred in making the necessary application changes, in preparing the configuration data and in the associated V&V.

However, the level of assurance in the integrity of existing software will depend on the status and availability of documentation and hence on the confidence which can be gained in the original process. Verification and audit must ensure that correct processes were followed. Verification must also concentrate on ensuring that the processes or any modifications which were made have not had any unintended effect which compromised the software integrity. Other issues to be considered include the extent of the experience base, the level of support available from the supplier, the degree of similarity of the intended new use to the previous use and the safety classification of the system. The user must still consider the future maintenance of the system and possible changes in its operational role.

There are a number of specific issues associated with configurable software. Many new software systems are designed to be configurable. The intention of this approach is to minimize the amount of new software required to produce the new application. Generally, software development requires higher skills and greater effort than data definition. The major part of the development work for the

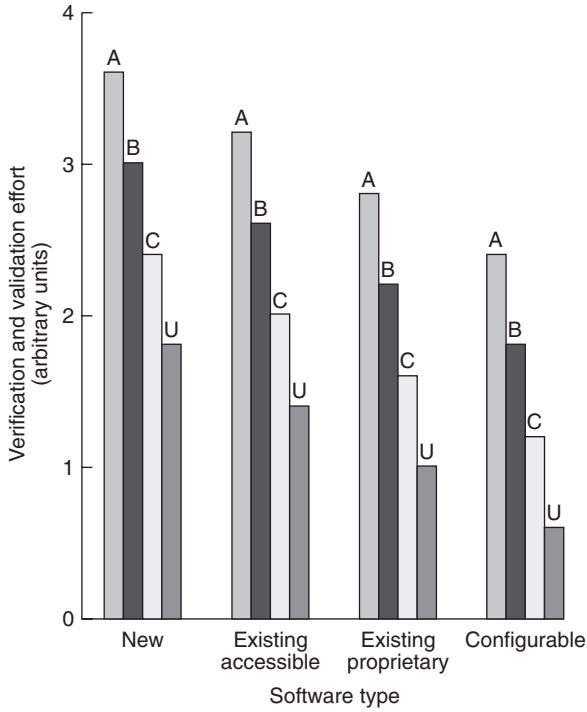

FIG. 2. Desired relationship of safety category, software type and V&V effort.

application is then the identification, verification and validation of the configuration information.

In its simplest form the new application might be achieved by input of new configuration data alone. For example, the process would identify each signal by reference number, title and range, and give the warning and trip values. However, the configuration could be more complex. For example, a programmable logic controller (PLC) might be configured using a ladder diagram, and in this case another issue arises — is this configuration information or programming in an application specific language? A similar situation arises for systems with newly generated graphical interfaces or visual display unit (VDU) displays. The display screens may include diagrams of plant layout, showing input signals, control functions and output signals. The verification of the displayed information and layouts may well require more than simple data checking.

The tools and support environment used for the development and demonstration processes of the application will require some demonstration of acceptability. This is becoming of far greater importance as the reliance on tools to perform repetitive checking functions increases.

Current developments in software engineering are directed at reducing the cost of implementing new systems. This applies both to costs of development and to costs of V&V. The target would be to achieve a relation between software category, software type and cost of V&V such as is shown in Fig. 2 (using arbitrary units of effort).

2.2.1. New software

In this publication, the development of new software will be discussed following the life cycle of IEC 880. This is shown in Fig. 3. The discussion assumes that all the V&V activities described in sections 4.1 and 5.1 of IEC 880 are followed [4]. In every case it is assumed that all the documentation defined is produced. The documentation related to the IEC life cycle is shown in Fig. 4. The life cycle and documentation for new software followed in this publication are shown in Fig. 5.

The route of new software production has the advantage that control can be exercised at the outset, to ensure that the processes adopted allow for suitable V&V. This can greatly reduce the workload and cost of development.

2.2.2. Existing accessible software

It is normally desirable to use existing software to minimize cost and to take advantage of significant operating experience. This gives confidence that the system will function correctly. In some cases the software may be unchanged, for example when devices such as sensors and actuators use microprocessors which include software. The adoption of such software implies that the software has been developed, verified, validated and preferably licensed, as well as operated in a similar application or in a plant design which is being repeated.

For software to be classed as existing and accessible, the source code and all necessary documents produced during the development should be available for evaluation and validation against the application requirements. Operating experience of the software may be used for validation, although the claims must be made cautiously and must be justified (Section 2.2.3). The availability of the design and code documents allows additional analysis to be undertaken if there is concern regarding the integrity. For example, additional static analysis of code would be possible.

The life cycle derived from IEC 880 for this type of software is parallel to that for new software, as shown in Fig. 6 [4]. For modules that have to be modified, the full development life cycle of Fig. 5 within the total system life cycle of Fig. 6 would be appropriate. It may be possible to relax this if changes are only to data on the organization and structure of the code. Such data changes may best be verified using the approach outlined for configurable software.

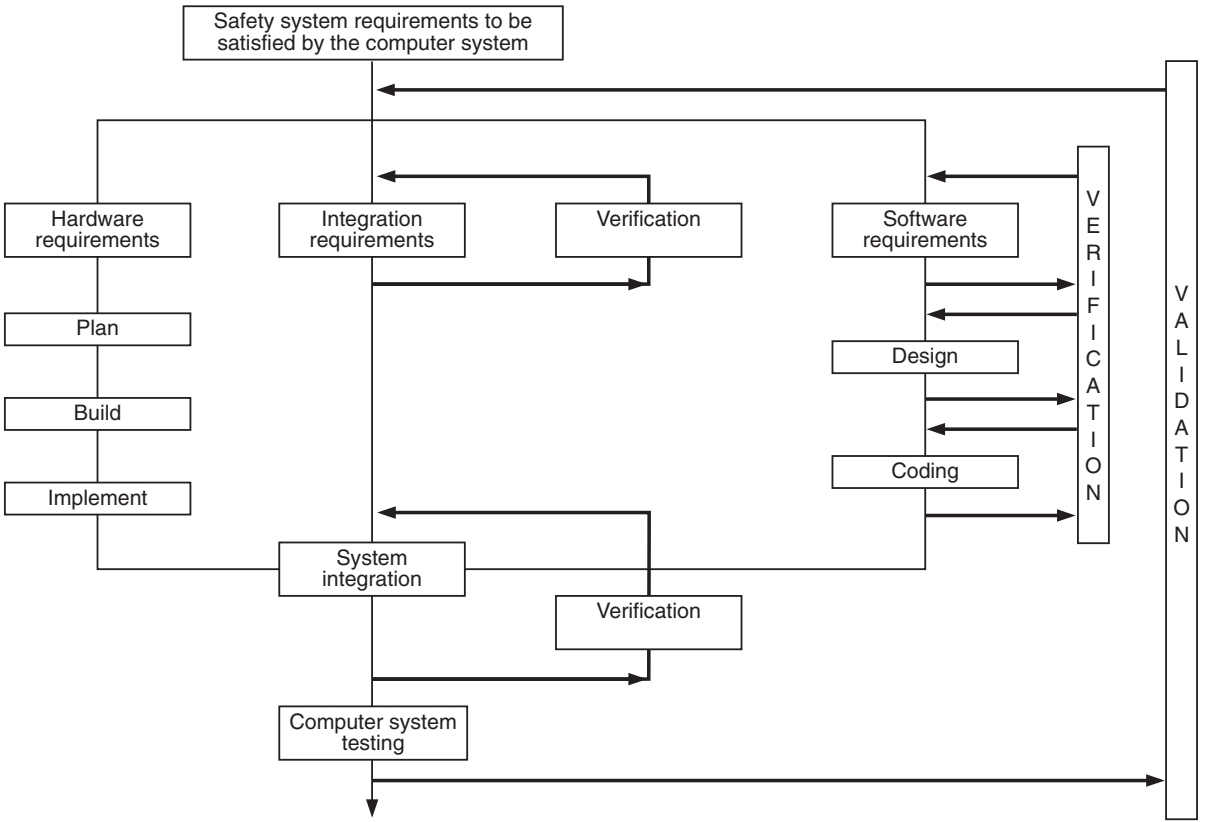

FIG. 3. IEC 880 development life cycle.

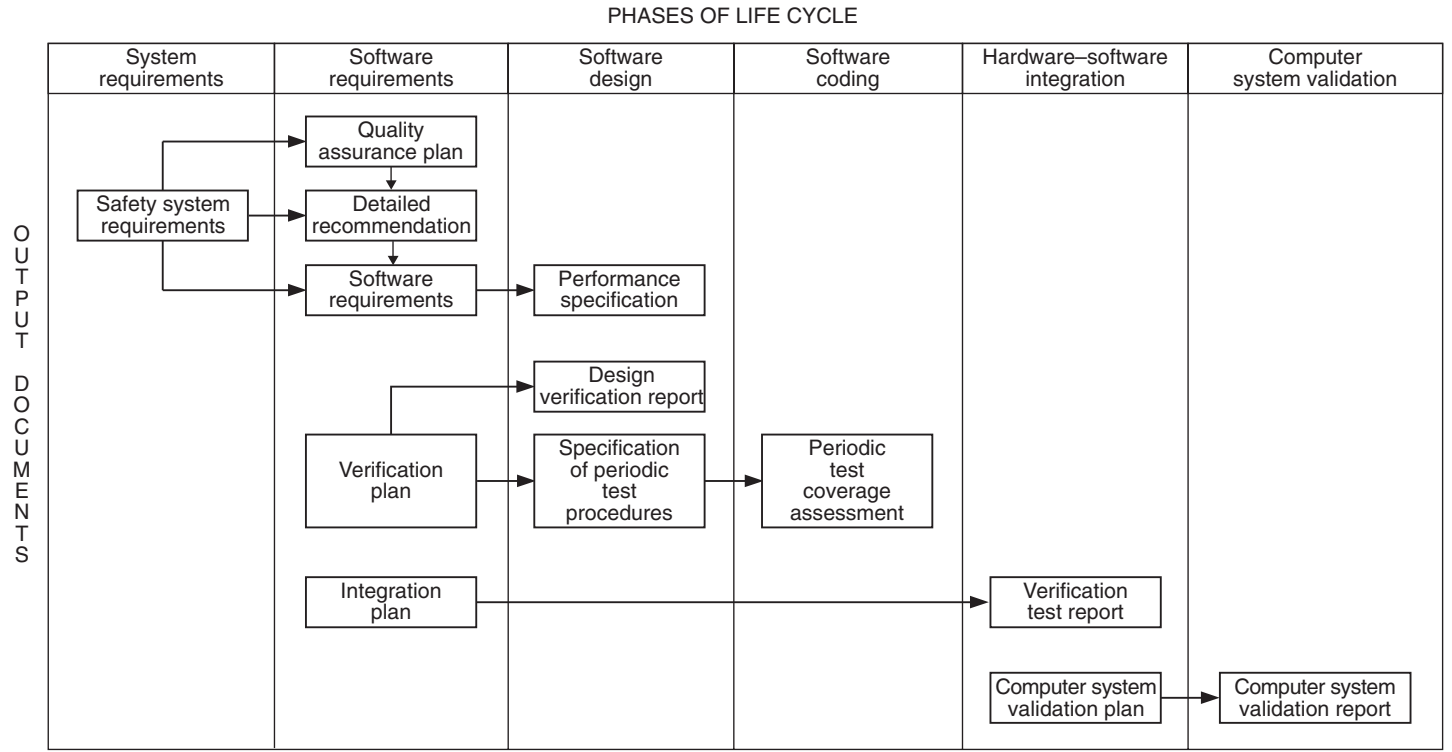

FIG. 4. Software development life cycle documentation based on IEC 880.

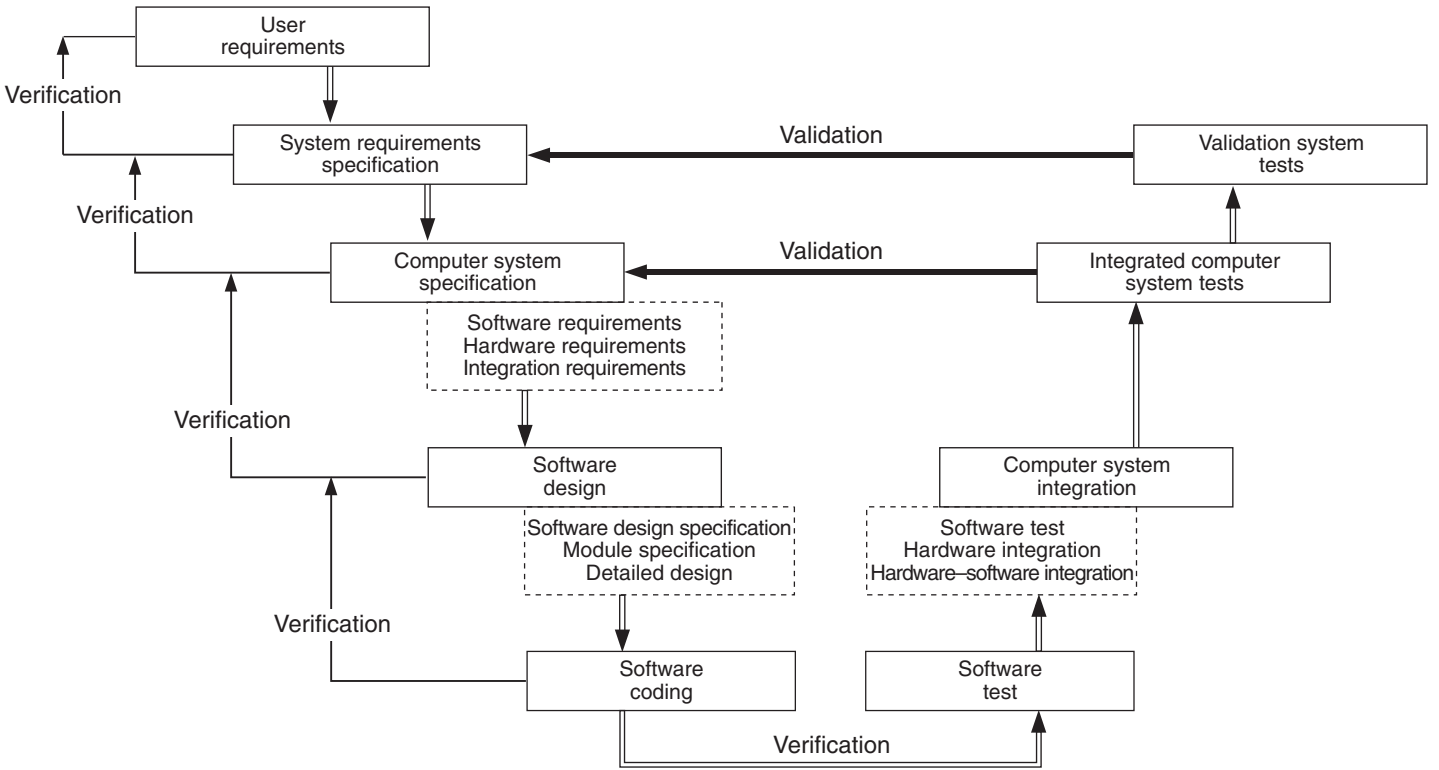

FIG. 5. Life cycle and documentation for new software.

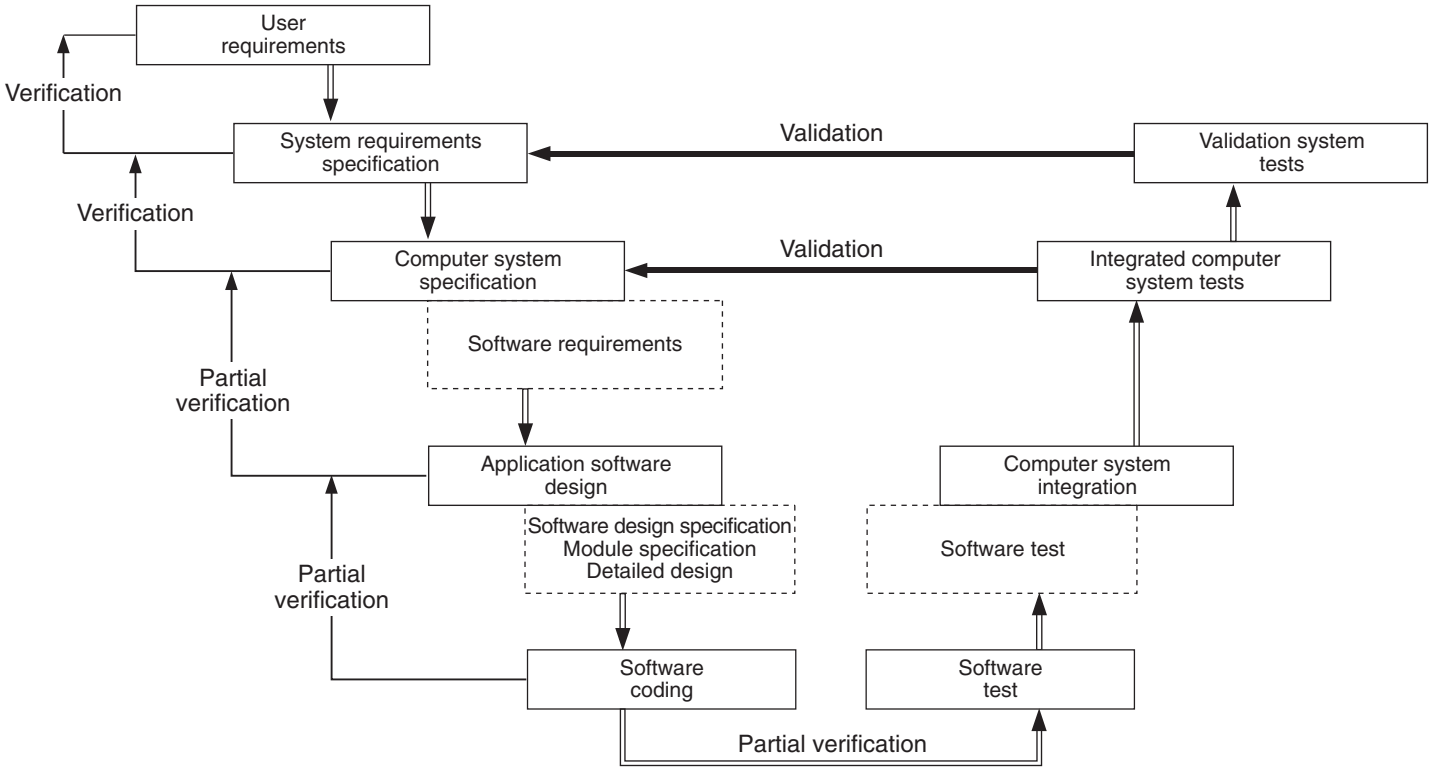

FIG. 6. Life cycle and documentation for modification to existing accessible software.

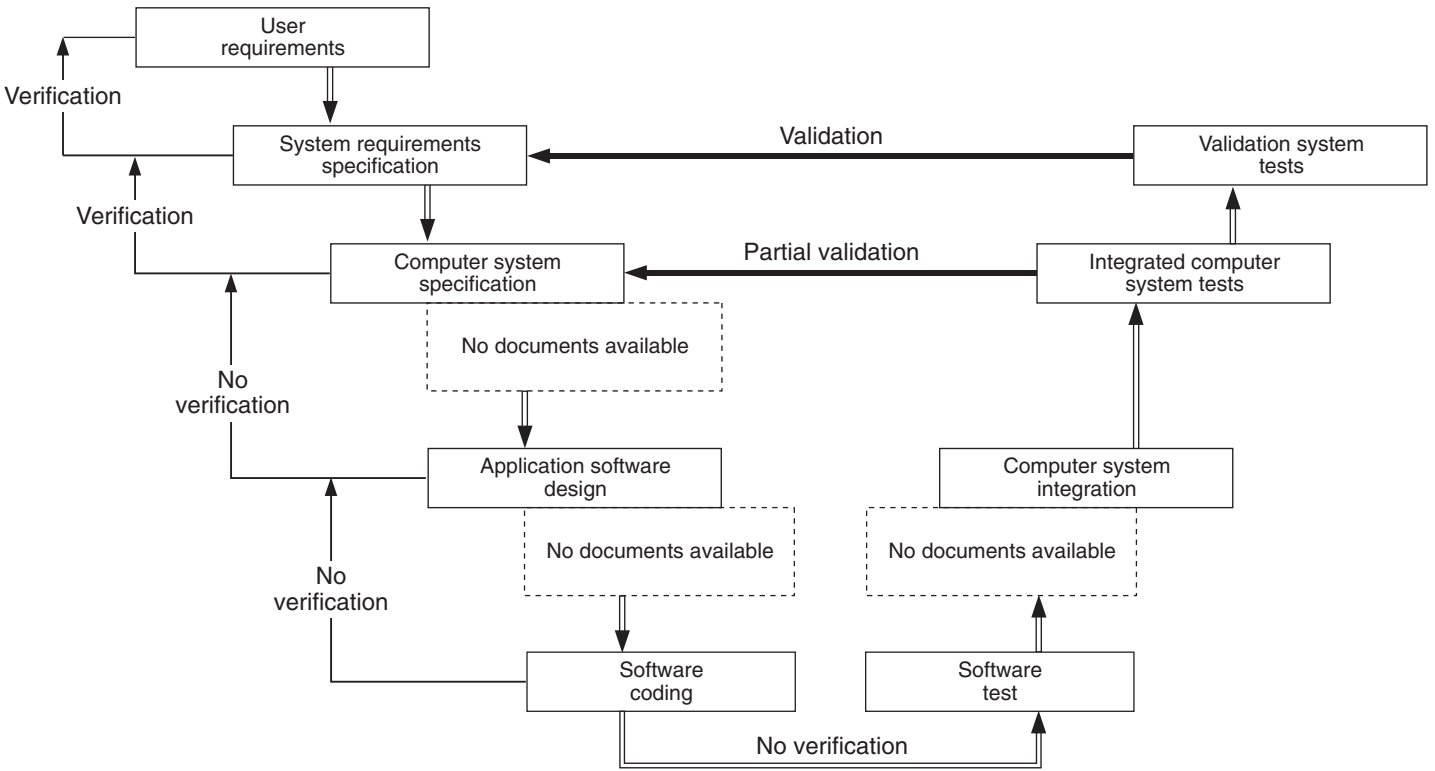

FIG. 7. Life cycle and documentation for existing proprietary software.

2.2.3. Existing proprietary software

Existing proprietary software is generally adopted to minimize cost. However, the source code and the development documentation are usually not available for verification. Therefore, the assessment of integrity of the application software must be based on validation. Hence, the development life cycle derived from IEC 880 is parallel to that for new software, as shown in Fig. 7 [4]. In this case operating experience and functional testing must be evaluated to compensate for the lack of original documentation. It may only be possible to present an adequate justification in this way for category C and perhaps category B systems.

