IAEA-TECDOC-1328

Solutions for cost effective assessment of software based instrumentation and control systems in nuclear power plants

Report prepared within the framework of the Technical Working Group on Nuclear Power Plant Control and Implementation

INTERNATIONAL ATOMIC ENERGY AGENCY IAEA

December 2002

The originating Section of this publication in the IAEA was:

Nuclear Power Engineering Section
International Atomic Energy Agency
Wagramer Strasse 5
P.O. Box 100
A-1400 Vienna, Austria

SOLUTIONS FOR COST EFFECTIVE ASSESSMENT OF SOFTWARE BASED INSTRUMENTATION AND CONTROL SYSTEMS IN NUCLEAR POWER PLANTS

IAEA, VIENNA, 2002
IAEA-TECDOC-1328
ISBN 92–0–119902–3
ISSN 1011–4289

Printed by the IAEA in Austria
December 2002

FOREWORD

The introduction of software based instrumentation and control (I&C) systems for use in nuclear power plants, mainly due to I&C modernization activities, has raised many issues of safety and economics. Many of these issues have been raised in the IAEA Technical Working Group on Nuclear Power Plant Control and Instrumentation (TWG-NPPCI) meetings and by other organizations, such as the OECD and the European Commission. One increasingly important issue is the need for engineering solutions to justify them for the cost effective assessment and deployment of software based I&C systems. To address this important issue, the IAEA put together the Co-ordinated Research Project (CRP) on Solutions for Cost Effective Assessments of Software based I&C Systems.

The overall objective of the project is to facilitate the cost effective assessment of software based I&C systems in nuclear power plants. This is necessary to address obsolescence issues, to introduce new beneficial functionality, and to improve overall performance. The engineering solutions developed in this CRP will contribute to this overall objective.

The objective of the CRP was reached through co-ordinated research and collected experience in the areas of project management; requirements specifications; use of software explicitly designed for nuclear applications, use of commercial off the shelf (COTS) products, generic pre-qualification of systems and components; safety and reliability enhancements; verification and validation; and licensing impact. This TECDOC is the result of the research and collected experience put together under this CRP. The CRP participants gave presentations on their work performed as part of this CRP at the various meetings of the group.

The first meeting of the CRP was held in Vienna on 8–12 November 1999 in which the participants developed the objectives, scope, and outline of this report. The second meeting was held in Halden, Norway on 4–8 December 2000 and the final meeting was held in Vienna on 29 October–2 November 2001. At this meeting, the final research results were incorporated into the report. A small consultants meeting was held in Vienna on 18–21 February 2002 to finalize the TECDOC.

This report