Since operating experience is important in this case, it is also important to define how this should be assessed. This will necessitate that the requirements be clarified and that the common functions provided by the software be identified. Once this is done, the relevance of the operating experience to each factor listed below can be assessed:

— How many systems exist in service, identified by user and version?
— How similar are the new and the evaluated systems?
— Which parts of the software are used by the evaluated systems?
— How much operating time exists? The records should show the amount of continuous time, the number of years and the number of hours per day of operation.
— What records of failures and of revealed and corrected faults have been made?
— Are the challenges to the system in the new application similar to those in the existing applications?

It is suggested that significant operating experience is required; for example, at least two systems should have been in operation for several months before a meaningful evaluation can be made. A statistical approach to testing and demonstration of reliability growth may be of some value for low reliability requirements, but could be misleading. In the absence of details of the code structure, Bayesian statistics should be applied to black box test results (Sections I.2.3.2 and I.2.3.4 in Appendix I).

The confidence that can ultimately be achieved will depend on co-operation between the system developer and the vendor of the software to complete the evaluation. This should include, for example, methods and tools for the retrospective evaluation of a reference installation and methodical evaluation of reported faults and failures that have occurred. It is therefore important that the vendor has an existing process for handling problem reports.

2.2.4. Configurable software

A range of different forms of software can be described as configurable. Some forms are quite simple and are essentially data driven. Others are much more complex and functionally very rich, taking the form of application specific languages.

One form of configurable software consists of a set of basic software elements and a set of rules describing how these elements can be combined. The user is provided with a functional language to configure the basic elements into the desired system. Examples of these basic elements are AND and OR logic functions, integrators and timers. Another form of configurable software consists of basic processes such as signal input, signal checking, actuator operation and package control logic.

The requirements document may be converted into logic diagrams to define the signals and processing processes. For closed loop control these will include processes such as signal averaging, demand change rate limits, threshold comparison, proportional–integral–derivative (PID) control, auto/manual transfer, and output and check-back of actuator demands for movement. For multiplexed control these will include operations such as valve opening and closure, with alarms generated for valve movement failure and switchgear faults. It is usual for such operations to be performed by standard blocks of software, configured by the supplier of the applications software.

The process for establishing the configuration may involve the entry of signal tag numbers, ranges, control constants and their allowed ranges of adjustment, actuator tag numbers and permitted rates of operation, through a menu driven or manual pseudo-code generation set-up process. The selection of standard model objects for control logic options may be followed by entry of data to define and configure the characteristics for a specific object. Alternative processes involve the selection and positioning of standard blocks from a menu using a computer aided design (CAD) package, the inclusion of signal interconnection information by drawing lines linked with tag numbers, and the entry of tables of configuration information to the logic blocks.

One increasingly important form of configurable software is that used to produce VDU displays and touch screen displays, where the display and screen designs form the configurable information. In some systems, configuration can be performed at set-up time on-screen, graphically, through a graphical user interface (GUI). In other cases alphanumeric data strings or menu entries define the assignment of data to basic functions.

The deployment of existing configurable software may allow a claim that operating experience provides a valid justification for the correctness of the existing basic software. Therefore, only the application specific element need be subject to V&V processes. This is the basis of the commercial dedication process being

developed in the USA as part of the instrumentation upgrade programme of the Electric Power Research Institute (EPRI). This includes allowance for audits of the procedures and documentation of vendors.

Existing configurable software is often used in PLC applications. PLCs use a number of programming concepts. Ladder logic and symbolic function blocks are typical of the languages that are used to program PLCs. However, it has been found that the same ladder logic can execute in a different manner on different PLCs. Recent developments, including the use of portable equipment to reprogram PLCs in the field, have made considerable changes to their potential for different classes of use.

2.3. DEVICES CONTAINING SOFTWARE

Software is increasingly used by manufacturers of standard I&C devices to minimize product costs. One example is a microprocessor used to adjust the range and calibration of an instrument, rather than making the adjustment with fixed resistors and precision potentiometers. This software is often concealed, since it may not be stated in the manufacturer's product information that the instrument uses a microprocessor. If it is present, then it should be subject to V&V in the same way as other accessible or proprietary software in the system.

The specification of the product and the complete user documentation should be available. Traceability of the software releases should be provided with the product and the device identification details, since the software in this type of device is often changed to enhance the product.

2.4. SOFTWARE TOOLS

Software tools are becoming an indispensable part of the software development process. As the dependence on software tools in the development and V&V processes increases, so their potential to cause a fault that remains undiscovered increases. The faults could be simple, and result in the wrong version of software or data being used because of a configuration management fault, or more complex, and result in the introduction of illegal code owing to a code generator fault. In either case, the result could be catastrophic. The consequences of tool errors of this kind are such that active measures, based on the way software tools are used and on the way their products are checked, should be put in place. Existing and draft software standards (such as the IEC 880 supplement [11]) generally require that the integrity of a tool be the same as

that of the application, but lower integrity is acceptable if the tool output can be verified. An example of such a measure is source code comparison to reveal unauthorized code (Section III.7.2.2 in Appendix III).

3. SOFTWARE RELATED ACTIVITIES AND DOCUMENTS

The nuclear community currently considers that the requirements of IEC 880 [4] form a good basis for the production of software for nuclear safety systems. As indicated above, the present report assumes the development life cycle of IEC 880 and that illustrated in the IAEA QA guide, Technical Reports Series No. 282 [3], and assumes classification into categories A, B and C according to IEC 1226 [6].

The life cycles are illustrated in Figs 1 and 3. Each identified activity or phase in the life cycle is cross-referenced to either IEC 880 or TRS 282. For a full understanding of the different activities or phases, the reader should study the complete text in the two publications. The full life cycle applies to category A systems and simplified forms of the life cycle may be applicable to category B and C systems.

This section defines activities and documentation related to the development of any software. For systems not important to safety, activities and documentation used for V&V can be selected from the life cycle activities as necessary. The use of documentation for V&V purposes, as a function of the system classification, is defined in Sections 4 and 5.

Before the start of any system development, a nuclear plant project should request a feasibility study to consider the requirements and to:

— Show that at least one solution exists which is acceptable on cost and technical grounds, allowing for V&V;
— Find appropriate available technologies to realize the system.

The feasibility study should contain:

— A set of possible solutions ranked according to cost, risk and technological complexity or novelty;
— For each solution, the necessary planning and allocations of resources, to allow choice of the most cost effective and best technical solution.

In ranking the solutions, the different ways of implementing a computer based solution should be considered, with particular reference to V&V. Since different solutions will place very different demands for V&V, depending on the level of

assurance of integrity needed, V&V can have a major impact on the viability of the proposal. The output documents of the feasibility study are feasibility study reports.

3.1. LIFE CYCLE FOR NEW SOFTWARE

On the basis of the life cycles and document titles defined in the above mentioned references, new software has the following phases:

— System requirements specification;
— Computer system specification;
— Software design;
— Coding;
— Computer system integration;
— Integrated computer system test;
— Validation and commissioning tests;
— System handover;
— Operation, maintenance and modification.

The purpose of each of these steps is described below.

3.1.1. System requirements specification

During the system requirements specification phase the overall system requirements are written down. If the system is a safety system (IEC 880, section 4, para. 1 [4]), the system requirements specification is named the ‘safety system requirements’. The overall system requirements consist of:

— Functional requirements that come from the analysis of the plant behaviour under fault conditions, process analysis, human–machine interface analysis and performance analysis, according to the regime in which the system is to be applied;
— Applicable system safety principles, such as are claimed in the final safety analysis report or in the operating manuals for systems formally not important to nuclear safety;
— Other general requirements such as software quality assurance.

The system requirements specification is normally written using a natural language. It uses the necessary and correct nomenclature for plant components identification, tag numbers, specification of limit values, etc. In order for the system to fulfil the intended function and not jeopardize the safe operation of the plant, great

effort should be made in the compilation of the system requirements specification, and this is discussed in Section 1.3.1. The IAEA Technical Report TRS 282 (Section 2.1) states [3]:

> "The requirements specification…should contain enough information to totally identify and define the required functions so that the top level of software design can address all these functions in a traceable manner. To ensure that the requirements specification is read and understood, its contents should be simply and concisely presented. It should, however, be sufficiently detailed to show how all the relevant system requirements are being satisfied."

The output documentation of the system requirements specification phase is the system requirements specification or safety system requirements specification.

3.1.2. Computer system specification

During the computer system specification phase the hardware and software of the computer system are described. The computer system specification states the objectives and functions assigned to the computer system. The first part of this phase divides the functions among hardware and software to allow the individual requirements to be developed.

The output of this phase is the computer system specification, giving the computer system architecture, and is used to separate the three types of requirements:

— Software;
— Hardware;
— Integration.

Recommendations regarding the computer system specification can also be found in TRS 282 (Section 2.1, para. 1), described as the 'requirements specification', and additional information is available in IEC 880, section 4 and appendix A [3, 4].

3.1.2.1. Software requirements

The software requirements are specified in one of the three sets of documents produced during this phase of development. Their production is an activity in the computer system specification phase and the requirements are derived from the system requirements specification phase. The software requirements are the basis for the software design and must contain all information necessary for the design of the software.

The purpose of the software requirements is to state the software functional and performance requirements from an external and synthetic view, without stating how

they are programmed. A purpose of the software quality plan (required in TRS 282, Section 3.5, and IEC 880, section 3.2) is to state the programming rules to be applied [3, 4]. Verification will be required to check that the rules have been applied.

The output document of the software requirements activity is the software requirements documentation. In IEC 880, appendix F, this document is called the 'software requirements specification (functional and reliability requirements)' [4]. In TRS 282, Section 2.1, it is called simply the 'requirements specification' [3].

3.1.2.2. Hardware requirements

The hardware requirements are derived and established from the system requirements specification as an activity in the computer system specification phase. IEC 987 [1] deals with hardware and complements IEC 880 and TRS 282 regarding the specification, design and testing of the hardware. The hardware requirements are the basis for the hardware design and must contain all information necessary for the design of the hardware.

The output of activities to define the hardware requirements is the hardware requirements documentation.

3.1.2.3. Integration requirements

The integration requirements are established from the system requirements specification by an activity in the computer system specification phase. The integration requirements are the basis for the integration of the hardware and software within the computer system as well as the integration of the computer system in the plant. The integration requirements contain all information necessary for these two integration activities.

The integration requirements may be included in the software requirements documentation or the hardware requirements documentation or both. The integration requirements may also be documented in a standalone document identified as the integration requirements documentation.

3.1.3. System test plans

In parallel with the development of the computer system specification, it is normal practice to complete the generation of the integrated computer system test plans. The plans are based on the system requirements specification and the computer system requirements and are produced during this phase of the development. The plans should include the objective, the input data and the success criteria for acceptance test performance. It is convenient to develop a matrix showing the traceability of requirements to the system and acceptance test plans at this stage.

Integrated computer system test plans based directly on the system requirements specification have two benefits. Firstly, they reveal whether the stated requirements are testable; those that are not cannot be accepted as requirements without careful qualification. Secondly, the test plans will reflect the top level requirements that must be achieved. Test plans based solely on design requirements often define tests to demonstrate what the system can do rather than what the system is required to do. Verification must confirm that the tests refer to what is required to be done, and also assess whether the key performance parameters and functions are defined by the software requirements.

3.1.4. Software design

The software design phase contains three activities:

— Software system design;
— Detailed module specification;
— Module design.

The software design phase nominally ends when the design description is completed. The software requirements documentation should be completed and verified before work begins on the software design phase.

The output documentation of this phase consists of all the documentation produced as a result of the activities described above. This documentation is called the software performance specification in IEC 880, section 5 [4].

3.1.4.1. Software system design

Software system design starts with an analysis of the software requirements specification. The results of this activity are a breakdown of the software requirements into functional parts and subsystems (e.g. for data acquisition and for processing), with a document of the results, so that the design breakdown can form part of the software performance specification.

The output document of the design activity is the software system design specification. This specification may be improved and expanded in an interactive process during the design and coding phases, provided that any required verification activities are repeated.

3.1.4.2. Detailed module specification

In the module specification activity, the software architecture, as specified in the software system design specification, is used to identify the functions and produce

from them the necessary level of detailed information. The functional requirements are divided among the software modules. The modules are normally not larger than a few hundred lines of code.

The output documents of this activity are the ‘module specifications’. In TRS 282, this documentation is called the ‘detailed functional specification’ [3].

3.1.4.3. Module design

The module design activity includes definition of the actual algorithms and mathematical equations, as well as the detailed logic and data operations to be used to perform the required functions. The module design is the basis for coding.

The output documents of this activity are the ‘detailed design specifications’, one for each module or group of modules.

3.1.5. Coding

The software design description is the input to the coding phase. In this phase, the design is translated into code in a programming language, arranged to perform the functions stated in the computer system specification. The product of this activity is source code, available in a form able to be read and processed by a computer.

The output documentation of this phase is a printout of the source code, or a readable electronic record of the code.

3.1.6. Computer system integration

The computer system integration phase consists of three activities:

— Software tests;
— Hardware integration;
— Hardware and software integration.

3.1.6.1. Software tests

Software tests may be performed on modules, on groups of modules (e.g. sub-systems) and on the entire software specified in the software requirements specification. The hardware used to test the software may be different from that which will be used in the final system. The software tests must be performed according to a written test plan. The test plan may be part of the V&V plan or may be referenced in the V&V plan.

The output documentation of this activity is the software test report. For most systems this will be in the form of a set of test reports for each module and subset of modules, giving the test configuration, input data, acceptance criteria and comments on the results.

3.1.6.2. Hardware integration

Hardware integration must be completed before the hardware–software integration. This activity includes all aspects of hardware assembly and integration described in the hardware requirements documentation. This includes everything possible from populating boards to connecting cables, before transfer to site for installation. IEC 987 [1] gives valuable guidance here.

The output document of the hardware integration activity is the hardware integration report.

3.1.6.3. Hardware and software integration

This activity consists of two parts:

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

The output document of the hardware–software integration activity is the hardware–software integration report.

3.1.7. Integrated computer system tests

The integrated computer system tests should be performed before the system is transferred to site and installed. The final integrated computer system test is often combined with the factory acceptance test (FAT) to form a single test activity. As indicated in Fig. 6, these tests form part of the overall system validation.

The system is exercised during the integrated computer system tests by static and dynamic simulation of input signals. The tests should simulate normal operation, anticipated operational occurrences and accident conditions, as well as anticipated faults on the inputs to the computer system such as sensors out of range or ambiguous input readings. Each function should be confirmed by representative tests and it is usual to explore the performance boundaries, including use of signal inputs at their limits, during this phase.

The output document of the integrated computer system tests is the integrated system test report, more usually referred to as the factory acceptance test report.

3.1.8. Validation and commissioning tests

Validation and commissioning testing is carried out to verify that the computer system has been connected correctly to the rest of the plant and to confirm the correct

action of the system. The actions tested should cover all normal operation, anticipated operational occurrences and accident conditions as well as anticipated faults. All input conditions should be represented directly or simulated at the inputs to the computer. Each function should be systematically confirmed directly or by representative tests. However, it may not be possible to perform complete tests in all cases owing to restrictions on the operation of the plant or the system whose functions the computer system supports.

The validation and commissioning tests are usually combined with the site acceptance test (SAT), which includes verification of the operation and maintenance of the equipment.

The output documentation of the validation and commissioning tests phase is the commissioning test report, more usually referred to as the site acceptance test report.

3.1.9. System handover

Handover is a milestone in the project rather than a project phase. The responsibility for the system is transferred from the developer or system supplier to the user on successful completion of the validation and commissioning tests. It is necessary to establish that all requirements which the user has specified have been met.

The output documentation of handover includes a list of all items still to be resolved. This protocol must be agreed and approved by both parties.

3.1.10. Operation, maintenance and modification

During the operation, maintenance and modification phases, the system is in operation and may be subject to:

— Periodic tests, performed in order to verify that the system is not degrading;
— Modifications, implemented to enhance or change the functionality or to correct errors, in which case regression testing will be required;
— Change of parameters;
— Diagnosis activities, e.g. the execution of special diagnostic programs;
— Hardware components replacement, as made necessary by random failures.

The output documentation of the operation and maintenance phase includes log books, maintenance reports, periodic test reports and other relevant operation reports, including anomaly reports. If any form of modification is required (adaptive or corrective), then an appropriate life cycle for implementing a change would be

required. This life cycle should contain the phases of the main development, including V&V. These activities together with an impact analysis and regression testing will be necessary to ensure that the changes have been correctly implemented and no new errors introduced. This is part of the justification needed to return the system to service.

3.2. EXISTING ACCESSIBLE SOFTWARE

When a system is built of both new software and existing accessible software, the new system should be developed as described in Section 3.1. The existing software may be modified and integrated as described below.

3.2.1. Mapping to new software life cycle

When a system is built from existing accessible software, the documents and related information belonging to the product should be assembled and mapped directly onto corresponding information requirements of the life cycle for new software outlined above. The material should then be reviewed to ensure that it meets the requirements of the new software life cycle for development and verification documents and records.

Even if no changes of function of the accessible software components are required, new sets of documents for integrated computer system tests and for validation and commissioning tests are required. New FAT and SAT documents are needed, unless the system is a replication with no changes at all. These will have to be prepared from the new system requirements. If software change is required, which is usually expected, then a modification process will have to be adopted. This process should follow the development life cycle for new software, and should result in a new or revised set of documentation. The content of this may be quite similar to the existing documents, depending on the scale of change. This change life cycle could be identical to the life cycle used to complete any work that has to be done again for the development of new software for the system.

3.2.2. Identification of existing material

The following material should be identified when completing the mapping of the material of the existing product to the life cycle identified for new software:

— System requirements specification;
— Computer system specification:
  • Software requirements specification,

- Hardware requirements specification,
- Integration requirements specification;

— Software performance specification:
- Software system design specification,
- Module specification,
- Detailed design specification;

— Printout of source code;
— System integration report:
- Software test report,
- Hardware integration report,
- Hardware–software integration report;

— Integrated system test report/factory acceptance test report;
— Commissioning test report/site acceptance test report;
— Operation, maintenance and modification reports.

If the software has been used in a system of a similar safety category or integrity requirement, there may be extensive documentation from processes undertaken for an independent assessment and for licensing. This will be in addition to the documentation available from the development process. This material should be available to the licensee and will form an important part of the demonstration that the software is adequate for its intended use. This demonstration is particularly important for the higher safety categories of software.

3.3. EXISTING PROPRIETARY SOFTWARE

When a system is built up of both new software and existing proprietary software, the new software is developed as outlined in Section 3.1. The existing proprietary software may be integrated as described below.

3.3.1. Mapping to new software life cycle

The following system life cycle activities should be performed fully when using existing proprietary software:

— System requirements specification (Section 3.1.1);
— Integrated computer system tests (Section 3.1.7);
— Validation and commissioning tests (Section 3.1.8);
— System handover (Section 3.1.9);
— Operation, maintenance and modification (Section 3.1.10).

3.3.2. Identification of existing material

For existing proprietary software, the full documentation identified in Section 3.1 is not normally available for evaluation. To compensate for this, operating experience feedback is used to support the evaluation of the software. Experience feedback should use records with suitable QA, over suitable periods, as described in Section 2.2.3. Additional testing requirements will have to be derived from the new system requirements. The process requires that the documents of the integrated system tests and the validation and commissioning tests be generated, and the tests done.

For proprietary software only part of the documentation listed in Section 3.2.2 may be available. Depending on the safety category, documents described in Sections 3.1.2–3.1.5 may have to be generated.

If the same proprietary software has been used before, additional documentation may exist. Third party assessments and the justification done to demonstrate fitness for purpose of the software should be investigated, since this work normally produces extensive documentation. The availability of this documentation will greatly aid system justification. However, it is unlikely that application of the software in a category A system would be acceptable without a similar level of demonstration to that provided for accessible software.

3.4. CONFIGURABLE SOFTWARE

Configurable software is discussed with respect to its two parts, the basic software and the application specific configuration data. The former could be the operating system in a PLC or the standard functional modules available. The latter will be configuration data to define parameters such as signal ranges, alarm and trip levels, actuator and sensor identities, actuator and sensor names, functions assigned to each control switch and actuator, and other plant characteristics.

3.4.1. Basic software

The basic software is normally either existing accessible software or existing proprietary software. Depending on the case, the reader is referred to Section 3.2 or 3.3 for the documents and activities needed.

3.4.2. Configuration data

Although the data form an integral part of the total software, they can be separated from it. This allows software to be developed as a service, configured by the data for each application. The application data can then be prepared from the plant

information separately, generally by staff without software skills, and checked by manual or automatic error trapping methods. The same rigour is recommended in preparing and checking data as is applied to the production of the software. This implies that changes of data should follow the same processes as for changes in software.

The use of off-line software tools to process data into the form required by the basic software, and to check their form and consistency, is important in minimizing the labour of data verification. The potential for error can be restricted by careful menu driven operations and controlled key entry sequences, with prespecified checks and limits on settings and ranges.

In applications important to safety, the design process should carefully identify changes to data which are intended to be made during operation. These may include adjustment to instrument calibrations and to control system parameters using desk switches or their equivalent. Other information may require modification less frequently, under a controlled procedure. This may include alarm limits, instrument calibration scales, signal titles and VDU display layouts. The new information will have to be entered by downloading according to a controlled process before it can be run on-line. Physical control over changes should include locked controls or cubicles. Verification of the software should have confirmed previously that bounding and typical values of data which define constants and settings can be loaded correctly, in both cases.

4. VERIFICATION BY PHASE

Verification should be performed throughout the software life cycle to check the translation of information between defined phases. This extends from the system requirements specification phase through design to the operation, maintenance and modification phases.

It is recommended, and often a project or regulatory requirement, that the development and the verification teams be independent. This is required for category A by IEC 880, section 6.2.1 [4]. The verification policy and commitment to quality, as well as the procedures for ensuring that the policy is carried out, must be defined. If these procedures are not followed or the commitment is weak then the effectiveness of the verification will be undermined to a corresponding extent.

The V&V process is divided into phases as described in Section 3. This section describes the verification activities in each phase.

The elements of the process are summarized in a table for each phase (Tables I–XVII), which lists the input information, the output documents and the verification tasks. The tables also provide recommendations for the methods, tools

and techniques that can be applied to each category of software as determined by IEC 1226 [6]. The activities, with recommended or optional verification methods, are listed in the tables.

The purpose of verification of each phase is to confirm that the development products from that phase correctly reflect the requirements that served as inputs to that phase. The input and output documents listed in the tables are those used for verifying the phase, and not only the output documents of the phase activity. The verification process consists of reviewing input documents, code and data, and applying various tools and techniques to them to produce output documents that contain the results of the verification. Review is an important activity and it is defined in Section 1.5. Two review techniques are identified and described in Appendix I: walk-through and inspection.

4.1. NEW SOFTWARE

The verification of new software is based on the phased development programme. The system development plan has to identify clearly the development phases that are to be used. Each phase should be defined with respect to the required inputs, the specific activities and the expected outputs. The verification methods for identification of potential problems within the phase and the manner in which the verification team will be given access to the products of the phase should be defined.

Software verification usually begins with the software requirements specification. Experience has demonstrated that some failures in the software of critical systems can be traced to errors in the system specification. This specification should be verified, but this is not usually the responsibility of the software engineer. If the software developer finds that something was either misinterpreted or left undefined in the system specification, an anomaly report should be raised for the specification. The system design decision should not be made by the software developer.

The phases to be used in the development cycle are not prescribed here. However, information on applying verification will be illustrated using the development phases described in Section 3.1 and repeated below:

— System requirements specification;
— Computer system specification;
— Software design;
— Coding;
— Computer system integration;
— Integrated computer system test;
— Validation and commissioning tests;

TABLE I. SYSTEM REQUIREMENTS VERIFICATION

Verification input documents	Verification output documents
System requirements specification	System requirements verification report System requirements inspection checklist Anomaly report
is verified against the following documents:	
Concept documentation Development quality plans	

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Review of requirements	I.2.1.2	R[a]	R	R
Review of methodology, past experience and maintenance requirements	I.2.1.2	R	R	R

[a] R: recommended task.

— System handover;
— Operation, maintenance and modification.

The verification of new software between the software development phases is shown in Fig. 5. The documents needed to support the verification process and their interrelationships are also shown in Fig. 5. These documents are not necessarily the same as the output products from the phase as defined in the software development plan. For example, the inputs to the system requirements phase include concept documentation and the outputs include the system requirements specification. Since verification must ensure that the intent of the input documents is reflected in the output documents, both the concept documentation and the systems requirements specification are included as input documents to the verification of the system requirements phase. For the verification of each phase, the use of checklists is recommended to ensure that all aspects of the phase requirements have been addressed, but a checklist should not be considered a substitute for those analyses and engineering evaluations that are part of the verification process.

4.1.1. Phase 1: Verification of system requirements specification

The elements for verifying the system requirements specification are summarized in Table I. The purpose is to determine whether the system requirements specification reflects the system requirements that will satisfy the user needs, as

TABLE II. COMPUTER SYSTEM REQUIREMENTS VERIFICATION

Verification input documents	Verification output documents
Computer system specification, containing: — Software requirements — Hardware requirements — Integration requirements	Computer system requirements verification report Computer system requirements inspection checklist Anomaly report
is verified against the following documents:	
System requirements specification System development quality plans Minutes of formal requirements reviews Safety analysis report User documentation Applicable standards	

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Review of computer system specification	I.2.1.2	R[a,b]	R	O[c]
	I.2.1.3	R[b]	O	O
Review of functions allocation	I.2.1.2	R	R	R
Review of methodology, past experience and maintenance requirements	I.2.1.2	R	R	R

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

[c] O: optional task.

expressed in the concept documentation. The concept documentation can include such documents as statement of need, project initiation memorandum, feasibility studies, governing regulations, company procedures and policies, and end user acceptance criteria. The concept documentation is used as a secondary check on the verification of the system requirements.

An important part of verification of the system requirements specification is to determine whether all the functions have been clearly written down and described. Additionally, the verification should take into account other aspects of the design activity, including design methodology, past experience and maintenance requirements. A key vehicle in the verification of the requirements is a requirements traceability matrix, which provides a mapping of all system requirements to the upper level of documents. The use of a matrix to support a requirements traceability analysis

through each document level helps to ensure that all aspects of the system requirements have been addressed.

4.1.2. Phase 2: Verification of computer system specification

The elements for verifying the computer system specification are summarized in Table II. The purpose of the verification is to determine whether the proposed computer system reflects the system requirements as expressed in the system requirements specification and to show how the requirements trace back to that specification. The computer system specification can be produced with three distinct parts: software requirements, hardware requirements and integration requirements, including test requirements. Each part can be verified as a standalone element.

An important part of verifying the computer system specification is to determine whether the functions allocated to hardware, software and the interface have been clearly identified and suitably allocated. The functions allocated to software become the basis for the software requirements specification and subsequent design. In addition, the verification should consider other aspects of the system requirements such as impact on design methodology, past experience of the design team and maintenance requirements. Verification must also assess whether any non-functional requirements are introduced in this phase, for example those related to self-testing and internal diagnostics. Verification should show that these are fully documented and justified.

The verification of the computer system specification for a configurable system requires that the configuration data sources be annotated and traceable within the project design documents for I&C. This information will normally be transferred to an intermediate database identifying all signals, actuators, control logic and VDU designs. Verification should confirm that the database is a correct reflection of the design documentation of the project and that all deficiencies in source documents have been identified as anomalies and corrected.

4.1.2.1. Verification of software requirements specification

The elements for verifying the software requirements specification are summarized in Table III. The purpose of this verification is to determine whether the software requirements accurately reflect the functions allocated to software in the computer system specification and to ensure that all non-functional requirements have been carried through to the next phase of development. This includes analysing the software requirements for completeness, correctness, consistency and accuracy and ensuring that traceability of the specific requirements from the computer system specification is maintained. The verification of this phase also ensures that the requirements for the software interfaces with the hardware, the users and other

TABLE III. SOFTWARE REQUIREMENTS VERIFICATION

Verification input documents	Verification output documents
Computer system specification **Software requirements specification**	Software requirements verification report Software requirements inspection checklist Anomaly report
are verified against the following document:	
Computer system requirements specification	

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Review of software requirements	I.2.1.2	R[a,b]	R	R
	I.2.1.3	R[b]	O[c]	O
	I.2.1.6	O	O	O
Review of interfaces (with hardware, users, etc.)	I.2.1.2	R	R	R
Review of traceability	I.2.2	R	O	O

[a] R: recommended task.
[b] For each task, at least one of the recommended techniques has to be performed.
[c] O: optional task.

software are evaluated with respect to completeness, correctness, consistency, accuracy and readability.

4.1.3. Verification of system test plans

The elements for verifying the system test plans are summarized in Table IV. The purpose of this verification is to confirm that the functions specified in the system requirements specification and elaborated in the computer system requirements specification are subjected to adequate testing. The verification should check that all the requirements are testable. If untestable requirements are identified, these should be reported as anomalies for immediate resolution. If a matrix showing traceability of requirements to the system and acceptance test plans is available, this should also be checked at this stage.

One very important feature of the verification is to ensure that the tests relate to what the system is specified to be able to do, not what it can do. This is particularly important with respect to timing, response and refresh requirements.

TABLE IV. SYSTEM TEST PLANS VERIFICATION

Verification input documents	Verification output documents
System test plan *is verified against the following documents:* System requirements specification Computer system requirements specification System development quality plans	Software test plan verification report

Tasks	Techniques (Appendix I)	Category		
		A	B	C
Review of test plan	I.2.1.2	R[a]	R	R
Review of traceability	I.2.2	R	O[b]	O

[a] R: recommended task.
[b] O: optional task.

4.1.4. Phase 3: Verification of software design specification

The elements for verification of the software design specification are summarized in Table V. The purpose of verification is to confirm that the software design specification correctly represents the software requirements and that the design will satisfy the needs of the user as expressed in the system documentation.

An important part of the verification is to ensure that all of the functions allocated to software have been clearly incorporated into the design and to ensure that no extra functions have been added. The activities should ensure that the design is traceable to the required functions. Any design features added to meet non-functional requirements during this phase must be identified and justified. This includes the system health and self-test functions. The verification should not comment on the desirability of such features in the overall context of the application. However, it must check that the information or action offered by the non-functional requirement is compliant with the software requirements specification. It is noted in this context that an application system that includes continuous self-test functions could fail in the self-test mode, and would thus not be available to execute its safety function.

Verification of the software design should assess the data requirements with regard to the software requirements, and also check whether the required data are available. Methods of verification of configuration data should be identified and shown to be feasible using sample data. The verification must determine whether the required standards, practices and conventions are being followed sufficiently closely during the design process. It should consider other aspects of the design and the needs

TABLE V. SOFTWARE DESIGN VERIFICATION

Verification input documents	Verification output documents
Software design specification **Module specification** **Detailed designs**	Software design verification report Software design inspection checklist Anomaly report
are verified against the following documents:	
Computer system specification Software requirements specification System development quality plans	

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Review of software design specification	I.2.1.1	R[a,b]	R[b]	O[c]
	I.2.1.2	R[b]	R[b]	R
	I.2.1.6	O	O	O
	I.2.1.7	O	O	O
Formal verification of design	I.2.1.3	R	O	O
Assessment of performance parameters	I.2.1.6	R	R	O
Traceability of allocated functions	I.2.2	R	R	R
Assessment of data required	I.2.1.1	R	R	R
	I.2.2	R	O	O
Analysis of fault tolerance	I.2.1.1	R	R	R
	I.2.4.1	R	O	O
	I.2.4.2	R	O	O
Review of traceability	I.2.2	R	O	O

[a] R: recommended task.
[b] For each task, at least one of the recommended techniques has to be performed.
[c] O: optional task.

of maintenance. An optional task would be the formal mathematical verification of the design, focusing on the functional mapping between inputs and outputs.

4.1.5. Phase 4: Verification of software coding

The elements for software coding verification are summarized in Table VI. The purpose is to determine whether the source code and configuration data accurately reflect the functions allocated to software in the software design. This entails the

TABLE VI. SOFTWARE CODING VERIFICATION

Verification input documents	Verification output documents
Software coding	Source code verification report Source code inspection checklist Anomaly report
is verified against the following documents:	
Software design specification Module specification Detailed designs System development quality plans Minutes of formal coding reviews Unit test results (if available) Code analysis results (if available)	

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Analysis of source code modules	I.2.1.1	R[a,b,c]	R[b]	R[b]
	I.2.1.2	R[b,c]	R[b]	R[b]
	I.2.1.3	R[c]	O[d]	O
	I.2.1.4	O	O	O
	I.2.1.5	O	O	O
	I.2.1.7	R[c]	O	O
Review of use of standards, practices and conventions	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of maintainability	I.2.1.2	R	R	R
	I.2.1.7	O	O	O
Review of test results	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of traceability	I.2.2	R	O	O

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

[c] A combination of these techniques should be performed as appropriate.

[d] O: optional task.

analysis of source code modules and components for completeness, correctness, consistency and accuracy, and for traceability to specific elements in the software design. This verification phase ensures that the implementation of the module interfaces with the hardware, with users and with other modules is evaluated for completeness, correctness, consistency and accuracy.

TABLE VII. COMPUTER SYSTEM INTEGRATION VERIFICATION

Verification input documents	Verification output documents
Computer system integration	Integration verification report Anomaly report Report on computer system integration tests
is verified against the following documents:	
Computer system specification Software requirements specification Hardware requirements documentation Integration requirements documentation	

Tasks	Techniques (Appendix I)	Category		
		A	B	C
Review of test coverage	I.2.1.2	R[a]	R	R
Review of module integration and testing	I.2.1.2	R	R	R
Review of software test results	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of hardware–software integration test results	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Assessment of performance	I.2.3.6	R	R	R
Review of traceability	I.2.2	R	O[c]	O

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

[c] O: optional task.

Verification must also assess whether the predefined key performance parameters and critical functions are addressed and ensure that the required standards, practices and conventions have been followed sufficiently closely during the implementation process. In addition, the verification should take into account the needs arising from future maintenance activities. One element of the verification might be the examination of the results of unit tests or code analysis completed during this phase. Formal mathematical verification of elements of the design, focusing on the functional mapping between inputs and outputs, may be done as part of code verification. Comparison of the code function with the specified function is sometimes performed at this stage to provide a double check on the previous verification steps.

TABLE VIII. METHODS FOR COMPUTER SYSTEM INTEGRATION TEST

Input documents	Output documents
Computer system integration	Anomaly report Report on computer system integration tasks
is tested against the following documents:	
Computer system specification Software requirements specification Hardware requirements documentation Integration requirements documentation	

Techniques	References (Appendix I)	Category		
		A	B	C
White box testing	I.2.3.1	R[a]	R	O[b]
Black box testing	I.2.3.2	R	R	R
Load testing	I.2.3.3	R[c]	R[c]	R[c]
Statistical testing	I.2.3.4	O	O	O
Reliability growth model	I.2.3.5	n.a.[d]	O	O
Timing analysis	I.2.3.6	O[c]	O[c]	O[c]

[a] R: recommended task.
[b] O: optional task.
[c] Recommended if load and timing cannot be demonstrated by deterministic methods.
[d] n.a.: not applicable.

For configurable software, the equivalent to code verification is the verification of the detailed configuration data. In some cases the data will be expressed as extensive data declarations in the code. In other cases the data may be prepared manually from documents or by tools from databases and formatted for use by software. The data, as source code, or in formatted form, should be verified against the design document by inspection and by the use of software tools to check for completeness, consistency and accuracy.

4.1.6. Phase 5: Verification of computer system integration

The elements for verifying the computer system integration are summarized in Table VII. A list of test techniques related to the verification of the integration of software and hardware is provided in Table VIII. The purpose is to check that the functions allocated to software in the system design are correctly demonstrated by the tests on the integrated software.

TABLE IX. INTEGRATED COMPUTER SYSTEM TEST VERIFICATION

Input documents	Output documents
Integrated system (at factory) **Factory acceptance test report** *are verified against the following documents:* System requirements specification System test plan System development quality plans Factory acceptance test procedures	Factory acceptance test verification report Anomaly report

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Review of validation program (Section 5)	I.2.1.1	R[a,b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

TABLE X. METHODS FOR INTEGRATED SYSTEM TESTS

Input documents	Output documents
Integrated system *is tested against the following documents:* System requirements specification System test plan	Factory acceptance test report

Techniques	References (Appendix I)	Category A	Category B	Category C
Black box testing	I.2.3.2	R[a]	R	R
Load testing	I.2.3.3	R[b]	R[b]	R[b]
Statistical testing	I.2.3.4	O[c]	O	O
Reliability growth model	I.2.3.5	n.a.[d]	O	O
Timing analysis	I.2.3.6	O[b]	O[b]	O[b]

[a] R: recommended task.

[b] Recommended if load and timing cannot be demonstrated by deterministic methods.

[c] O: optional task.

[d] n.a.: not applicable.

TABLE XI. VALIDATION AND COMMISSIONING TESTS VERIFICATION

Verification input documents	Verification output documents
Integrated system (at site) **Site acceptance test report**	Site acceptance test verification report Configuration audit report Anomaly report
are verified against the following documents:	
System requirements specification Site acceptance test procedures System development quality plans Site acceptance test procedures Factory acceptance test procedures Factory acceptance test report Configuration documents	

Tasks	Techniques (Appendix I)	Category		
		A	B	C
Review of functional test results	I.2.1.1	R[a,b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of timing performance	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
	I.2.3.6	R[b]	R[b]	R[b]
Review of interfacing to other systems	I.2.1.1	R	R	R
Review of correction of anomalies and errors	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of installation	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of site conditions fulfilment	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of traceability	I.2.2	R	O[c]	O

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

[c] O: optional task.

TABLE XII. METHODS FOR VALIDATION AND COMMISSIONING TESTS

Input documents	Output documents
Integrated system	Site acceptance test report Anomaly report
is validated against the following documents:	
System requirements specification Site acceptance test procedures System development quality plans	

Techniques	References (Appendix I)	Category		
		A	B	C
Black box testing	I.2.3.2	R[a]	R	R
Load testing	I.2.3.3	R	R	R
Reliability growth model	I.2.3.5	n.a.[b]	O[c]	O

[a] R: recommended task.
[b] n.a.: not applicable.
[c] O: optional task.

The integration can take place in two steps. Firstly, there is the software integration testing of source code modules and components to demonstrate that the integrated software elements behave correctly, consistently and accurately and are functionally complete. The verification provides a check that the modules have been correctly assembled to the design and that the software tests provide sufficient coverage to confirm that the integrated software performs its design functions. This testing should preferably be done on the same hardware configuration that is to be installed. If this is not possible, the claims that the system meets the specified requirements will have to be confirmed during the integrated computer system tests. Secondly, the integrated software is combined with the target hardware and hardware–software integration testing is completed to demonstrate that the integrated hardware and software elements will operate correctly.

4.1.7. Phase 6: Verification of integrated computer system tests

The elements of the verification of the integrated computer system tests are summarized in Table IX. A list of test techniques related to these verification tests of the integrated computer system is provided in Table X. The purpose of this verification is essentially to check the intended or completed testing of the actual

TABLE XIII. SYSTEM HANDOVER VERIFICATION PROCESS

Verification input documents		Verification output documents		
Handover documentation *is verified against the following document:* Contractual requirements		Document checklist		
Tasks	Techniques (Appendix I)	Category		
		A	B	C
Review of documentation	I.2.1.1	R[a,b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

functions and performance of the integrated computer system against the specified functions and performance.

The verification phase confirms that the module interfaces with the hardware, the users and other modules are complete, correct, consistent and accurate. The verification of the integrated computer system tests also provides a check on the data values and ranges and verifies the validity of computed values which result from models or assumptions about the system operation or operating environment. The verification of the integrated hardware and software modules must assess whether the key performance parameters are achieved, whether the critical functions perform as required, and whether the hardware–software interface requirements have been satisfied.

4.1.8. Phase 7: Verification of validation and commissioning tests

The elements of verification of the validation and commissioning tests are summarized in Table XI. A list of test techniques related to the validation and commissioning of the integrated computer system is provided in Table XII. The purpose of this verification phase is to confirm that the integrated computer system has been installed correctly and is commissioned correctly, such that its performance and functions are satisfactory for use.

The essential part of this verification is to ensure that the scope of the testing provides a full demonstration of system functionality and operation, and to confirm the results of those parts which were simulated during the FAT. The verification thus concentrates on system timing, performance and interfaces to external systems. The acceptance test will also be run to verify that any errors and anomalies identified

during the system validation have been corrected. A review of the installation should be conducted to ensure that the integrated computer system has been installed correctly and accurately executes the functions allocated to it. This will include confirmation of all parameters dependent on the site or the site conditions to ensure that the initial values are correct.

4.1.9. Phase 8: Verification of system handover

The elements of system handover verification are summarized in Table XIII. The purpose of this verification phase is to determine whether all the documentation required for the operation and maintenance of the system is in place prior to the handover. This will entail a systematic review of the documentation to verify that it includes specific plant licensing and design basis documentation, engineering specifications and analysis, drawings and diagrams, vendor manuals and software documentation, test specifications and qualification test documentation. This review will assess the technical approach and scope of the engineering documentation that supports the computer system installation.

4.1.10. Phase 9: Verification of operation, maintenance and modification

The verification of the operational use, maintenance and modification processes can be very complex. In its simplest form it can be assumed that no corrective or adaptive maintenance of the software has to be performed. In this case, the verification activity will be limited to the performance of periodic functional tests as described in Section I.2.3.1 of Appendix I. Operational use will cover normal operation as well as the replacement of defective parts and changes to the software. It is necessary to examine the impact of the required change against the original requirements. This will determine whether the corrective or adaptive maintenance procedures in place are sufficient. If so, verification will be confined to a check on the content of the operation and maintenance manuals. It should ensure that proper procedures with appropriate configuration controls are in place for performing tasks such as checking the installed software, reinstalling the software and replacing programmable read-only memories (PROMs).

If use and maintenance require software changes to be made, then corrective and adaptive maintenance procedures should be available. The verification processes should show that use and maintenance procedures are complete and will allow software changes to be performed in a traceable manner using a process that results in correct software. The corrective and adaptive maintenance change procedures would normally adopt a life cycle and contain a series of verification steps similar to those for new software. It is expected that these change procedures would require an impact assessment. Typically, this might examine the consequences of the change on

TABLE XIV. SOFTWARE CHANGE VERIFICATION PROCESS

Verification input documents	Verification output documents
Changes in system requirements specification	Software integration test report
Changes in computer system requirements specification	Computer system verification report
Software integration test plan	Anomaly reports
Computer system test plan	Non-regression test reports
Anomaly reports	Maintenance change log
Change orders (including non-regression test)	

Tasks	Techniques (Appendix I)	Category		
		A	B	C
Review of impact of changes analysis	I.2.1.1	R[a,b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
	I.2.4.1	R	O[c]	O
	I.2.4.2	R	O	O
Review of tasks to be performed according to the phases for new software	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of non-regression analysis	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

[c] O: optional task.

the data and data structures and on the control and data flow. The scope and accuracy of this analysis would also be checked as part of the verification process.

The elements of software change verification are summarized in Table XIV. The purpose of this verification phase is to determine whether the modified software will accurately implement the requirements and that 'secondary errors' have not been introduced into the software. This requires that the verification of the phases which were affected by the change be repeated to ensure correctness, consistency and accuracy.

Software change verification must assess whether key performance parameters, critical functions and other system requirements continue to be satisfied to the same level of confidence that was achieved during the original verification. The verification steps of this process are similar to those for new software. There will be an additional requirement to ensure that no unintended changes have been introduced. Verification

may therefore involve regression testing of the system. One additional verification task is to ensure that all the anomalies identified in the original documentation have been cleared.

4.2. EXISTING ACCESSIBLE SOFTWARE

Software reuse is introduced in Section 2.2.2. The process of evaluation and assessment is described in the first supplement to IEC 880 [11], and includes verification activities and evaluation of feedback of experience. Before verification for reuse is possible, the user or application developer must have access to a large part of the documentation which describes both the development and the previous application of the software. The verification serves two distinct purposes. The first purpose is to ensure that the functionality and performance claimed for the system and existing software correspond to the documented information. The second purpose is to ensure that the mapping of software functions is performed correctly from the software proposed onto an arrangement which meets the new requirements. This may require full V&V as part of an adaptive maintenance procedure.

Verification of the claimed functionality and performance will require use and review of the existing documentation, in the same way as has been described for new software. In this case, the development life cycle for new software will be followed and previous verification reports will be used. If anomalies are found, these will be recorded, rather than being resolved immediately in the normal development life cycle. It would be expected that the anomalies would be cleared, for example as part of the maintenance process, as software changes.

The elements of software change verification are summarized in Table XIV and are discussed in Section 4.1.10. The verification of any change process will follow a maintenance life cycle.

4.3. EXISTING PROPRIETARY SOFTWARE

The reuse of existing proprietary software is discussed in Section 2.2.3. It is considered unlikely that reuse of proprietary software will be possible for category A systems, since the owner of the software may not permit access to the documentation which describes the development and past application of the software. The process of evaluation and assessment is described in the first supplement to IEC 880 [11], and includes verification activities. It is likely to stress evaluation of feedback of experience.

A prerequisite for using existing proprietary software is verification based on available documentation and system requirements. The verification process serves

TABLE XV. CONFIGURABLE SOFTWARE: ADDITIONAL VERIFICATIONS

Verification input documents	Verification output documents
In addition to verification according to new software, configurable software *is verified against the following documents:* Applicable documents according to new software	Verification report

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Assessment of data requirements (phase 3)	I.2.1.1	R[a,b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
	I.2.1.3	R[b]	O[c]	O
Review of database structure (phase 4)	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
	I.2.1.3	O	O	O
Review of data completeness, correctness and accuracy (phase 4)	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
	I.2.3.2	O	O	O

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

[c] O: optional task.

two distinct purposes. The first is to ensure that the functionality and performance claimed for the system and the existing software correspond to the documented information. The second purpose is to ensure that the mapping of software functions is performed correctly from the software proposed onto an arrangement which meets the new requirements.

The available documentation will be used to review the product against the requirements in much the same way as described for the documentation of new software. The application of verification to existing proprietary software within the software development phases defined in Section 3.2 is shown in Fig. 5. This production life cycle can be followed for existing proprietary software and any existing verification reports will be used. For the verification of proprietary software, more emphasis must be put on additional black box testing and statistical techniques to establish the acceptability for use in category B and C applications.

4.4. CONFIGURABLE SOFTWARE

Configurable software is normally a combination of basic software and an application specific extension. The basic software can be treated as new or existing software, with either accessible or proprietary documentation. For these cases, the verification process of Section 4.1, 4.2 or 4.3 is applied. In many cases, the basic software has already been verified for some other application. It is then necessary to review the verification and decide whether it was of acceptable rigour for the application.

The application specific extension can be treated as new software. It can consist of either configuration data or code, and each should be appropriately verified and validated. If the modules of the basic software have been modified for the application, they should be verified as new software to the best extent possible.

If the application part of the configurable software is treated as new software the process outlined in Section 4.1 should be followed. Since the application part of the configurable software may consist of data, additional V&V activities for this are summarized in Table XV and described below.

4.5. DATA VERIFICATION

Experience indicates that data verification should be repeated at least twice:

— During the data identification phase, comparing the intermediate database generated by the data identification activities with the project data sources. The intermediate database should be shown to be correct and then used as a secondary source.
— During the production of final data for loading, comparing them with the secondary source.

It is a clear advantage if the data for loading can be directly inspected. Printouts with suitable QA are needed. VDU inspection methods can be used.

The use of automatic error trap software is of great value and such traps should be used throughout the verification process. Checks that may be carried out on the data include those for the following:

— Completeness;
— Consistency;
— Range;
— Implied rules: for example,
  • Naming conventions,

- Abbreviations,
- Information order.

Full exploitation of automated checks is possible only if the source data have been put into a master project database. Potential differences between different sections of the database include differences in drawing or document issue level and differences of time of data entry from project sources and drawings. These will be critical in achieving satisfactory data. If the tool is used too early in the project time-scale, very many unnecessary or misleading errors will be found owing to inadequate sources of information. Effective methods of referring discrepancies for resolution by the plant design and engineering organization are essential.

4.6. CATEGORY B AND C SYSTEM VERIFICATION

Many category B and C systems use proprietary software, often with configuration data to define input signals, control algorithms, VDU display designs and record contents. The verification for category B and C systems may use a less rigorous process than that described for category A systems. This is due to the lower importance to safety. The relaxation of rigour in verification can be counterbalanced by more extensive historical evidence of production and installation than is usually available for category A systems. The availability of this evidence determines the permissible reduction in rigour of verification for a given confidence in integrity. Good evidence permits greater confidence to be placed in the quality of the products. Sufficient verification may include only simple peer review of the software design documents, with simple testing on a functional basis. The use of path test coverage metrics, code analysis or other more advanced methods is not normally needed.

Typical examples of category B and C functions are:

— Closed loop control of the reactor power or flux;
— Closed loop control of reactor temperatures, especially where reactivity is affected;
— Display of alarms;
— Display of event history;
— Records for diagnosis.

4.6.1. Category B and C requirements and design verification

The requirements for a category B or C system are seldom as rigorous or formal as those for a category A system. Considerable freedom may be allowed to a supplier

in developing a VDU display system or in providing control functions from a standard library. A two way process of agreement of requirements and of development of the VDU display designs or of the presentation methods for control algorithms is frequently followed. This allows for the use of standard packages of software from the supplier.

The documents given to the software supplier to define the requirements will consist of project design drawings and descriptions for the mechanical, electrical and other equipment. These documents should be reviewed and agreed by the supplier. The initial process of verification may be a review of the documents by the supplier's organization, to confirm their suitability, to clarify the requirements and to return any discrepancies or obvious errors for resolution by the project management.

The design verification of these requirements is important, as discussed in Section 1.4.1, but is outside the scope of this publication. In many category B or C systems, these requirements are presented to the supplier or developed by the supplier as configuration information, VDU layouts and signal lists, or as code in a simple application language, as discussed in Sections 2.2.4 and 3.4.

The documentation may therefore be in the form of lists of configuration constants, tables and control diagrams showing the interconnections and signal flow, rather than any time based flow of operations and information. Verification planning must ensure that this documentation exists and is suitable for manual and automatic checking processes.

The process of establishing the VDU display designs for a project is complex and requires care and planning. Several hundred displays may be involved, which must accurately reflect the plant design and instrumentation, the agreed naming conventions and the groups of operations needed. The designs should be verified by reference to the operating staff of the end user and by review by human factors specialists.

4.6.2. Category B and C implementation verification

The development of applications software requires the preparation of many detailed documents. Depending on the project management arrangements, these may be the responsibility of the project or of the software system supplier. The project quality plan should define these documents, the responsibility for originating them and the need for review or verification of each document or listing.

Review of the software output documents (such as listings of configuration data and signal interconnections) is desirable for category B and C systems, either as an internal supplier verification or as a review by the customer. Methods of verification are discussed in Sections 3.4.2 and 4.5.

Independent verification of code for category B or C systems may not be necessary if there is sufficient confidence in system maturity, with proven capability,

and in the reliability of the compilation tools used. Any verification needed should be identified in a V&V plan, and developed as a verification document for the application requirements.

Where a system is dependent on extensive configuration data, and the data are automatically produced by tools operating on a database, the capability and reliability of the tools must be demonstrated. This is discussed in Section 5.5

5. VALIDATION

Validation is performed after the system integration phase and before the system is put into service. If any subsequent changes are made to the software, then revalidation would be required before the system is returned to service.

Validation can be broken down into a number of tasks, outlined in general terms below. The elements of validation are summarized in Table XVI. A list of test techniques related to validation is provided in Table XII.

The purpose of validation is to demonstrate that the integrated system of hardware and software correctly executes the functions defined in the system requirements specification. This requires testing of the integrated computer system to determine that the system performs all specified functions completely, correctly, consistently and accurately. During the validation process, the interfaces with the other systems and with users are evaluated with respect to completeness, correctness, consistency and accuracy. Validation must assess whether the key performance parameters, critical functions and other system requirements have been achieved, and must demonstrate that the system fulfils the system requirements specification.

For validation to be fully effective, its scope must be clearly defined. Validation is based on the system requirements specification. Part of the validation of a safety system can be the validation of the actions of the safety functions of the system, as claimed in the safety justification. This will involve a demonstration that each safety action is performed correctly and that the functionality is still available in an acceptable manner for all anticipated abnormal conditions. These should include sensor, equipment and actuator failure, failure of communications and operation during maintenance and testing activities.

General descriptions of the validation activities are given in Sections 3.1.7 and 3.1.8, and of their verification in Sections 4.1.7 and 4.1.8. Normally, these validation activities are done by means of planned FATs and SATs, with appropriate reports of the testing done and anomalies found. The system test plans and their verification are discussed in Sections 3.1.3 and 4.1.3, respectively.

TABLE XVI. INTEGRATED SYSTEM VALIDATION PROCESS

Validation input documents	Validation output documents
Integrated system	System validation report Acceptance test report Anomaly report
is validated against the following documents:	
System requirements specification Factory acceptance test procedures	

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Review of test coverage	I.2.1.1	R[a,b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of validation test results	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of anomaly records	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of verification documentation	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of operation and maintenance records	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of system documentation	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of traceability records/matrix	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

5.1. NEW SOFTWARE

5.1.1. Validation process

The validation process is based on the requirements of the complete system. It is therefore not dependent on the means of development of the system which contains the software. Traditionally, validation has begun from the system requirements

specification. Experience has shown that failures of software in critical systems can often be traced back to errors in the system requirements specification (Section 1.4.1). Notwithstanding this, the computer system validation activities considered here can only take the system requirements as input, and a validation activity can only provide any degree of confidence in the system performance from this starting point.

The documents taken as input to the validation process include:

— System requirements specification;
— Computer system requirements specification;
— Validation test plan;
— Validation test procedures;
— Validation test design;
— Validation test cases.

Static and dynamic simulations of input signals can be arranged to operate the computer system in modes which represent normal operation, anticipated operational occurrences and accident conditions which require system action. For reactor protection systems, each function should be confirmed by conducting a representative test for the actions caused by each parameter for trip or protection, singly and in combination. The validation test design should therefore define the required input signals with their sequences and values. The validation test cases must describe the anticipated output signals with their sequences and values and the signal level acceptance criteria. Additional requirements for validation include verification that the proposed test coverage includes all functions. This should be confirmed by a documented trace to the requirements specification (Section 1.4.2).

These tests should also demonstrate any features of the computer system that are claimed in the safety justification.

5.1.2. Output documents

The output from validation will be two documents:

— System validation report;
— Anomaly report.

The results of the computer system validation are given in the computer system validation report, which should summarize the following:

— Hardware and software tested;
— Equipment used to complete the tests;
— Equipment calibration status;

— Simulation model used;
— Functions tested;
— Discrepancies found.

The system validation report should state either that the software has passed validation or that it has failed, and give the reasons for failure. As with the verification reports, the validation report should not advise on how the problems could be overcome, and should not indicate preferred solutions. In addition, as part of validation, it will be necessary to make an assessment of and to record the test coverage. Validation should check that all the requirements have been covered by the tests. These should include functions, timing performance and abnormal conditions.

The anomaly report documents generated during validation should record each anomaly detected and give the following information:

— Description and location of the anomaly;
— Impact;
— Apparent cause of the anomaly and nature of the error;
— Criticality of the fault;
— Recommendations for action.

Each anomaly should be resolved satisfactorily before validation can formally be completed.

5.2. EXISTING ACCESSIBLE SOFTWARE

Validation of existing accessible software will involve the same steps as for new software, to the extent that the documentation is available. If any changes are required, these should be made using adaptive maintenance procedures, which would require a repeat of validation for the affected parts of the software. Thus, if there were a change to the basic requirements, the whole cycle would be followed, with verification of all phases. A change in the data describing the configuration would possibly only require verification of the integration phase. Validation would take place after these steps.

To allow validation of existing software, it should meet the following requirements:

— The existing software was developed according to good software engineering practice, following a QA plan developed in accordance with recognized standards, such as IEEE Std 830, IEEE Std 1012 and IEC 880 [4, 7, 16];

— A documented history exists, demonstrating reliable operation;
— An accurate user manual exists.

If source code and object code are not accessible, then the following documentation should be analysed:

— Functional specifications;
— Interface specification;
— Design documents;
— Test files.

The analysis should check the use of interrupts, memory allocation and recursion. Suitable tests should be defined to show that these features operate correctly. Checks of the error handling resources and of self-tests (such as those of memory stacks) are needed. Validation testing of the complete system, including the existing accessible software, should be done as described earlier in Section 5.

5.3. EXISTING PROPRIETARY SOFTWARE

The validation process for systems using existing proprietary software cannot normally make use of:

— Verification documents, produced during development of the proprietary software;
— Source code documentation;
— Software requirements for the proprietary software.

In consequence, the validation of the end products must be based on the verified system requirements specification.

Operational history over a suitable period should be evaluated for the confidence possible in the performance of the proprietary software and in its suitability for use within the proposed system.

The operational history should include:

— History of release;
— Number of licensees and extent of use;
— History of errors and modifications to correct defects (fixes).

On the basis of the operational history, additional validation testing should be defined where doubt or lack of confidence exists. Validation testing of the complete system, including the existing proprietary software, should be done as described earlier in Section 5.

TABLE XVII. DATA VALIDATION PROCESS

Validation input documents	Validation output documents
Integrated system	System validation report Acceptance test report Anomaly report
is validated against the following documents:	
System requirements specification Factory acceptance test procedures	

Tasks	Techniques (Appendix I)	Category A	Category B	Category C
Review of data dictionary	I.2.1.1	R[a,b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]
Review of validation test results	I.2.1.1	R[b]	R[b]	R[b]
	I.2.1.2	R[b]	R[b]	R[b]

[a] R: recommended task.

[b] For each task, at least one of the recommended techniques has to be performed.

5.4. CONFIGURABLE SOFTWARE

The validation of systems containing configurable software can be divided into two parts: the validation of the basic software and the validation of the configuration data. The validation of the basic software will follow the approaches described in Sections 5.2 and 5.3 for existing software. The configured aspects of the software should be validated as new software. Extensive experience of validation of this type of software exists for PLCs.

Validation of the basic software should include tests of each module or software functional unit for the full allowed range of operations, and may be done in the factory. The supplier should be expected to produce evidence of validation of each such function, or to perform such tests. The basic functions may be existing accessible or proprietary software and should be validated as described earlier in Section 5.

5.5. DATA

Data validation will be divided into two parts: validation of the data values used and validation of database management tools. Data validation is required to ensure

that the data items used are correct and will subsequently generate the correct functionality for the system.

5.5.1. Data validation

The elements of validation of data are summarized in Table XVII. The purpose of data validation is to ensure that each item of data in the system in its operating state is correct for the requirements. The validation process for data may take many forms, some of which may be automated with considerable gains in efficiency. This may be done by processing the primary plant database used by the project to generate test information and to provide reference lists in the form of test plans.

Validation of data requires testing to show that input signals are assigned to the correct computer inputs, with the correct hardware, and with correct software characteristics, alarms, signal titles and display features. The correct actions must be shown to be taken for control switches or soft controls, with the correct output actuation drive hardware, and with correct software assignments of control function and logic.

Site acceptance testing is normally combined with plant connection of sensors and control outputs, with standard written tests for each class of signal. A typical nuclear plant may involve many thousands of such signals, with a requirement to validate several characteristics of each signal. Therefore, careful QA and rapid methods of resolution of anomalies are needed.

The validation of data can exploit the use of test platforms off-site to reduce pressure on on-site activities. This will allow testing of much of the configuration and functionality information in FATs.

5.5.2. Validation of database management tools

IEC 880 recommends the use of automatic tools for data management, but recommends that the tools be thoroughly and appropriately tested [4]. Other reference documents require that validated tools be selected for the necessary integrity level. The basic principle is that if a tool is used to produce code or information which cannot be otherwise verified and validated, then it must meet the full technical, QA, verification and validation requirements of the application.

It is essential that the use of automatic handling of data during all possible stages not be discouraged by unrealistic requirements for tool validation. Experience indicates that data recorded and produced automatically using a database, and checked by automatic error trappers, are more accurate and more dependable than manually produced data.

The tools used must be assessed carefully. For system data important to safety, the appropriateness of the tools and processes used should be justified in a document. Guidance is given in the supplement to IEC 880 [11].

5.6. VALIDATION OF CATEGORY B AND C SYSTEMS

Application of the most rigorous requirements of IEC 880 is not usually expected for category B and C systems. However, the most important tasks of validation of category A systems may be carried out for category B and C systems. This should include extensive functional testing; qualification is required by IAEA Safety Guide 50-SG-D8 (Section 4.4) and will necessitate such testing [15].

The level of independent management and validation planning required for category A systems need not apply to category B and C systems. It is recommended that validation be carried out with the systems installed on-site, so that it is subject to representative environmental and plant conditions. Some validation requirements may be less demanding for category B and C systems, and feedback of experience is very important as a quality indicator.

Validation is normally considered to be the most important part of the process of V&V for category B and C systems. Validation tests should be generated from the system requirements to demonstrate that the system will function as specified. An important part of validation of many category B and C systems is data validation, described earlier.

The validation tests may also include those states identified as potential hardware failure modes, or ambiguous, out-of-range, inconsistent or incorrect inputs. A methodology for identification of such states and a recommendation for validation testing to resolve them are given in outline in IEEE Std 7-4.3.2-1993 as "Identification and resolution of abnormal conditions and events (ACE)" [17]. The method can result in testing which shows that the system does not do things it is not supposed to do. This has frequently been required by site commissioning engineers, and such testing is sometimes referred to by them as 'negative testing'.

It is also good practice for validation to be used as part of the customer acceptance process, and therefore for the documents to be provided for review by the end user. Validation test reports are normally produced as part of the FAT and SAT process and for the plant commissioning records.

6. LICENSING

Countries constructing and operating nuclear power plants and other facilities are expected to establish a regulatory body, whose duties include the enforcement of applicable regulations and of the terms of licences [18]. The approach to licensing or regulation differs between countries because of their different legislative backgrounds and engineering practices. The activities and actions of the licensing authority,

including the relationship with the licensee, are therefore governed by the regulatory framework that exists in a particular country.

Licensing activities range from strict and thorough reviews and inspections of all available material, in order to establish confidence in the design and development processes, to a minimum of reviewing activity, depending on the safety role of the system and the national regulatory practice.

The responsibility for the safe operation of the nuclear plant rests with the licensee. This responsibility includes the safe implementation of software based systems, whether designed directly by the licensee or by an outside design organization. The role and responsibilities of the licensee and outside design organization in the software life cycle should be defined in detail to allow for this.

Licensing is in part a matter of confidence between the licensing authority and licensee. Generating confidence is a complex process, and confidence will be based on experience, culture, legislation, tradition and other social factors. These factors imply that the licensing requirements cannot be defined in an exact way in any international guideline or standard. Generally, a series of safety documents must be prepared for the regulator, including documents which give details of the V&V done and which justify the performance and safety of the system. Relevant recommendations for V&V can be given only to cover available tools and methods, together with their benefits and drawbacks. Information of this type is given in this publication and the reader must judge its suitability in relation to specific licensing needs.

The licensing authority should consider the overall arrangements by which the licensee and the relevant suppliers intend to implement and maintain the software based system. These include the licensee's and supplier's QA arrangements and V&V programme and the application of technical standards for safety critical software. In addition, the licensing authority may choose to select specific areas of interest for in-depth review. It is important not to lose the safety perspective in the licensing process. The safety significance of the areas selected for in-depth review needs to be considered by the licensing authority.

Licensing cannot be exclusively based on reviews, inspections or test activities. Licensing must also involve confidence in methods and quality. It will involve the application of QA standards or guidelines, such as ISO 9000, IEEE Std 730 and IAEA TRS 282, and of standards for the development and design of software based systems, such as IEC 880 [3, 4, 9, 10, 19]. The licensing authority must also have confidence in the licensee and the ability of the relevant supplier to use and to implement the methods outlined in standards and guidelines. Methods used to build up the necessary confidence may include audits, inspections and reviews.

It is beneficial for both the licensing authority and the licensee, and consequently to the supplier and licensee, to reach an understanding on:

— What development methods are to be used;

— What approach and methods the licensing authority will use for assurance of the correct use and implementation of the methods.

This understanding should be established before the development and design activities begin.

The licensing authority should be able to evaluate the results of reviews and inspections. This evaluation can be undertaken by the licensing authority alone or with support from independent experts.

The review process should include consideration of independent assessments (third party reviews) performed on the software. This could include the application of inspection and analysis techniques identified in Appendix I. For example, code inspection, static analysis, tool validation and dynamic testing could all be applied on a sample basis. Consideration of the appropriate time to apply the independent assessments is required. For example, the application of static analysis following completion of the manufacturer's V&V processes could be used to build confidence in the delivered software.

7. CONCLUSIONS

The main conclusions of this publication are given below. They are not in any order of importance.

(1) Verification and validation are two different processes with different end objectives that make use of different techniques. The two processes are complementary and the confidence in the final software product will be dependent on the effectiveness of both processes.
(2) There is no single definitive method for application and completion of the processes of verification and validation of software and computer systems. The methods adopted for V&V will depend on both the processes and the product and therefore they should be selected on a case by case basis.
(3) Verification is needed to check that the inputs to and the outputs from each step of the development process correspond correctly. Verification of the software development steps is usually undertaken from documentation. For example, in a conventional development, the design documents are checked against the specification to provide assurance that the original requirements are reflected accurately in the software design and therefore will permit successful representation in the software code.
(4) Validation is necessary to confirm the software performance and functionality by tests against the system requirements specification. Validation provides

assurance that the integrated system performs in the manner required. Validation processes typically include extensive testing and review of the intermediate and final products generated during the development and verification.

(5) The amount of V&V performed can be varied according to the importance to safety of the computer based system. The highest level of assurance is necessary for category A safety system applications. Lower levels of assurance can be accepted for category B and C applications.

(6) Category A applications require that V&V be conducted by a team which is independent of the development team and whose competence is at least equivalent to that of the development team. This can be achieved by arranging that the management of the V&V team be independent of that for the development team, in accordance with IEC 880, sections 6.2.1 and 8 [4]. These teams can also be subject to external review.

(7) Category B and C applications may also require independent V&V. The team completing the V&V should be independent of the development team. While staff may be members of both teams and so undertake both development and verification activities, self-verification should not be permitted. The rigour of the V&V measures will usually be less than that applied for category A systems, although the same tools and techniques as for category A can be used.

(8) The development of new software should follow a recognized life cycle with well defined phases, inputs and outputs. The verification of the outputs against the inputs should be planned into the process at the start of the development and include suitable hold points and reviews. The nature of the verification intended should be defined and agreed before the process is started.

(9) The production of a new system based on the reuse of existing software requires different practices to be followed. If the complete documentation is available, then a life cycle similar to that for corrective maintenance of new software can be followed. Although all changed items will be verified, this may be only a partial verification of the system. This partial verification has to be supported by appropriate evidence provided during the complete system development and by an impact analysis to assess the consequences of the changes.

(10) When existing proprietary software is used, where the documentation is not always available, only limited V&V will be possible. The evaluation of feedback of operating experience is important, and guidance is being prepared for this in the first supplement to IEC 880 [11]. The limited verification may consist of extensive testing to confirm that the system requirements specification is being met. This is essentially validation rather than verification. It is very unlikely that software reused in this manner could be included in a category A system.

(11) Some forms of software are configurable, and are made ready for use by supplying data or preparing statements in an application specific language. In this case, verification has two parts, that of the basic software and that of the

data or application language. The verification of the basic software should have been completed beforehand, while the second part of the verification concerns the new information. When an application specific language is used, the new software life cycle should be applicable. Verification of data requires different techniques such as reference to the original plant design documents and the project master lists of information.

(12) Category B and C applications often use some specially written application software and extensive existing system software. The V&V for such systems should follow the recommendations given in this publication. It should thus contain a check on the accuracy of the design against the requirements, the accuracy of the implementation against the design, and the performance and functionality of the integrated system compared with the requirements. This can be achieved for the specially written software by using the methods and tools described in this publication. The V&V of the existing software should already have been completed and recorded by the software supplier. The acceptance of the system will be dependent on the assessment of these records, for compliance with appropriate international and national standards and for compliance with project requirements. This assessment should be supported by extensive validation, which must include testing.

(13) If existing software is used, the system will also normally require application software, which may be automatically generated. Current methods of software generation allow extensive systems to be built from existing software modules, often by configuration using graphical methods and database tools. Such software requires V&V of the software building material and of the software tools used. Tools are typically used for preparation of the requirements and their translation into the application information, configuration data or code. If the validity of these tools can be accepted, then the V&V of the application can often be simplified. For example, it may be sufficient to undertake a systematic confirmation that the configuration data are correct and correspond to the information in the source documents. These sources describe the plant characteristics, signals, output actuators and actions.

(14) Suppliers of commercial grade process control and protection equipment normally will modify their basic software continuously to add extra features and to make corrections. If this type of equipment is qualified for use in a particular application, then care must be taken to ensure that neither the application nor the system software is changed during the period of service. This requires that strict configuration control be applied to ensure that any replacement or additional unit contains the same version of system software that was qualified for use.

(15) The use of software tools for the preparation of requirements, their translation into code, the assessment of code and the preparation of configuration

information for computer based systems is a topic whose full discussion is beyond the scope of this report. IEC subcommittee SC45A is producing tool requirements in the first supplement to IEC 880 [11]. One relevant principle which has been identified is that if a tool is used to produce code or information which cannot be otherwise verified and validated, then it must meet the full technical, QA, verification and validation requirements of the application. To avoid the complexity of applying such a stringent requirement, the output of tools should be simple to verify against the source information. It has been found in practice therefore that tool output should generally be printable, either directly or after simple translation, and the prepared tool output for any loading to a target machine should be able to be inspected by the use of a VDU display.

(16) Nuclear plants are making increasing use of VDUs for the presentation of information to the plant operators. The V&V of such information is often complex, and depends on the concurrent verification and validation of the plant signals, and in some cases of the plant output actuators and control room switches. A very large quantity of information must be verified and shown to be correct. For example, the total data might be 20 000 signals, sometimes with ten or more characteristics each, and 400–500 detailed display layouts. Verification must be followed by validation to show that the installed signal transmitters and plant arrangements are correctly reflected in the system. In this situation, very careful administration must be employed, and must be able to handle the large volume of data and record the potentially large number of defects in the configuration.

Appendix I

TECHNIQUES FOR VERIFICATION AND VALIDATION

I.1. INTRODUCTION

I.1.1. Scope

This appendix describes some techniques available for software verification and validation. They are grouped according to their properties. The choice of techniques and the way they are applied are clearly dependent on the case concerned, as is the importance attached to the outcome of the process. The description of each technique includes a summary of its advantages and disadvantages. The material is intended to be indicative of current methods and tools, and is not claimed to be comprehensive or to give complete guidance on methods of application.

I.1.2. Classification

The techniques are classified according to the following scheme:

(1) Review techniques (Section I.2.1):

- Inspection (I.2.1.1);
- Walk-through (I.2.1.2);
- Formalized descriptions (I.2.1.3);
- Symbolic execution (I.2.1.4);
- Program proving (I.2.1.5);
- Prototype execution (I.2.1.6);
- Quality measurement by metrics (I.2.1.7).

(2) Traceability analysis (Section I.2.2).

(3) Testing techniques (Section I.2.3):

- White box testing (I.2.3.1);
- Functional black box testing (I.2.3.2):
  - (a) Equivalence partitioning,
  - (b) Boundary value analysis,
  - (c) Cause–effect graphing;
- Load testing (I.2.3.3);
- Statistical testing (I.2.3.4);

— Reliability growth models (I.2.3.5);
— Timing analysis tests (I.2.3.6);
— Periodic functional tests (I.2.3.7).

(4) Safety analysis techniques (Section I.2.4):

— Failure mode effect and criticality analysis (FMECA) (I.2.4.1);
— Fault tree analysis (I.2.4.2).

Safety analysis techniques are included because they can be used to identify abnormal conditions and events (ACE), which include external events as well as computer system internal conditions. ACE analysis should serve as input to appropriate V&V activities.

I.1.3. Format of survey

The techniques are described with respect to the following attributes:

— *Goal:* a statement of the aim of using the technique;
— *Description:* a short description of the technique and explanation of how it is applied;
— *Advantages:* a description of the advantages gained by using the technique, e.g. ease of application, automated V&V procedures or completeness of results;
— *Conditions:* the conditions, if any, which are required in order to apply the technique;
— *Disadvantages:* a list of disadvantages connected with the use of the technique;
— *Tools:* the types of tools, often given as examples of available tools, which may be used to apply the technique;
— *References:* references to descriptions of the technique in textbooks, journals or other open literature or to projects where the technique has been used.

I.2. TECHNIQUES

I.2.1. Review techniques

I.2.1.1. Inspection

Goal: To detect anomalies and faults, mostly of predefined types, in software documentation. Typically, documents containing detailed design, code, test cases or test data are inspected.

Description: The inspection consists of checks of a document or a set of documents by one or more persons. These people form a team of limited size, in charge of detecting anomalies and faults of specified types. The possible types of anomalies and faults are related to the method or technique used to develop the documents and to the nature of the document. These anomaly and fault types could be related to non-functional requirements such as dependability, safety or security. The fault types are limited to a defined set. The inspection is carried out in four steps:

(1) The documents are distributed. They should preferably be of the same kind. The documents should be chosen to provide a global view of one activity (either all the documents of one phase or a representative sample). The readers should be chosen, on the basis of their experience, for effective anomaly and fault detection. They should have a comparable background to the authors of the documents. At least two but not more than four readers are recommended.

(2) A specified period is left for reading the documents and issue of comments. The inspection itself consists of reading the documents and checking against predefined anomaly and fault types. Usually, the set of anomaly and fault types is provided to the readers as a checklist. Each reader provides comments using a standard form. The comments are characterized according to the kind of anomaly and fault detected.

(3) A meeting is organized to analyse the comments and to determine their status and the need for corrections. The meeting is attended by all the readers. It may also be attended by the author or a representative of the team responsible for the documents. The meeting may be chaired by a moderator to balance decisions between the author and the readers. During the meeting, comments are classified and their relevance is discussed, i.e. whether an anomaly or error really exists and whether it is of a predefined type. Suppression of duplicate comments can be anticipated. Each anomaly and fault classed as genuine leads to a correction request. Responsibilities are identified for the correction of the documents and for the verification that all the corrections are made

(4) The documents are corrected and verification is performed to ensure that all the requested corrections have been taken into account. The responsibility for verifying the modifications and the corrections is generally assigned to a member of the inspection team. Time-scales for these actions should be set. Typically, the whole process takes less than two weeks and the meeting is ideally restricted to two hours. The meeting report should include in addition to the date and attendee list:

— The references to the input documents;
— A record of the checklists used by the readers, if any;
— A list of the minor comments, the associated suggestions and the deadline for corrections;

— A list of the major comments, the associated suggestions and the deadline for corrections.

Advantages: Inspection permits rapid detection of anomalies and faults in documents in both natural language and also in more formalized descriptions. When the kind of anomaly or fault is well known and simple to detect, readers with less experience than the authors may be used.

Conditions: The readers should have sufficient expertise. Known types of fault may be well identified but difficult to discover.

Disadvantages: Depending on the readers' expertise, the technique may be limited to the discovery of known predictable faults and would not then cover all aspects of the documents. The technique is best used to check a limited volume of documentation. It should be adapted for inspection of a large volume of documentation produced by a development team.

Tools: The analysis process is totally dependent on human activity and is not automated or significantly tool assisted.

References: [20].

Note: The concept of inspection has different meanings according to the application domain. For example, the definition in GAM T17 (the French military standard) [21] is: "Software quality evaluating activity that by examination, observation and measurements ascertains the degree of application of a standard to the conformance of a documentation or a CSCI (computer software configuration item) with a predefined quality clause. In the case of a document, an inspection can concern both the text and the form."

The US Department of Defense document DOD-STD-2167 [22] introduces the idea of product inspection dedicated to examining the delivered product in order to verify that it is complete with respect to the requirements.

Reference [23] recommends code inspection in order to confirm removal of coding faults and to provide assurance that the code conforms to the software code standards.

I.2.1.2. Walk-through

Goal: To check completeness and consistency of documents resulting from a creative activity (e.g. design) or requiring a high level of justification (e.g. test plans); and to reveal faults in all phases of program development. The method must be adapted for documents issued from a team.

Description: Walk-through consists of the presentation of documents by their authors to other persons, who are charged with questioning all aspects of the documents. Walk-through has the following four steps:

(1) The documents are distributed to the walk-through team members, who should evaluate and criticize the most important aspects of the documents with respect to their strengths and weaknesses. The walk-through team members usually belong to the development team. Normally, three to five persons are necessary for this evaluation.

(2) A planned period is left to the team to read the documents.

(3) A meeting is organized. During the meeting, the authors must present the documents completely, systematically demonstrating the accuracy of their logic, and the walk-through team members must put questions for further explanation or justification. The systematic browsing of the documents may be organized in different ways. One is to summarize the chapters sequentially. Exchanges between the team members should lead to a free and general discussion, to raise as many questions as possible. Discussion also helps team members to discover new problems. The walk-through meeting results in a list of recommendations for improving or correcting the documents. Responsibilities for taking the recommendations into account are identified. The recommendations may lead to corrections or justification of the original documents as correct.

(4) The documents are amended. A final verification gives assurance that for all recommendations a correction or a document justification exists.

Note: In Ref. [8], walk-through is the check of a sample of a chain of documents; it is similar to inspection.

Advantages: Walk-through can be organized in a short time. It is best used to check documents resulting from a creative activity. It can be applied effectively to design documents or documents requiring a high level of justification, such as test plans. The freedom allowed to the discussion (the only constraint is to discuss all the documents) permits coverage of all aspects of the documents and a check for completeness, consistency and adequacy. Walk-through does not require extensive preparation. The whole process can be concentrated in a few days. It also serves the purpose of sharing information among the development team. Therefore, it can be helpful in training newcomers.

Conditions: Walk-throughs must be foreseen and planned by the development team, in order to ensure the availability of readers and authors.

Disadvantages: The walk-through team should involve persons independent of the development team, to avoid bias in their criticism. As the preparation time is short, some critical parts may be missed or not sufficiently considered. The technique is best

used to check documents issued by a single author, and it should be adapted to tackle documents issued by a team.

Tools: The process is totally dependent on human activity. It requires in-depth understanding of the document and of the author's explanations and justifications. It is not automated or tool assisted.

References: [8, 24].

I.2.1.3. Formalized descriptions

Goal: To produce a complete, unambiguous requirements description and/or specification that can be used as a starting point for V&V activities.

Description: The term 'formalized description' is used for techniques providing a well defined notation (syntax) and model (semantics) suitable for expressing some aspects of the specification or design of software. 'Formalized' indicates here that the reasoning and manipulations which are used have some mathematical basis. In some cases the degree of formalism allows, in principle, the generation and discharge of all proof conditions needed to show that the description is complete and correct.

Formalized techniques include:

— Use of modern real time languages such as ADA and OCCAM.
— Application oriented graphical languages, with a fully defined notation (syntax) of functional elements each of which has a precisely defined semantics; examples include instrumentation and control function diagrams.
— General graphical languages with a formally defined semantics; examples include Petri nets and graphical data flow languages.
— Methods based on logic and algebra, including real time temporal logics, algebraic or axiomatic approaches and assertional calculus.

Advantages: The specification can be demonstrated to be both complete and consistent according to the properties offered by the language. Properties of the specification can be derived and checked against the requirements for correctness, including both written requirements and requirements as perceived by the system designers and plant operators.

The completed specification forms the basis for code verification using static analysis techniques. The specification can also be used as the starting point for continuation of the development using a formalized design method which subjects the steps of refinement to formal proof. The application of a formalized method, even if the proof aspects are not considered, quickly reveals missing information because of

the systematic approach to specifying the functions. Formalized description is a prerequisite for automatic code generation.

Conditions: A high level executable language must be available which has elements able to express precisely the kinds of problems to be solved. For example, the Vienna development method (VDM) is valuable for model based system descriptions but not always helpful for time based problems, where temporal methods such as the concurrent calculus system (CCS) are more valuable.

Disadvantages: Many of the methods require the specifier and validator to have some knowledge of the language used. The resulting system description is usually not easily intelligible to the traditional system engineer or software designer.

Tools: Examples of tools include:
— Development tools based on ADA and OCCAM;
— CAD tools for PLC programming;
— ASA+™, SAGA™, SPACE™ (specification and coding environment);
— For VDM: SPECBOX™, MURAL™, compilers' synchronous language tools.

References: [25–30].

I.2.1.4. Symbolic execution

Goal: To show, mathematically, the agreement between the source code and the specification, and to identify the relationships between output and input variables.

Description: The program is 'executed' by replacing the left hand side by the right hand side in all assignments. Conditional branches and loops are translated into Boolean expressions. As a result, the output variables are expressed as algebraic or logic functions of the input variables.

Advantages: No input data are needed. The results can be used in program proving.

Conditions: Formal specifications (with assertions) are necessary to check the results. Tool support is a necessity for all but the simplest programs.

Disadvantages: The algebraic expressions which result are often difficult to interpret. It can be difficult to analyse loops with a variable number of iterations. For many programs, the number of possible symbolic expressions is prohibitively large. The method is unlikely to detect missing paths.

Tools: MALPAS™, SPADE™.

References: [31, 32].

I.2.1.5. Program proving

Goal: To provide a basis for comparison with the program specification.

Description: Assertions are made at various locations in the program, which are used as pre- and post-conditions to various paths through the program. The proof consists of two parts. The first involves showing that the program transfers the pre-conditions into the post-conditions according to a set of logical rules defining the semantics of the programming language, provided that the program actually terminates (i.e. reaches its proper conclusions). The second part is to demonstrate that the program does indeed terminate (e.g. does not go into an infinite loop). Both parts may need inductive arguments. The result is the demonstration of the correctness of the program with respect to the assertions.

The method can be used together with program walk-through or inspection. At the present time, it is recommended only for the most critical of programs.

Advantages: A rigorous statement on correctness is achievable.

Conditions: Assertions must be determined and written into the program. This can be ensured if a suitable constructive development technique is used, such as that described in Ref. [33].

Disadvantages: Requires highly skilled and specialized personnel.

Tools: MURAL™.

References: [33].

I.2.1.6. Prototype execution

Goal: To check the feasibility of implementing the system against given constraints; and to communicate the specifier's interpretation of the system to the customer, in order to locate misunderstandings.

Description: A subset of system functions, constraints and performance requirements is selected. A prototype is built, using high level tools if possible. At this stage, constraints such as target computer, implementation language, program size,

maintainability and reliability need not be considered. The prototype is evaluated against the customer's chosen criteria and the system requirements may be modified in the light of this evaluation.

Advantages: The customer has a chance to assess the system at a very early stage and to verify the requirements and the understanding of the requirements. Resources are not wasted by building the wrong system.

Conditions: No special conditions.

Disadvantages: A significant amount of resources is required.

Tools: Development tools, and especially those supporting formal methods (e.g. SMALLTALK™).

References: [34].

I.2.1.7. Quality measurement by metrics

Goal: To predict the attributes of programs from properties of the software itself rather than from its development or test history.

Description: Techniques using metrics are used to evaluate some measurable properties of software, e.g. complexity, and to relate them to quality factors, e.g. reliability. Software tools are required to evaluate most of the measures. Some of the metrics which can be applied are given below:

- *Graph theoretic complexity:* this measure is based on the complexity of the program control graph, represented by its cyclomatic number.
- *Number of ways to actuate a certain module (accessibility):* the more a module is accessed, the more likely it is to be debugged.
- *Software science:* this measure computes the program length by counting the number of operators and operands; it provides a measure of complexity and estimates development resources.
- *Number of entries and exits per module:* minimizing the number of entry/exit points is a key feature of structured design and programming techniques.

In general, this technique is used on the source code. Moreover, if formalized languages are used, this technique can also be used at the level of design or specification.

Advantages: Some characteristics of the code structure can be verified without any execution of the code.

Conditions: The source code, and preferably the design or specification, correctly identified, must be available.

Disadvantages: As the correct relationship between measurable items (metrics) and high level attributes is still a matter of research and may change from application to application, this information may be misleading.

Tools: Static analysers are available as off the shelf products, e.g. LOGISCOPE™, QUALMS™, MALPAS™, CONCERTO AUDIT™, MacCabe's Tools™.

References: [31, 35–37].

I.2.2. Traceability analysis

Goal: To verify that the input requirements of an activity have been exhaustively taken into account by checking their trace in the resulting documentation; and to verify that all critical requirements have been identified and included through the development life cycle, i.e. from requirements analysis to final tests.

Description: Traceability analysis consists of identifying the input requirements and confirming how they have been taken into account by checking the target documents. For example, the analysis might check the system requirement documentation translation into the software requirement documentation, or the software requirement documentation into the software performance specification, the system requirement documentation into test files, etc.

Traceability analysis may be carried out by a single person. It requires an in-depth understanding of the input requirements and also of the method or technique used to develop the target documentation. When necessary, traceability analysis may involve a step to confirm the requirements in order to be certain to trace the actual requirements and not merely chapters or paragraphs of the input documentation.

A knowledge of specific probabilistic reliability allocation techniques may be necessary to carry out the traceability analysis of reliability requirements, and similar expertise may also be needed for other non-functional requirements. The analysis results in a document justifying that all the requirements have been taken into account properly. The document will identify, requirement by requirement, which part or component, or chapter of the documentation, fulfils it. The results are usually summarized in the form of a matrix of input requirements versus elements of the target documentation. The matrix can also be extended to include the source code.

Advantages: Traceability analysis confirms the links between the documents and, if applied during development, strengthens those links.

Conditions: A person must be available with a deep understanding of the requirements and of the way the target documents have been developed. Traceability analysis is made easier if the input requirements can be simply referred to without ambiguity. Some predefined rules should be followed; for example, one requirement per sentence or numbering of requirements allows a simple trace reference method.

Disadvantages: Traceability analysis is easier to apply for functional requirements. Non-functional requirements, such as safety, security and maintainability requirements, are less simple. These may be fulfilled by a certain structure or by a co-operative behaviour of the components. For these, the traceability matrix may often usefully be accompanied by a case by case justification.

Tools: Establishing the links is a process which is driven by human activities, which cannot be automated. However, tools exist which are able to capture both input documents and target documents and traceability links between them. It may be assumed that other tools can also be used, e.g. hypertext and electronic documentation tools.

References: [38].

Note: Traceability analysis is best used in the earliest phases of the development, e.g. software requirements traceability analysis and software design traceability analysis [7], but can also be used at any phase to verify that critical requirements have been taken into account.

I.2.3. Testing techniques

I.2.3.1. White box testing

Goal: To find program bugs, referring them to corresponding parts of the specification; and to check systematically that all corresponding requirements are met, and that there is no unreachable source code.

Description: The chosen part of the program is executed together with a test data set. This must be chosen from a global test data set which refers to the relevant section of the design specification or is provided by automated test data set generation. Appropriate test sets should be chosen to detect program faults

systematically. The paths through the code should be systematically traced. This can be done using code instrumentation. The instrumentation of source code can be assisted by various available compilers and testing tools. When the test routine is first run, all counters implemented via code instrumentation are reset. Starting the program module with a relevant data test set, certain counters will be incremented. Thus, a corresponding trace can be shown for that test set. The test routine is repeated until a specified set of structural elements of the program, such as a sequence of statements, all branches or a proportion of paths, has been covered. Thus, it is possible to show a certain test coverage measure or metric.

Test coverage is generally defined as follows:

C: All paths should be executed at least once. (C is path test coverage.)

C0: Every statement should be executed at least once. (C0 is statement or condition test coverage.)

C1–: Every branch which is not empty should be executed at least once.

C1: Every branch should be executed at least once.

C1+: Every branch should be executed at least once, but there is an additional condition that every possible result of a logical decision should be traced at least once. (C1 is branch test coverage.)

C2: Every branch should be executed at least once, but there is an additional condition: every loop (iteration) within a program will be executed at least once with the smallest and at least once with the largest number of iterations, and at least once if the loop body was not executed earlier.

The results are a set of test results covering the structure of the tested part of the software. An automatic comparison between planned and actual values of tracing counters is possible. Sufficient documents are generated to demonstrate that testing has been achieved to a level regarded as the state of the art.

Advantages: White box testing has the following features:

— The software module can be thoroughly, but not necessarily exhaustively, tested.
— Automation is possible to minimize the effort required to complete the whole test routine.
— Certain test coverage metrics can be produced.
— It can be shown that there is no dead, unused or unreachable source code within the software module.
— Relevant test data sets can be chosen out of the global test set according to source code documents.
— By looking at counter values, it is possible to mark a certain trace. This is of importance for analysing traces for software modules containing WHILE or CYCLE constructs.

Conditions: The specification of the program requirements must be well structured. Relevant test data sets should be generated, and automatic test set generation may be needed for effective testing.

Disadvantages: White box testing can only show that there is no unreachable source code hidden in the software module, but there is no real possibility of detecting faults or inconsistencies within the requirement specification. The state of the art, according to IEEE Std 829 [39], is to demonstrate a C1 test coverage measure.

With increasing complexity of software modules, the number of test runs increases more than exponentially. This causes the human effort needed to grow rapidly, so that normally only a 100% C1 test coverage measure can be shown. Normally, 100% path coverage is not feasible owing to the huge number of paths even in a simple program.

The correct outputs, as success criteria, can sometimes be difficult to determine in advance.

Tools: Tools for automated program analysis are available, e.g. LDRA Testbed™, ADA Test™. There are also a number of tools with automatic test data generators, mostly without the feature of automated test data set generation. New approaches applying expert system design methods are under development.

References: [24, 31, 39, 40].

I.2.3.2. Functional black box testing

Goal: To check whether there are any program errors in meeting particular aspects of the given specification and whether all specified requirements have been met.

Description: A test data set is derived solely from the specification and then the chosen software module is executed. The correctness of the software module is verified by testing against the specification.

Functional black box testing covers:

— Equivalence partitioning;
— Boundary value analysis;
— Cause–effect graphing.

Advantages: The software module is normally tested in its final form on hardware that is the same as or similar to that in which the program will run. Information is obtained if any specified requirement is missing. Demonstration that the chosen software module meets all specified requirements is possible. Black box testing helps to show whether or not the software module fulfils specified requirements.

Conditions: The specification of the software module must be well structured.

Disadvantages: It is very difficult to define relevant test data sets comprehensively enough to detect errors with maximum probability. This test method only detects non-conformance and it cannot be used to prove the correctness of a software module. The method is applied very late in the software life cycle.

Tools: Functional black box test tools are available.

References: [24, 31].

(a) Equivalence partitioning

Goal: To check whether there are any program faults in meeting particular aspects of the specification.

Description: The input domain of the program is divided into a number of equivalence classes, the elements of which are to be treated in a uniform way, according to the required actions given in the program specification. The result is the identification of valid and invalid equivalence classes for the inputs and corresponding outputs.

Advantages: The number of test cases is limited to a reduced subset of all possible input states.

Conditions: It must be possible to generate equivalence classes from the specification.

Disadvantages: It is difficult to determine the boundaries of the equivalence classes. Thus, isolated values, singularities and other special cases may be overlooked.

Tools: No specific tools are available for this technique.

References: [24, 31].

(b) Boundary value analysis

Goal: To check whether there are any program faults in meeting particular aspects of the specification.

Description: As in the previous method, the input domain of the program is divided into a number of input equivalence classes. The tests are designed to cover the boundaries and extremes of the classes. Choices often include maximum, minimum and trivial values, together with any likely error values. The result is documentation to show that the program behaves correctly across boundaries between different functional areas.

Advantages: The method checks that the boundaries of the input domain of the specification coincide with the boundaries of the input domain of the final program.

Conditions: It must be possible to generate equivalence classes from the specification.

Disadvantages: Identification of all boundaries is difficult. The possible combinations of all input classes generate a large number of boundaries to be tested.

Tools: No specific tools are available for this technique.

References: [24, 31].

(c) Cause–effect graphing

Goal: To check whether there are any program faults in meeting particular aspects of the specification.

Description: The distinct input conditions, as causes, and output conditions, as effects, given in the specification are identified and their relationships translated into a graph. This graph is used to derive test cases in a systematic manner, with each test case representing a combination of conditions. The result is the detection of incompleteness and ambiguity in the specification and a useful set of test data.

Advantages: Additional insight into the specification is gained through the different perspectives revealed by the translation into graphical form. Unwanted side effects are highlighted.

Conditions: It must be possible to generate equivalence classes from the specification.

Disadvantages: Boundary conditions are not adequately examined. The translation of the specification is expensive.

Tools: Some tools to automate the generation of test data from the graph are available.

References: [24, 31].

I.2.3.3. Load testing

Goal: To verify that the system will meet requirements to handle bursts of input signals or of messages, and to guarantee that specified response times are met.

Description: In systems sensitive to high load conditions due to bursts of input signals, e.g. caused by a plant transient or trip, load tests are performed to verify the ability to handle the input load according to the specification.

Load testing is based on the assessed worst case load of the system or its message transfer systems. In order to make the test realistic, it is important to analyse the number of status changes or of messages transmitted, relating them to time as accurately as possible, and margins should be specified.

For example, a trip might be tested by simulation of 500 status changes during the first minute, 200 in the second minute and 50 in the third minute after an initiating event. Far higher numbers for longer periods have been experienced in some plants. A test harness able to inject up to 1000 status changes per minute, with random timing, for 5 minutes, could allow suitable performance tests with a margin.

The burst of input status changes is simulated by devices which allow alteration of the number of status changes over the test period. Simulated operator actions and demands for displays may be included in the test. Message transfer systems should simulate transfer of many messages, with random timings and lengths, and with failure conditions such that the expected bandwidth limits are approached.

During tests of this type, the following key parameters can be measured or monitored:

— CPU load;
— Backlog in event recording;
— Number and identity of events not recorded or messages not transferred;
— Time delay for output action as a result of status change;
— Download of events to presentation devices, e.g. VDUs or alarm panels;
— Recovery parameters.

Advantages: This type of test is the only one that is able to verify that systems sensitive to bursts of input signals or demands are able to handle the increased load.

Conditions: Performance data must be specified in advance. The worst case of input load must be defined and a simulator designed which is able to generate the input load required.

Disadvantages: No disadvantages are known.

Tools: Input signal or message generators (simulators) are needed.

References: [41].

I.2.3.4. Statistical testing

Goal: To derive reliability figures on the final program performance.

Description: Statistical tests are tests of a program or a software module with test data selected with a statistical distribution of the input space of the object to be tested. The statistical distribution may be derived from:

— Information about the input space in the specific application;
— Assumptions about the fault distribution in the object to be tested;
— Assumptions about the frequency with which the different parts of the input space will be exercised.

In the case of protection system software, this technique requires extensive input from the possible trip scenarios to determine the portion of input space of relevance and the frequency with which its various conditions might be experienced.

The evaluation of statistical tests can be restricted to determine the probability of a correct response by simply monitoring the outputs of the object to be tested. Automated test harnesses have been found valuable for this. Side effects should be confirmed to be acceptable, such as:

— Unintended memory access;
— Unintended change of data in other software products;
— Wind-up of stack;
— Excessive processing time.

These effects must be investigated fully and shown to be acceptable using other means.

Advantages: Figures for correct response frequency can be derived without detailed knowledge of the operation of the object under test. The results can be combined with the output of other verification techniques.

Conditions: Each test run must be supervised so that every fault is detected. A complete specification is needed to derive the correct results to be expected from the

system under test. The number of tests must be large so that the results are statistically meaningful, i.e. sufficiently large that the predicted probability of a failure falls below a certain limit with a given level of confidence. Other test techniques should be used before this technique is applied, so that the number of errors detected during the testing is at a very low level.

Disadvantages: The number of test data scenarios required is potentially very large. Consequently, the production of the test data and the evaluation of the correct results become an enormous and demanding task. The effort to establish the part of the input space to be tested and the distribution of test data is also very large. For safety critical software, this technique can be used only to supplement other methods.

Tools: No specific tools.

References: [4, 17, 40].

I.2.3.5. Reliability growth models

Goal: To derive a reliability figure on the basis of recorded fault data.

Description: All software errors detected during debugging, testing and operation are logged. A reliability growth model is selected and the parameters in its failure probability density function are determined on the basis of this information. The results can be the probability density function of faulty execution, the expected number of remaining errors or the mean time to the next failure, etc., depending on the model used.

Advantages: If there is detailed knowledge of the test object and its corresponding failure probability density function, then only a moderate effort is required for this technique.

Conditions: Error data must be stored during all phases of the life cycle.

Disadvantages: An essential problem is to determine a relevant failure probability density function that will be significant to the test object. The results of this technique are directly influenced by the choice of failure probability density functions. This test method should not be used alone for V&V activities for safety critical software. The method is unlikely to be of use for safety critical applications. If a statistically meaningful number of errors were found, there would be some concern about the suitability of the software and development method.

Tools: City University (London) tool package.

References: [42].

I.2.3.6. Timing analysis tests

Goal: To prove that a software module or task meets specified time constraints. A safety critical command message must be generated correctly and meet given time constraints on when it is generated. This may be described by the term timeliness.

Description: Real time systems are characterized by reliability and predictability. So, for event driven systems, requirement specifications must be proved to be met. To complete this, run time instrumentation is required for the dynamics of run time behaviour. In some cases it may be possible to measure the system response directly, e.g. by the time delay between signal input and output, using simple measurement methods. Recently, more complex tools have been developed.

Advantages: Dynamic testing has the following features:

— Monitoring of system operation;
— Flexible management of primitive events (e.g. single alarms) based on on-line monitoring and maximal target system independence;
— High performance, fully distributed event recognition;
— Object oriented monitor programming language;
— Flexible data analysis and visualization capabilities.

Conditions: One of the timing analysis test methods is based on events and intervals monitored by a versatile timing analyser (VTA). To specify events and intervals, an event definition language was developed and is available for dynamic validation purposes. Thus, a detailed analysis at run time can be made in order to prove that the whole system will meet all time constraints. The test equipment is connected via an Ethernet deterministic local area network (LAN) with the complete hardware and software system. Different analyses and statistics can then be produced to check whether all system reactions meet the appropriate time constraints.

This method requires use of Ethernet IEEE 802.3 D (deterministic Ethernet, developed by the Institut national de recherche d'informatique et d'automatique in France), a VME bus system and a closely specified target CPU.

Disadvantages: This technique is very time consuming and it is usually not possible to demonstrate that the worst case has been identified. Dynamic analysis of real time systems requires run time instrumentation. The instrumentation causes the

implementation of certain patches in the source code to be analysed without stopping the task. In order to minimize delay times caused by the implemented 'probes' in the target software, three possible sizes of patches are provided: one at a minimal size, implementing a MOVE statement only, one at a small size to record one parameter, and one at a maximum size to record a certain number of parameters connected with a certain event. Depending on the CPU hardware chosen, the effective delay times caused by certain patches can be calculated.

Tools: VTA, combined hardware–software system, and object oriented event definition language (GOLDMINE).

References: [43].

I.2.3.7. Periodic functional tests

Goal: To ensure that the tested system is capable of performing its design function. The IAEA Safety Guide 50-SG-D8 [15] requires that safety related digital computer systems be designed for testing during the lifetime of the plant. Periodic tests can be performed automatically or manually.

Description: Periodic functional tests are part of the overall monitoring of items important to safety. All systems important to safety must have arrangements that permit test and calibration at intervals based on the equipment function, expected calibration drift, failure rate and other time dependent characteristics as well as on the reliability requirements.

Periodic functional tests are normally performed as black box tests. If possible, on-line or built-in test facilities should be included in the design. If the on-line test detects an error, the system must go to a fail-safe state and give an alarm.

A functional test of a computer system important to safety must, as a minimum, consist of one or more of the following activities:

— Inject test signals of an appropriate magnitude to give a suitable actuation of outputs or readouts, as required;
— Test automatically calculated set points or variables to verify responses to each input variable;
— Test the status and operability of interlocks, bypasses, test indications and annunciation circuits;
— Monitor the appropriate parameters during the test.

Tests should be overlapping, implying that the whole chain from sensor to actuator is tested.

Advantages: If implemented in the system design, periodic tests allow testing of the on-line system.

Conditions: The computer system must be tested under closely defined conditions corresponding to those in which it is expected to fulfil its intended functions. However, during testing, the configuration of the system can be changed in order to isolate the computer system from the rest of the system or to connect it to a test loop. After the test, the original configuration of the system must be restored. Periodic functional tests should be performed as on-line tests.

Disadvantages: Design faults and common cause faults are normally not detected by periodic functional tests.

Tools: Black box testing tools.

References: [4, 44–46].

I.2.4. Safety analysis techniques

I.2.4.1. Failure mode effect and criticality analysis

Goal: To verify that the system can meet, on a continuing basis, the specified safety requirements as stated in the system requirements specification. Failure mode effect and criticality analysis (FMECA) should be performed at appropriate stages in the design of the safety systems to be effective.

Description: To complete an FMECA, the consequences of each failure type must be analysed. Information gathered includes failure description, causes, defects, detection means, resultant safety consequences or other hazards, and recovery methods and conditions.

The principal procedure of FMECA can be described as follows:

— All possible random or common cause failures and their consequences are identified;
— All identified failures relevant to safety must either lead to a fail-safe system state or trigger alarm functions;
— The effect on shared supporting systems of a short circuit or a fire must be identified;
— The effect of failures in supporting systems must be identified and evaluated.

The failure analysis can be performed during or after the writing of the system requirements specification.

Advantages: This technique can identify potentially dangerous components.

Conditions: The documentation describing the design of the analysed part of the system must be available. Compliance with the requirements for independence must be established. The failure analysis must be performed by a team having the appropriate expertise.

Disadvantages: The technique may require considerable effort.

Tools: FMECA™, SOFIA™, RELIASEP™ and, dedicated to software: AIDL™, CECILIA™, FIABILOG™, MSI-REALITY™.

References: [14, 15, 47, 48].

I.2.4.2. Fault tree analysis

Goal: To identify conditions leading to a significant consequence (hazard) with a significant probability of occurrence. The probability can be quantified or estimated in a qualitative manner.

Description: Beginning with a system hazard, the analysis identifies or makes a hypothesis for the immediate and proximal causes, and describes the combination of environment and events that preceded the hazard, usually in the form of a directed graph or an 'AND/OR tree'. It is often accompanied by an 'event tree analysis' showing relevant event propagation information.

Advantages: The method is well known in safety analysis. Consequently, a variety of tools are available.

Conditions: The causes and consequences of an event must be identified and there must be data available on event probability.

Disadvantages: Fault trees and event trees are often very large. Temporal (dynamic) aspects are hardly ever treated. The method was developed for hardware systems; adaptation to software must be handled with care. Establishing the base events and the probability of outcomes can be problematic and become subjective rather than objective.

Tools: PHANISS™, Fault Tree™, Risk Spectrum™.

References: [49, 50].

Appendix II

SINGLE FAILURE CRITERION

II.1. INTRODUCTION

The single failure criterion is a term commonly used in a deterministic approach to ensure that a system will be available and operate when a demand is placed on it. It refers to a principle that safety functions shall be still possible if a single failure occurs. It specifies a simple design requirement for a certain minimum redundancy of a system or a group of equipment.

In the IAEA Code 50-C-D (Rev. 1), Section 329 [13], the single failure criterion is defined as follows: "An assembly of equipment satisfies the single failure criterion if it is able to meet its purpose despite a single random failure assumed to occur anywhere in the assembly. Consequential failures resulting from the assumed single failure are considered to be an integral part of the single failure."

The Code requires the single failure criterion to be applied to each safety group incorporated in the plant design which performs all actions required for a particular postulated initiating event (PIE) (Section 330). A method to test compliance is outlined in the Code. As the Code does not give details of the interpretation of the criterion, the IAEA has published a Safety Practice on Application of the Single Failure Criterion [45]. Although the Safety Practice does not deal expressly with the application of the single failure criterion to computer systems, it does give practical examples of its implementation for certain safety requirements.

The design of a nuclear power plant is required to consider all identified PIEs and ensure that the required safety functions can be accomplished with acceptable reliability for each PIE.

To achieve a high degree of confidence that the safety functions will be performed according to the Code, a number of measures are specified. In order to ensure and maintain adequate reliability of each safety system and component, three requirements must be satisfied:

(a) The independence of the safety system or component from the effect of the PIE must be maintained.
(b) The probability of failure of the components of the safety system must be minimized by adopting high quality standards and engineering practices for design, construction, operation and maintenance.
(c) The design of the safety system must be tolerant of failures without loss of the ability to perform the safety function.

II.2. APPLICATION OF SINGLE FAILURE CRITERION TO SYSTEMS IMPORTANT TO SAFETY

The single failure criterion is applicable to each safety group. A safety group is an assembly of equipment which performs all actions required for a particular PIE, in order that the limits specified in the design basis for that event are not exceeded. If a computer system is an active part of a safety group for which the single failure criterion is applicable, the criterion is also applicable to the computer system.

For the single failure criterion, two aspects must be considered:

(a) The single failure criterion is a basic safety principle and as such should be part of the overall system requirements. The single failure criterion will influence the design with respect to redundancy, independence, fail-safe design and auxiliary support. The ability of the design to fulfil the criterion must be verified after the system requirements specification phase.
(b) Implicit in the application of the single failure criterion is the requirement for testability of the system and components to which this criterion is applied. Otherwise undetected failures must also be taken into account.

One important aspect to be considered is that application of the single failure criterion only provides tolerance of single random failures. The occurrence of random failures is overcome by redundancy. However, redundancy achieved by means of identical channels may be defeated by common cause failures. Therefore, common cause failure analysis must be carried out in addition to analysis to show compliance with the single failure criterion.

Software faults do not normally result in random failures in service. They are due to errors in the design and implementation process. The result of such an error is a software fault, which may be challenged by system conditions or signal trajectories (which may appear in a random way), and then results in a system failure. Software faults should be detected and corrected during the development and V&V activities, before the system is put into service. An undetected software fault can result in coincident failure of two or more redundant channels which provide a safety function. This will occur if similar signal conditions or trajectories affect all channels and are incorrectly processed owing to the common fault. Analysis and tests related to the single failure criterion alone will, for these reasons, not be able to detect system sensitivity to software faults.

By definition, computer systems are active components. Active components may have active failure modes as well as passive failure modes. For computer systems, no distinction is made in reliability analysis between active and passive failures. A passive random failure is simply assumed to occur on demand and is

handled like an active failure. Active and passive failures may occur at any time during, shortly after or a long time after an accident.

Examples of random passive failures are given below:

— Limit value not changed in the database after a modification;
— Circuits failed closed or open in response to outputs from the computer system.

An example of a random active failure is unintended output from the computer system due to a faulty input signal.

Examples of random failures which can be either active or passive are as follows:

— Hardware failure preventing the computer from performing its intended functions (passive);
— Hardware failure causing the computer system to be actuated when not requested (active).

The method most commonly used to detect random failures is testing, and because of this, safety systems are normally tested periodically. In-operation test methods may also be used.

The reliability of a system not only depends on the degree of redundancy but is also a matter of the test strategy. During testing, the configuration of the system is often changed in order to isolate the computer system from the remainder of the system or to connect it to test circuits. After the test the original configuration of the system must be restored. Any of these operations may be the cause of faults, especially if such tests are done routinely. The system reliability analysis must allow for such testing and for the potential for such faults to reduce the reliability of a system very greatly by making it inoperable. The probability of unrevealed fault conditions must also be considered in reliability analysis. For this reason, optimal test intervals and procedures must be identified in relation to the operational possibilities and hazards.

Computer technology offers an advantage since, to a great extent, self-tests can be performed. Self-testing can take place on-line and will not usually interfere with normal operation of the safety system. This feature must be considered in the system requirements specification.

Periodic tests are normally performed as black box tests. The purpose of these tests is to verify that different combinations of input signals will result in expected reactions and outputs from the computer system.

Appendix III

EXPERIENCE

A number of countries have experience with the V&V of computer based systems for use in the safety and safety related systems of nuclear power plants. This appendix records, country by country, some of the experience gained in the V&V of software for nuclear systems, but is not intended to give a history of what has happened. Each section has the same structure, as follows:

— The standards and practices currently in use are given;
— The systems in use are identified;
— The role and safety classification of the systems are described;
— The verification and validation of the systems are discussed;
— The anticipated direction of development of V&V practices are discussed.

III.1. CANADA

Since 1971, all CANDU type NPPs have used computers for reactor and plant control, alarm annunciation, data display and logging. The basic elements of verifying and validating these systems included reviews of requirements and design documentation, code walk-throughs, white box and black box testing using regression test data methods, dynamic plant simulations and system acceptance testing. Evolution of these non-safety-critical systems from the earlier to the more recent plants has occurred more in the planning of system development and verification, in the scope and detail of documentation and in the introduction of independent validation than in the fundamental techniques.

The introduction of the use of computers for safety critical systems occurred in the early 1980s. Computer based systems were designed for process trip functions for the two safety shutdown systems. In the late 1980s, for the Darlington plant, the two shutdown systems were fully computerized, including trip logic, displays to the operator and computer aided testing. For the safety critical trip logic, formal mathematics based verification techniques and reliability testing were introduced.

Experience from verifying and validating the Darlington trip logic system has led to the development of a set of standards to form the basis for engineering and licensing software for safety critical and safety related nuclear applications.

III.1.1. Standards and practices

Experience with software engineering for safety critical applications showed that there is no generally agreed, measurable definition of acceptability for this type of software within the technical community. Furthermore, there is no widely accepted and adopted practice for the specification, design, verification and testing of safety critical software. To provide a basis for licensing software for nuclear applications, Ontario Hydro and Atomic Energy of Canada Limited (AECL) have jointly developed a set of standards based on the principles of IEC 880 [4] and on the experience gained in developing and licensing the Darlington plant shutdown systems and earlier computer systems. The framework of standards includes:

— A guideline to categorize software with respect to its failure effects on nuclear safety, with four categories defined, similar to IEC 1226 [6] but more quantitative;
— High level standards, independent of methodology, addressing the software engineering process for the three safety related categories;
— Sets of standards, procedures and guidelines which are dependent on methodology, for specific activities within the engineering process for each category;
— A guideline defining how to qualify predeveloped software.

The high level standards define the requirements for each stage of the development process from system requirements to final system integration and testing. The requirements for verification at each development stage are specified.

The outputs from each development process must be reviewed to confirm that they comply with the requirements specified in the inputs to that process. These outputs are written using mathematical functions. This is a requirement for safety critical software. The outputs are systematically verified against the inputs using mathematical verification techniques.

Typical V&V outputs for safety critical systems include reports on requirements review, design review, design verification, code review, code verification, hazards analysis, unit tests, subsystem tests, validation tests and reliability tests.

Quality attributes are specified for the outputs from each development stage, which provide a basis for verification review. These include completeness, correctness, consistency, verifiability, modifiability, modularity, structure, traceability and understandability. The specific attribute set which is used is dependent on the development stage.

These standards have been applied to software developed for safety critical applications for the Pickering plant and, in the Republic of Korea, for Wolsong 2, 3 and 4. They are also being used for non-safety-critical backfit applications on Bruce A.

III.1.2. Pickering digital trip meter (Ontario Hydro)

A digital trip meter, using an embedded microcontroller, has been developed to solve operating problems with the existing analog meters used on the heat transport high temperature trip system.

A software development methodology which meets the high level standard for safety critical software was used. This methodology is called the rational design process (RDP) and evolved from the techniques used on the Darlington computerized shutdown systems. The required behaviour of the software is defined by mathematical functions using notations with well defined syntax and semantics. The input/output behaviour is defined in tabular format.

The verification process, which was carried out by an independent team, included reviews at each stage of development. System requirements, software requirements, software design and code stages were verified, with the reviews being based on the quality attributes defined in the standard. Systematic mathematical verification was performed between the software requirements and software design phases and between the software design and code development phases. A code hazard analysis was done from a bottom-up perspective to identify potential design weaknesses.

White box testing was carried out to verify code compliance with the design and the software requirements. Black box validation testing and a reliability test based on 7000 randomly initiated input trajectories demonstrated compliance with the system functional and reliability requirements.

The trip meter was in the final stages of licensing review at the time of preparation of this publication. The development process proved effective in producing reliable code, with no significant errors being identified by the verification testing and none by the validation and reliability tests.

III.1.3. Wolsong 2, 3 and 4 shutdown systems SDS1 and SDS2 (AECL)

Each of the two independent, triplicated reactor shutdown systems SDS1 and SDS2 uses computers to perform the logic for the process system trip parameters, low and high primary heat transport pressure, low steam generator level, low flow, etc. These parameters have power dependent set points, which are adjusted by conditions at low power.

SDS2 uses a microprocessor based system. An object oriented computer language was used to produce the code using the RDP software development method described above.

SDS1 uses a process controller which embodies a graphical functional language for code generation. This graphical function notation was used to specify the software requirements and design. The software development process using this graphical function notation is called the integrated approach (IA). The notation is

mathematically precise but fairly easily understood. This facilitated formal review of the requirements and design by systems analysts and control engineers. A code disassembler was developed to allow verification of the executable code against the design specification.

Code hazard analysis was carried out on both SDS1 and SDS2. No hazards were identified, but some modifications were made to provide more resilience to potential hardware faults.

Verification testing against the software design description and software requirements was done using both white box and black box testing. Testing revealed no significant errors in SDS2 and none in SDS1. About 30% of the total effort was spent at the front end of the software development process, to provide assurance of completeness of the system level specifications and their correct translation into software requirements. It is considered that this was a direct contributor to ensuring that few errors were identified at later stages. Use of the graphical functional language for SDS1 greatly simplified formal verification.

Reliability testing was done using 10 000 test profiles which were generated by a system simulator to cover all PIEs for which the systems provide protection. Each trajectory used randomly selected initial values. The test inputs were applied to both the system under test and a validated test oracle. The results were compared automatically and discrepancies logged. Successful completion of the tests indicated that the target reliability was met to a confidence level of about 65%.

III.1.4. Future development

The current verification processes for safety critical software embody peer reviews and formal mathematical reviews during the specification, design and code generation phases. These are coupled with code verification testing during the code development phase. Validation testing and reliability testing confirm compliance with system requirements. Tools are used to help to generate the requirements document and to design in a consistent and traceable fashion. This aids the ability to review, and test case generators and test oracles reduce the testing effort. Further development is proceeding in these areas to improve the cost effectiveness of the V&V process, both for original code development and for subsequent maintenance. The development of validated functional languages is another area which will simplify verification of application code.

For lower safety categories, formal methods are not required. Verification is carried out during the software development process, with reviews of requirements and design documents coupled with code walk-throughs for control applications. Sets of deterministic test cases are developed for white box and black box testing and plant simulations used to verify dynamic characteristics.

III.2. FRANCE

III.2.1. Standards and practices

The classification of I&C systems and equipment, including computerized systems, which is adopted by Electricité de France (EdF) for a given standardized plant series, is defined by the French Society for Design and Construction Rules for Nuclear Island Components (AFCEN) at the level of French codes. This classification follows National Fundamental Safety Rules based on IAEA Safety Guide 50-SG-D1 [48]. For the N4 series, this categorization defines four classes: 1E, 2E, important to safety but not classified (IPS-NC), and non-safety. Classes 1E, 2E and IPS-NC cover systems, equipment and software important to safety: 1E is the safety class, and 2E and IPS-NC are the safety related classes.

The QA programme is based on the IAEA QA Code and Safety Guides [2].

The AFCEN code, French Design and Construction Rules for Electrical Equipment of Nuclear Islands, defines common applicable software engineering rules for software important to safety and availability of an NPP. For 1E software, this code requires the application of IEC 880 [4].

Moreover, EdF also uses IEC 880 as a baseline for 2E software, the applicability of its requirements depending on analysis of experience feedback when available (e.g. for PLCs).

III.2.2. Systems in use

III.2.2.1. 1300 MW(e) series

Since the end of the 1970s, EdF has installed programmable systems for the control and protection functions in the 20 nuclear units of the 1300 MW(e) PWR series.

Feedback from the operation of this equipment is very satisfactory. The hardware failure rates recorded, especially for the automatic reactor protection system (SPIN 1300), are stable and conform to the objectives of the design studies. No severe software errors and only one minor error have been reported during operation since 1984.

III.2.2.2. N4 series

For the N4 series of 1450 MW(e) PWRs, EdF has specially developed the use of software based systems by innovation in the following aspects:

— Widespread use of computerized architecture based on PLCs for the I&C systems;
— Adoption of an entirely computerized control room, where operation of the plant, in all circumstances, is carried out from computerized workstations.

The main control room system is configurable. Software was developed to run on existing proprietary support equipment, which is largely commercial in origin. This system is configured by data; display formats and logical equations for animation of control functions are defined by the operator.

The reactor protection system (SPIN N4) and the other systems which perform safety class 1E functions have been specially developed, incorporating feedback from the previous series and technological progress.

An off the shelf configurable system (PLC), which has not previously been deployed in NPPs, has been adopted to fulfil the safety related functions, i.e. those required in the medium and long term after an accident (class 2E functions). This PLC is also used for functions not important to safety.

Finally, an integrated CAD tool has been developed to ensure consistent management of the data of the I&C system.

III.2.2.3. Verification and validation of SPIN software

The SPIN 1300 software was developed according to a structured approach. Coding was completed manually after detailed design defining the structure of the code and the call to elementary functions. The software of the SPIN 1300 contains 40 000 instruction lines in 68 000 assembler.

An independent V&V team performed successive steps of checks:

— Inspection of documentation (including specification) and listings.
— Systematic unit tests.
— Validation tests of the system on a SPIN configuration representative of the complete actual system. These tests have been formalized; they took place after integration tests performed by the development team.

Such an approach allowed discovery of faults at the earliest possible stage in the software life cycle. This software has been in operation since 1984, and in 20 NPP units has given a total of more than 100 unit-years of operation.

The experience acquired through the development of the SPIN 1300 software was used in the development of IEC 880, published in 1986 [4].

For the N4 series, 68 000 microprocessors are used. The language C was chosen for performance reasons and because it is widely used in industry. The software was developed according to IEC 880.

A design tool called SAGA was used; SAGA allows for automatic code generation. The general principle is the following:

— Software design is described using 'SAGA views', which represent inputs, outputs, functions, logic operators and links between them. SAGA allows for a top-down description of the design by successive refinements and performs validity checks on types, links and completeness of the views and on their consistency.
— Design documentation is automatically generated.
— C source code is generated in the form of main components, including the data structures, instructions and calls to subfunctions. Interfaces with 'elementary' functions not supported by SAGA, e.g. hardware interfaces and numerical algorithms, are also automatically defined. The code generated by SAGA is traceable against the design.

Some 80% of code was automatically generated using SAGA, and there are about two hundred elementary functions. Because of the use of this tool, the number of errors discovered during the test phases was reduced by a factor of 10, in spite of the size of the code, which is much larger than for the 1300 series.

Following the same organization as for SPIN 1300, the main tasks performed by the V&V team for SPIN N4 are:

— Verification of the software requirements;
— Verification of the design, including the SAGA views and elementary functions;
— Verification of the code of the elementary functions;
— Unit tests of elementary functions;
— Validation tests of programmed units;
— Validation tests of interconnected systems.

In addition, a review process was run, involving development, V&V and SQA teams. More than a hundred reviews took place during the development, which followed the software quality plan.

III.2.2.4. Verification and validation of PLCs for 2E functions (N4 series)

For PLCs used for 2E functions, confidence in the software quality was based, in particular, on:

— Analysis of the development process against the recommendations of IEC 880;
— Analysis of the experience feedback;

— Additional functional tests.

EdF classified the required PLC functionalities into several classes, i.e. application, communication and system modules, in order to have a rigorous methodology for validation. For each class, the status of the supplier's tests was analysed. This was done on the basis of the test coverage, as shown by the supplier's documentation.

Experience feedback was utilized in two different directions:

— On one hand, operational times corresponding to PLC implementation were analysed with the modifications performed on the software, in order to find indicators of validation by experience feedback of the different software functions;
— On the other hand, a specific analysis was made for software functions which are not always used, although existing in every implementation of the PLC, since their execution depends on the final application and on the operational context.

This analysis allowed identification of software for which additional validation was necessary when the experience feedback was not sufficient.

EdF performed independent validation tests on software in order to check by sampling the efficiency of the preceding assessments. Performance tests are examples of tests repeated in this way.

III.2.2.5. Verification and validation of software supporting main control system (N4 series)

The support software used for the main control room system (classified as IPS-NC) was developed using an object oriented method and coded in ADA, in order to follow a structured and modular approach. Such an approach made it easier to define the V&V strategy. According to this strategy, verification tests covered tests from unit level to multitask and multimachine integration.

Validation tests on platforms were performed against the functional requirements, including human–machine interface requirements. Specific data sets were developed in order to check the functions and options offered and the limits of the system.

The main goal of the verification actions on-site corresponded to final checks to show that hardware and software correctly ran together. The software was also intensively used during the tests of the whole I&C system on-site.

III.2.2.6. Verification and validation of data

The V&V of data calls for procedures at the level of:

— *Design offices:* to check computerized information design and functional aspects;
— *Platform tests:* to check computerized system design and dimensioning;
— *On-site tests:* final check of the functionalities.

The V&V strategy rests on the concept of 'typical diagrams' as reusable processing modules. Once a processing module is checked for a typical diagram, it is checked everywhere it is used.

Verification of the data, including interface data between the main control system and the PLC based system, included:

— Checks implemented in the CAD tool, at the different steps of data production (completeness and consistency verification);
— Walk-through and inspection of data at the different steps of data production;
— Functional tests on the PLC based system platform (for data used at this level, including interface data with the control system);
— Functional tests on the control system platform (to validate the database used at this level and its integration in the control system);
— On-site tests, including tests on the occurrences of the different typical diagrams involved in the interface;
— Validation by use, as the I&C systems are largely exercised during the unit commissioning tests.

PLC configuration data items were verified according to a verification plan, in parallel with the validation of the PLC itself.

The data set plus control support system were validated at different levels:

— Static check of dimensioning, by tests of the actual data set on the EdF platform;
— Dynamic check of the behaviour.

The latter was performed by:

— Performance tests on a platform of the complete control system with the actual data set loaded (dimensioning functional scenarios, functional scenarios at limits);
— Specific monitoring of the control system during the on-site overall system tests (in particular during tests of the transient conditions).

III.2.3. Current and future states

For safety software, the current state is based on:

— Use of application oriented formal languages to generate code in an automatic way;
— Restriction of manual coding to a small number of short modules;
— Generalization of a reviewing process;
— V&V performed by teams independent of the development team;
— Readability of code and quality of the documentation;
— Use of IEC 880 [4].

For safety related software systems, which generally have larger codes, a trend is to use programmable electronic systems when possible. The main aspects appearing particularly important from the point of view of the V&V of existing systems are as follows:

— The user's requirements document should contain every requirement against which the system is verified and validated;
— The requirements must be realistic, presenting effective real needs and taking into account the status of the market;
— The human and financial resources required by assessments and tests should be taken into account;
— Formalization of the analysis should be based on feedback of experience.

For the future NPP series (REP 2000), the classification of the systems, equipment and software will follow the European Utility Requirements (EUR) for Light Water Reactor NPPs, based on the principles established in IEC 1226 (published in 1993) [6].

Recent standards or standards that will be published (e.g. IEC 1771 and IEC 1772 on control rooms [51, 52] and future supplements to IEC 880) will be taken into account for the V&V of software in NPPs.

The EUR and the results of the basic design studies of the French–German project for a PWR design named EPR will provide help in defining the rules for V&V of software in NPPs.

III.3. GERMANY

Requirements for safety and reliability of I&C functions and equipment are laid down in the KTA 35XX series (rules in nuclear techniques) of the Kerntechnischer

Ausschuß. However, there are no specific rules for computers, or even for software. Thus, some high level requirements for software must be derived from the KTA Rules. For more detailed requirements the German Standards Institute (DIN) has adopted the international standard IEC 880 [4] as a national standard. Thus, in practice, DIN IEC 880 is the main standard used for V&V of software, supported by some IEEE standards.

The latest digital I&C technology developed in Germany is characterized by:

— Graphical specification, syntactically and semantically fully defined;
— Automatic C code generation.

The German licensing procedure distinguishes between 'type assessment' (Typprüfung) and 'fitness for purpose assessment' (Eignungsüberprüfung). Type assessment is performed for components which are applied in the same way in all plants, e.g. a component which converts analog to digital signals, whereas fitness for purpose assessment considers plant specific properties of a system.

For a distributed computer system for I&C, where the I&C functions are implemented in software, the 'components' of the implementation tool were defined and assessed against the requirements of IEC 880 and some IEEE standards. Components are, for example:

— The functional elements of the graphical specification;
— Their representation in C;
— The function diagrams which are composed from the functional elements;
— The function diagram groups which are composed from function plans;
— The run time environment, which is a level between the function diagram groups and the operating system;
— The operating system itself (a specially designed system).

All these components are developed in phases, starting from the requirements specification, proceeding to computer (software) specification, design and implementation, and concluding with test phases. For each component and each phase an assessment was performed of the completeness, correctness and consistency of each phase. In addition, a traceability analysis was performed of each pair of component and phase, in which specific elements (especially all functions) were identified in each predecessor and successor document of the component.

A verification tool, which converts the output (C code) of the development tool to a higher level representation equivalent to the database content of the formal graphical specification, is under development. This tool will facilitate the fitness for purpose assessment. As the eventual code on the target machine is specific for the actual application, the assessment of the correct transformation from source code to machine code is also left to the fitness for purpose assessment.

Depending on specific agreements between licensees, licensing bodies and assessors, combinations of V&V methods such as inspection, walk-through, trace analysis, static analysis and dynamic analysis have been performed for conventionally developed and less critical systems.

III.4. HUNGARY

In Hungary, there are currently four nuclear power units and two research reactors in operation. There is currently no experience that can be considered as being of the depth and detail of the V&V activities described in this publication. In the past, I&C improvements could meet the reliability targets without the need for global and structural modifications to the systems installed in the Hungarian power plants. All the safety issues could be covered by gradual equipment replacement. It was also possible to correct some design deficiencies by simple modifications to the relay networks. Programmable devices for data acquisition from the secondary circuit were permitted. These would, however, fall into category C of IEC 1226 [6].

The I&C systems of one of the research reactors were reconstructed using PLCs and a great effort had to be put into the testing of the programmed controllers. The testing was not conducted by the Nuclear Safety Inspectorate (NSI), which only received and subsequently accepted the final report. On the Paks site, the current position for the four WWER-440/213 power units is that the I&C is not the most important contributor to the predicted core melt frequency. In probabilistic terms, the risk contribution of the electrical and I&C systems is 4%.

III.4.1. Systems in use

In 1993–1994, the NSI permitted the introduction of a new in-core monitoring system, VERONA, on two of the four power units. The operating experience of VERONA has been quite good in spite of the lack of formal V&V during its production. The staff of the Paks units concerned have already become familiar with the new systems, which provide all of the old and many new functions.

In the following years, the other two units were upgraded and a new in-core monitoring system was implemented. At the end of the commissioning phase the NSI conducted a check on the analog signal processing and the vital service functions, such as boot-up, foreground and hot standby redundant computer changeover, the data acquisition process, unit data display and operator warning functions.

The main weaknesses in the licensing process were as follows:

— There was no formal V&V, since the system is in category B of IEC 1226 [6];
— The functions were tested mostly during normal operation of the units, and there was no simulation of transients or accidents.

During one year of operation some data display functions were corrected to provide more convenience of use for the unit staff, and to eliminate some potentially ambiguous data display problems.

Two software bugs were detected and corrected:

— When isolating one of the six reactor coolant loops, the software could not handle the change in the plant configuration, which could lead to an error in the calculation of the thermohydraulic parameters;
— When changing from summer time to winter time, the fuel burnup accumulation routine could not handle the one hour change.

III.4.2. Future needs

Following discussions on the need for and nature of an I&C upgrade between the plant operator and the NSI, the Paks NPP management invited tenders for the replacement of the I&C systems. Up to date digital systems were offered for the upgrade of the reactor protection system (RPS) and emergency core cooling system (ECCS) control. These systems will be produced using highly automated hardware and software development systems.

The reliability targets that have been set in the preliminary task plan by the plant operator are as follows:

— Frequency of loss of actuation by the fast tripping system: $<10^{-7}$/year
— Frequency of unjustified trip, resulting from RPS error: $<10^{-2}$/year
— Frequency of loss of actuation of ECCS (all three channels fail to act simultaneously): $<10^{-7}$/year
— Frequency of unjustified ECCS actuation (one out of three): $<10^{-4}$/year

The NSI accepts these reliability targets, which are very high.

For the development of new safety codes, suitable to manage the licensing process for an RPS, the following activities were planned:

— The elaboration process of the I&C safety codes would be accelerated;
— For practical purposes, the NSI would be in touch with regulatory bodies from the relevant potential vendors' home countries;
— The NSI would aim for the safety code committee to consider a minimal set of reference standards for handling digital and programmable I&C.

The following reference standards were expected to be among those included in the set of documents: IEC 880 [4], IEC 987 [1], IEC 1226 [6], IEEE Std 730 [10], IEEE Std 829 [39], IEEE Std 830 [16] and IEEE Std 1012 [7].

III.5. RUSSIAN FEDERATION

III.5.1. Standards and practices

Each NPP of RBMK or WWER type in the Russian Federation is equipped with data acquisition and information presentation computer based systems. The systems are Scala for RBMKs and Hindukush for WWERs. There were no rules approved for V&V of software for such systems during their development. Traditionally, unit computer systems were considered as part of the reactor and required to comply with the existing set of standards for hardware. In spite of the absence of formal approved rules, the Scala and Hindukush software verification activities were performed in accordance with the national experience of developing software. V&V methods such as walk-through, white box testing and black box testing were used, although not in the same form as described in this report. The software validation phase was performed according to Russian standards on the development of technical systems, which require phases similar to factory acceptance testing and site acceptance testing. During these phases, comprehensive testing of the system must determine whether the integrated system executes the assigned mission, providing completeness, correctness, accuracy and consistency with the system specification.

III.5.2. Systems in use

The systems mentioned above are mainly in-core monitoring systems and operator support systems and do not carry out the control functions, which are systems of category B [13]. Scala and Hindukush perform data acquisition and information generation functions using a range of measured and calculated data. They also support information presentation, mainly via mnemonic schemes and VDUs.

Software of the systems performs calculation of parameters such as:

— Power in each fuel assembly;
— Flux sensor burnup;
— Neutron flux distribution;
— Operating reactivity margin;
— Critical power margin in each fuel assembly.

The systems also provide indication of equipment and device conditions, including equipment failure, and presentation of selected process parameters or deviation signals from set points. These are in the form of a digital printout and a mnemonic scheme of reactor core sensor locations. Some subsystems perform the function of recording parameters before and after an event.

During system operation, on-line diagnostics of system equipment is carried out to determine availability and the possibility of system reconfiguration. The rules for periodic testing of the software are defined in the maintenance documentation.

III.5.3. Current and future states

Since 1991, regulations on licensing and V&V of safety related software have been applied to new projects. The administrative procedure for V&V and licensing has been approved, but technical guidance on this procedure has not yet been formulated.

III.6. SWEDEN

There are a significant number of computer based systems in use in Swedish NPPs, but to date only parts of the plant protection systems have been equipped with software based systems.

Newer plants, such as Forsmark 3 and Oskarshamn 3, have hybrid control rooms where alarm functions and recording of historical data are performed in large plant computer systems. These systems have been classified as category B systems.

Older units are facing an upgrade phase, which will result in a much larger usage of computer based systems, including some category A applications. For example, digital three channel turbine governors, which are category B systems, have been installed at several units.

III.6.1. Standards and practices

The three main principles associated with the implementation and V&V of software in Sweden are presented below:

— Complex category A systems are developed according to IEC 880 [4]. V&V activities are performed in order to meet the requirements of this standard. Up to 1994, ANS/ANSI 7-4.3.2 was applied.
— Commercially available off the shelf category A components and complex category B and C systems are purchased or developed according to ISO 9000 [19]. V&V activities are performed in order to ensure that the requirements of ISO 9000 are satisfied.
— Commercially available off the shelf category A, B and C equipment with embedded software, e.g. transmitters, relays and sensors, should meet the requirements of ISO 9000. The V&V requirement for this type of equipment is partly determined on the basis of past operating experience.

III.6.2. Systems in use: Forsmark 1 and 2

Reactor neutron flux measurement systems for the low and medium power range have been installed in Forsmark 1 and 2. These systems also form part of the RPS, i.e. they will cause the reactor to trip (scram) under certain conditions. These systems are category A systems in terms of the IEC 1226 [6] classification and were developed from similar systems already in use (e.g. at Tokai 2 in Japan and at Ringhals 1 in Sweden).

The systems installed in Forsmark 1 and 2 were developed according to ANS/ANSI 7-4.3.2 and IEEE Std 730. The verification of each phase in the development process was performed by manual inspection of the output documents against the corresponding input documents. The systems were subjected to further analysis before going into service, the objective of which was to verify that the systems were not affected by input signal loading or interrupts.

In order to validate the systems, extensive functional tests were undertaken both at the manufacturer's works and on-site. To evaluate the new systems under operational conditions, they were operated for one year in parallel with the old systems.

III.7. UNITED KINGDOM

There are a significant number of computer based systems in use at NPPs and reprocessing facilities in the UK. These systems range from primary plant protection equipment through vital control equipment to information and alarm systems. Some important experience was gained from the use of computer based control of refuelling equipment and from controlling post-trip switching of electrical supplies at the most recent AGR type plants. Both applications would today be category A according to IEC 1226 [6].

The current position on verification, validation and justification of computer based systems in the UK is outlined below by reference to two category A RPSs.

III.7.1. Standards and practices

The licensing approach in the UK follows from the Nuclear Installations Inspectorate (NII) Safety Assessment Principles for Nuclear Plants [53], which in the case of software for safety critical systems currently requires a special procedure to be followed.

No specific national standards for software are applied to nuclear plants, although IEC 880 [4] is used as a major reference. Current practice is often compared with the requirements of UK military standards and of Ref. [23], and Nuclear Electric

plc (NE) has an internal standard in trial use. IEC 1508 [54] on the functional safety of process control computers is also used as a reference for good practice.

III.7.2. Systems in use

III.7.2.1. Dungeness B single channel trip system

The single channel trip system (SCTS) forms part of the primary protection system (PPS) for the Dungeness B AGRs and was introduced to provide protection against fuel channel inlet gag failure, which could cause high coolant gas temperatures and so lead to fuel failure. The system works by monitoring the coolant temperature in each channel and generating a trip demand if the temperature rises above a trip level. The system is a category A system in terms of the IEC 1226 classification.

The system was developed from a similar system that had been in service in a passive mode on the Dounreay PFR. The changes for the AGR application were made in an incremental manner, following a structured life cycle of specification, design, coding, integration and testing. The verification of the development was by manual inspection of the output documents of each phase against the corresponding input documents. The manual activity was possible in this case because of the small size of the system and it allowed some of the powerful features of the architecture to be exploited.

Validation was by a combination of testing and manual inspection, i.e. the test plans were verified against the requirements and the results of the test activities were used to validate the system.

The system was subjected to further scrutiny before going into service. Static analysis was used to compare a formal version of the specification, derived from the natural language requirements, against the source code. This was done using the Malvern program analysis suite (MALPAS). The PROM contents were reverse engineered and compared with the source code. The system was also subjected to an independent safety review that covered the functionality, the QA integrity of the development route and the use of PFR operating history to demonstrate the safety of the use of interrupt in the trip and vote processor units. The safety justification drew heavily on the properties of the ISAT^{TM} architecture and its powerful hardware error revealing properties.

III.7.2.2. Sizewell B primary protection system

The Sizewell B PWR, constructed and operated by NE, uses a computer based PPS, which has been installed and is fully operational. The overall protection is

provided by two systems of diverse technology: the PPS and the secondary protection system (SPS). The protection systems cause reactor trip, whereby the power supplies to the control rods are removed, enabling the rods to enter the core under gravity to terminate the nuclear reaction. The PPS uses microprocessor based logic, while the SPS uses magnetic core logic (laddic) technology.

The design requirement for the PPS was to provide reactor trip and engineered safety feature (ESF) actuation for all design basis faults, including faults with a very low estimated frequency of below about once in 1000 years. The design requirement for the SPS was to provide reactor trip and ESF actuation, in parallel with the PPS, for faults with a frequency in excess of about once in 1000 years. The reliability targets of the two systems were therefore set at 1 in 10 000 for the PPS and 1 in 100 000 for the SPS. This can be achieved by hardware, but the highest standards of software production available at the time and of demonstration of integrity had to be applied to provide assurance of this.

As initial plant designer, the National Nuclear Corporation Ltd (NNC) conducted various reviews of the design of a prototype PPS in the period 1980–1985, prior to the decision to place a contract with Westinghouse Electric Corporation. These reviews concluded that the PPS design was appropriate for the PWR protection system. The NNC was required to perform a manual verification of the PPS software for NE. This verification covered all of the PPS software documentation, code and data written by Westinghouse.

The acceptance process for the operating licence required very extensive review and analysis of the PPS equipment and software. The acceptance process required documented demonstration of quality. This was shown in design, in the depth of coverage of assessment activities and in comprehensive analysis and testing. An extensive programme of software verification was carried out on the PPS software to detect and remove any errors generated in the design and coding phases. This comprised several diverse techniques:

— Verification of all the PPS software documentation, code and data by Westinghouse's internal independent V&V team;
— A fitness for purpose review of all PPS software documentation, code and data, carried out on behalf of NE by the NNC;
— A separate NE independent design assessment (IDA) of the principal design information;
— Confirmatory assessment of the software using MALPAS (consisting of static and semantic analysis tools for all safety software), conducted under the TACIS programme of the European Union;
— Confirmatory assessment of the software, by comparison of the source code with the executable image stored in the system's program memory, using an automatic tool developed by the Technology Division of NE.

The normal practice for fully documented factory acceptance testing and site commissioning was followed. NE also performed dynamic testing of a prototype, using simulated fault scenarios with random variation of the test trajectories, on an equipment system designed and produced by Rolls-Royce and Associates. A requirements trace was undertaken by the NNC for NE of the testing done against the original functional requirements.

The Westinghouse design performs all functions in a deterministic, repetitive manner. There is a standardized approach to all modules, using a common design methodology. The common functions, which provide system level software functions, allow the application programs to be simple and direct. Their use allows each subsystem to perform its input/output duties without an operating system and in a way which demands minimal involvement from the application level.

At the application and subsystem levels, the software design has a common uninterruptible deterministic logic flow. Configuration data modules are divided into two main sections, and for each application or subsystem the data items are used to configure the relevant included functions as follows:

— The common functions employed by the application or subsystem (diagnostics, maintenance console, loop time synchronization, memory clearing, error reporting and shared memory) are all covered by the use of the agreed standard template, such that this section of configuration data always follows the same layout.
— For the local database and the interface boards used in the application or subsystem (analog input/output, digital input/output, data links, data highways and diagnostic boards), all of the relevant data structures for each function are duplicated with configuration values entered.
— The calibration data files are a list of calibration values, which may require modification during the lifetime of the system. Facilities to amend some modifiable sections of these data are provided through a maintenance console.

The total size of the source code for the reactor protection functions, excluding comments, support software for the autotesters and communications to other systems, is around 100 000 unique lines. A typical processor contains between 10 000 and 40 000 lines of source code, of which about half are typically from common functions, and the remainder form application code. In addition to the executable code, the PPS incorporates around 100 000 lines of configuration and calibration data per guardline associated with the reactor protection functions.

The system is decomposed into modules, each of which is understandable in its own right. This modularity facilitates verification, testing and subsequent maintenance. The design objectives included flexibility and clarity, as well as modularity, together with the simplification of verification. The design objectives had not required minimization of the total amount of code. Other features, such as the

diagnostic checks, also increased the number of source code lines from the minimum needed for the functions.

The NNC fitness for purpose review and verification covered all documents, code and data, which were meticulously reviewed and verified. The factors which allowed this were:

— Careful development of a detailed written verification plan, followed throughout the task;
— Use of a computer based file comparison tool, ensuring that the task was not unnecessarily increased by the verification of duplicated sections more than once;
— Systematic reporting of all findings, under a carefully controlled procedure;
— Confirmation of clearance of all relevant findings.

The NE IDA was conducted by a separate part of NE with totally separate management. The review covered all system and application code, together with the hardware and its protection requirements. Comprehensive and detailed reports were provided to the project management and were then assessed and responded to formally. Some detailed design changes were made to respond to concerns expressed by the IDA team. Specific concerns were that the requirements were expressed in natural language, that no full demonstration of the validity of the compilation and link loading system was available, and that the language used had not the full and unambiguous definition which modern practice for safety systems requires. Therefore, the project management decided to perform a full code analysis using MALPAS, with compliance analysis, which was able to show the full match of the requirements to the source code. To show the match of the loaded object to the source code, a full code decomposition and comparison process was undertaken.

An additional part of the overall validation of the system was the use of a dynamic testing harness. The purpose of this was to show that the performance of the system was functionally satisfactory for a wide range of simulated transients. This was done for about ten representative fault scenarios where safety interests demonstrated that the reliability of the PPS software was critical to the safety case. The system was made and the detailed testing performed by Rolls-Royce and Associates, and it was shown successfully that satisfactory protection was provided for about 50 000 individual transients. This therefore provided a high level of assurance of correct functionality, even for detailed variations in the transients foreseen by the reactor safety dynamic studies.

Sizewell B has required a major effort by Westinghouse, NE, the NNC and others to ensure that complex protective functions are implemented accurately in code and that there is a high degree of confidence in this. The quantity of software involved in the PPS is large, and it had been thought by some that it would be too large to verify

meticulously and in its entirety. The methods of design and implementation match the qualitative criteria for the highest reliability systems. There should therefore be every confidence that the PPS software is fit for its purpose of providing reactor protection with very demanding reliability requirements.

The full details of the acceptance process are provided in Ref. [55].

III.7.3. Current and future states

The systems licensed so far have allowed a suitable body of information to be built to allow a route to licensing to be established. For small systems, the route allows for manual inspection for the verification process and validation by testing. The safety justification draws heavily on the use of static analysis for comparing the source code with the requirements to ensure that these have been correctly implemented. This is complemented by the comparison of the PROM contents with their source code. The systems are subject to a period of passive operation or extensive operational testing before going into service.

There is not expected to be a significant change in the route, since it appears to be effective, although it is anticipated that the use of formal, mathematical methods such as VDM, B and CCS will increase, with discharge of the associated proof obligations.

The lower safety category systems are subject to manual inspection for verification and this is supported by validation testing. In many cases the systems are used for interlocking to control the movement of spent fuel, and the computer based system is supported by a hard-wired system. These systems are not currently subject to static analysis or source code comparison, although some code analysis is done.

III.8. UNITED STATES OF AMERICA

In the USA, there are currently more than a hundred NPP units operating. These are owned or operated by more than 45 separate utilities or energy companies. The designs include both PWR and BWR units of many different sizes, vintages and manufactures. The control systems are from a variety of vendors as well. This has resulted in an environment where almost every power plant is unique, with the exception of a handful of ‘sister plants’. Regulatory review is made more difficult by such a mosaic of equipment and system types.

The USA does not use the multitiered categorization criteria defined by standards such as IEC 1226 [6]. In the USA, systems are categorized as safety (Class 1E) or non-safety. Regulators are only concerned about reviewing those systems which are categorized as Class 1E. Utilities have much freedom with regard to installing digital devices in non-Class-1E systems.

III.8.1. Standards and practices

The legal requirements for NPPs are contained in Title 10 of the US Code of Federal Regulations, Chapter 50 (10 CFR 50). Appendix B to 10 CFR 50 establishes the basic requirements for a comprehensive QA programme. It is a policy statement that QA is an important issue in the design and implementation of safety systems. Upgrades to digital devices that use software as a key element have been made for a number of years under Rule 10 CFR 50.59, which basically states the conditions for the utility to replace one system with another on a one for one basis. As long as there are no changes either in functionality or in the licensing document, no regulatory review is needed. For systems that do not meet these criteria or that have safety applications, the designs are reviewed by the Nuclear Regulatory Commission (NRC).

However, appendix B by itself is not sufficiently detailed to enable it to be used for the review of software and digital systems. In the past the NRC used such standards as IEEE Std 7-4.3.2 (1982) and IEEE Std 603 (1981) to define ways for implementing the QA requirements of appendix B for digital systems.

IEEE Std 603 has established the functional design criteria for the power, control and instrumentation portion of safety systems of nuclear power generating systems. However, the guidance of IEEE Std 603 from 1981 did not address the unique nature of software for safety systems in systems design, stage by stage testing, overall performance assurance and documentation. IEEE Std 7-4.3.2 was developed in 1982 as an extension of IEEE Std 603 for computer based safety systems incorporating software to implement safety functions. NUREG 1.152 [56] states the NRC's endorsement of IEEE 7-4.3.2 and it became the regulatory basis for accepting software in safety systems. Conformance of the safety system software to these regulatory documents is to be addressed by the applicant's submittal, or equivalent criteria are to be presented to justify their absence.

As more experience was gained in the installation of digital systems and their review, both of these documents were updated. IEEE Std 603 was updated in 1991 [57] and IEEE Std 7-4.3.2 received an extensive rewriting in 1993 [17]. As before, IEEE Std 7-4.3.2 (1993) is an extension of IEEE Std 603 (1991), and must be used in conjunction with its companion document. The structure of IEEE Std 7-4.3.2 has been changed to make it congruent with IEEE Std 603, essentially expanding para. 5.3 of IEEE Std 603 and incorporating many of the recent lessons learned in the application of software to safety systems. It now considers the aspects of digital technology in safety systems, including:

— Computer based safety systems;
— Software to implement safety functions;
— Design and qualification of software;

— Integration of hardware and software;
— Importance of V&V in software development.

IEEE Std 7-4.3.2 makes reference to other documents, including IEEE Std 828 [58], IEC Std 880 [4], IEEE Std 1012 [7] and ANS NQA 2a (Part 2.7), issued by the American Nuclear Society [59].

The emphasis during V&V reviews of safety software is on methodology and process, since the procedures for developing the safety software, rather than the design attributes and characteristics, have been shown to be the dominating factor in safety software quality. Although significant use has been made of software in safety systems, there has been no consensus on what particular design attributes constitute 'good' or 'reliable' safety software. The criteria of IEEE Std 7-4.3.2 are based on an active V&V programme throughout the development process, providing assurances that the development is carried out in a manner to produce highly reliable software.

III.8.2. Systems in use

Extensive use has been made of digital technology in the USA, primarily in support systems such as:

— Feedwater control;
— Radiation monitors;
— Diesel generator load sequencers;
— Core performance calculators.

Some limited applications have been made to safety systems and RPSs, the most prominent being that of the Westinghouse Eagle 21 RPS installed at the Zion plant.

III.8.3. Current and future states

The nuclear industry and regulators have become more aware of the issues raised by the installation of software based digital devices in NPPs. The NRC, taking a conservative approach, suggested in a draft Generic Letter that the use of any software in a safety system (Class 1E) was an unresolved safety question. In response to this, the Electric Power Research Institute (EPRI) in conjunction with the Nuclear Management and Resources Council (NUMARC) initiated a project to develop guidelines for licensing digital upgrades in NPPs. After review by industry representatives and the regulators, the result was published in 1993 as EPRI TR-102348 [60].

The EPRI report is a guide for addressing design and licensing issues that arise when installing digital systems in safety systems. It is intended to supplement existing

guidance given in NSAC-125 and NSAC-105. Although the NRC has not formally endorsed these guidelines, their use is advised. Some of the major design issues that are covered in the EPRI report are:

— Design;
— Commercial dedication;
— Environmental effects;
— Human factors;
— Failure analyses;
— Diversity.

The report deals with licensing issues but acknowledges the importance of V&V. It does not include any V&V requirements directly but incorporates them by reference to annex E of IEEE Std 7-4.3.2-1993 [17]. The EPRI has since published a guideline on V&V [61].

REFERENCES

[1] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Programmed Digital Computers Important to Safety for Nuclear Power Plants, Standard 987, IEC, Geneva (1989).

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

[3] 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).

[4] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Software for Computers in the Safety Systems of Nuclear Power Stations, Standard 880, IEC, Geneva (1986).

[5] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Standard for Software Verification and Validation Plans, ANSI/IEEE Std 1012-1986, IEEE, Piscataway, NJ (1986).

[6] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear Power Plants — Instrumentation and Control Systems Important to Safety — Classification, Standard 1226, IEC, Geneva (1993).

[7] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Standard for Software Verification and Validation Plans, IEEE Std 1012-1987, Piscataway, NJ (1987).

[8] REDMILL, F.J. (Ed.), Dependability of Critical Computer Systems, 1, Guidelines Produced by the European Workshop on Industrial Computer Systems, Technical Committee 7 (EWICS TC7), Elsevier Applied Science, London and New York (1988).

[9] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, ISO 9000 Standard Series: Quality Management and Quality Assurance Standards, Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software, ISO 9000-3, Geneva (1990).

[10] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Standard for Software Quality Assurance Plans, IEEE Std 730-1989, Piscataway, NJ (1989).

[11] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Software for Computers Important to Safety for Nuclear Power Plants, Draft IEC 880-1, Geneva (1996).

[12] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Glossary of Software Engineering Technology, IEEE Std 610.12-1990, Piscataway, NJ (1990).

[13] INTERNATIONAL ATOMIC ENERGY AGENCY, Code on the Safety of Nuclear Power Plants: Design, Safety Series No. 50-C-D (Rev. 1), IAEA, Vienna (1988).

[14] INTERNATIONAL ATOMIC ENERGY AGENCY, Protection System and Related Features in Nuclear Power Plants: A Safety Guide, Safety Series No. 50-SG-D3, IAEA, Vienna (1980).

[15] INTERNATIONAL ATOMIC ENERGY AGENCY, Safety Related Instrumentation and Control Systems for Nuclear Power Plants: A Safety Guide, Safety Series No. 50-SG-D8, IAEA, Vienna (1984).

[16] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1993, Piscataway, NJ (1993).

[17] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations, IEEE Std 7-4.3.2-1993, Piscataway, NJ (1993).
[18] INTERNATIONAL ATOMIC ENERGY AGENCY, Code on the Safety of Nuclear Power Plants: Governmental Organization, Safety Series No. 50-C-G (Rev. 1), IAEA, Vienna (1988).
[19] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, ISO 9000 Standard Series: Quality Management and Quality Assurance Standards, ISO, Geneva (1990).
[20] FAGAN, M.E., Design and Code Inspection to Reduce Errors in Program Development, IBM Syst. J. No. 3 (1976).
[21] MINISTERE DE LA DEFENSE, Méthodologie de développement des logiciels intègres, GAM-T17 (V2), Ministère de la défense, Paris (1989).
[22] DEPARTMENT OF DEFENSE, Defense System Software Development, DOD-STD-2167, Dept of Defense, Washington, DC (1988).
[23] RTCA INC., Software Considerations in Airborne Systems and Equipment Certification, Doc. No. RTCA/DO-178B, Washington, DC (1992).
[24] MYERS, G.L., The Art of Software Testing, Wiley, New York (1979).
[25] BILLET, A., ESMENJAUD, C., "French feedback of experience on NPP safety software qualification", Proc. 2nd INEC Int. Conf. on Control and Instrumentation in Nuclear Installations, London, 1995, pp. 19–21.
[26] BJORNER, D., JONES, C.B., Formal Specification and Software Development, Prentice-Hall, Englewood Cliffs, NJ (1982).
[27] MORGAN, C., Programming from Specifications, Prentice-Hall, Englewood Cliffs, NJ (1990).
[28] JONES, C., Software Development: A Rigorous Approach, Prentice-Hall, Englewood Cliffs, NJ (1980).
[29] BENVENISTE, A., BERRY, G., The synchronous approach to reactive and real-time systems, Proc. IEEE **79** 9 (1991).
[30] SOULAS, B., ASA+: Outil pour l'analyse compartementale des systèmes temps réel, Rec. Actes J. d'électron. **95** (1995) 253–257.
[31] DEPARTMENT OF COMMERCE, Software Validation, Verification, and Testing Technique and Tool Reference Guide, Natl Bureau of Standards, Washington, DC (1982).
[32] DICK, A.J.J., Equational Reasoning and the Knuth–Bendix Algorithm — An Informal Introduction, Rep. DOC 84/1, Imperial College of Science, Technology and Medicine, London (1984).
[33] GRIES, D., The Science of Programming, Springer-Verlag, New York (1981).
[34] BOEHM, B.W., A spiral model of software development and enhancement, Computer (May 1988) 61–72.
[35] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Standard for a Software, Metric Methodology, IEEE Std 1061-1992, Piscataway, NJ (1992).
[36] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Std 982.1-1988, Piscataway, NJ (1988).

[37] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Std 982.2-1988, Piscataway, NJ (1988).
[38] DALE NELSEN, E., System Engineering and Requirement Allocation, IEEE Computer Society Press Tutorial, Los Alamitos, CA (1990).
[39] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Standard for Software Test Documentation, IEEE Std 829-1983, Piscataway, NJ (1983).
[40] EHRENBERGER, W., "Statistical testing of real time software", Verification and Validation of Real-Time Software (QUIRK, W.J., Ed.), Springer-Verlag, Berlin (1985).
[41] FORSMARKS KRAFTGRUPP AB, Computer Load Tests, Technical Report FQ, Rapport 94/48, Forsmarks Kraftgrupp, Östhammar (1995).
[42] MUSA, J.D., IANNINO, A., OKUMOTO, K., Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York (1987).
[43] SCHMID, U., Monitoring distributed real-time systems, Real-Time Systems No. 7 (1994).
[44] INTERNATIONAL ATOMIC ENERGY AGENCY, Surveillance of Items Important to Safety in Nuclear Power Plants: A Safety Guide, Safety Series No. 50-SG-O8 (Rev. 1), IAEA, Vienna (1990).
[45] INTERNATIONAL ATOMIC ENERGY AGENCY, Application of the Single Failure Criterion: A Safety Practice, Safety Series No. 50-P-1, IAEA, Vienna (1990).
[46] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Periodic Tests and Monitoring of the Protection System of Nuclear Reactors, Standard 671, IEC, Geneva (1980).
[47] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Analysis Techniques for System Reliability — Procedures for Failure Mode and Effects Analysis (FMEA), Standard 812, IEC, Geneva (1985).
[48] INTERNATIONAL ATOMIC ENERGY AGENCY, Safety Functions and Component Classification for BWR, PWR and PTR: A Safety Guide, Safety Series No. 50-SG-D1, IAEA, Vienna (1979).
[49] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Fault Tree Analysis (FTA), Standard 1025, IEC, Geneva (1990).
[50] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Dependability Management, Part 3: Application Guide, Standard 300-3-1, IEC, Geneva (1991).
[51] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Verification and Validation of Control Room Design of Nuclear Power Plants, Standard 1771, IEC, Geneva (1995).
[52] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Visual Display Unit (VDU) Application to Main Control Room in Nuclear Power Plants, Standard 1772, IEC, Geneva (1995).
[53] HEALTH AND SAFETY EXECUTIVE, Safety Assessment Principles for Nuclear Plants, HSE Information Centre, Sheffield (1992).
[54] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems, Standard 1508, IEC, Geneva (1998).
[55] Proceedings of a Forum on Safety Related Systems in Nuclear Applications, London, 1992, Royal Academy of Engineering, London (1992).

[56] NUCLEAR REGULATORY COMMISSION, Criteria for Digital Computers in Safety Systems of Nuclear Power Plants, NUREG 1.152, US Govt Printing Office, Washington, DC (1996).
[57] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Systems, IEEE Std 603-1991, Piscataway, NJ (1991).
[58] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard for Software Configuration Management Plans, IEEE Std 828, Piscataway, NJ (1990).
[59] AMERICAN NUCLEAR SOCIETY, Computer Software for Nuclear Facility Applications, ANS NQA 2a, Part 2.7, American Nuclear Soc., La Grange Park, IL.
[60] ELECTRIC POWER RESEARCH INSTITUTE, Guideline on Licensing Digital Upgrades, Rep. EPRI TR-102348, Palo Alto, CA (1993).
[61] ELECTRIC POWER RESEARCH INSTITUTE, Guideline for the Verification and Validation of Expert System Software and Conventional Software, Rep. EPRI TR-103331, Palo Alto, CA (1995).

CONTRIBUTORS TO DRAFTING AND REVIEW

Andersson, O.	Forsmarks Kraftgrupp, Sweden
Ashwell, R.	Atomic Energy of Canada Limited, Canada
Baldwin, J.A.	AEA Technology, United Kingdom
Cook, A.B.	Canatom Ltd, Canada
Elshin, A.V.	Science and Research Institute of Technology, Russian Federation
Esmenjaud, C.	Schneider Electric S.A., France
Ets, A.R.	Smartware Associates Inc., United States of America
Faya, A.	Atomic Energy Control Board, Canada
Ficheux-Vapné, F.	Electricité de France, France
Hamar, K.	Nuclear Safety Inspectorate, Hungary
Joly, M.	Electricité de France, France
Kersken, M.	ISTec, Germany
Kossilov, A.	International Atomic Energy Agency
Leret, E.	Electricité de France, France
Mertens, U.	Siemens AG, Germany
Neboyan, V.	CONSYST Co. Ltd, Russian Federation
Schildt, G.H.	Technical University of Vienna, Austria
Shinohara, Y.	Nuclear Power Engineering Corporation, Japan
Wall, D.N.	AEA Technology; subsequently Nuclear Installations Inspectorate, United Kingdom

Welbourne, D.	National Nuclear Corporation; subsequently consultant, United Kingdom
Yates, R.L.	Nuclear Installations Inspectorate, United Kingdom

Consultants Meetings

Vienna, Austria: 6–10 September 1993, 21–25 February 1994;
Paris, France: 25–29 September 1995

Advisory Group Meeting

Östhammar, Sweden: 14–18 November 1994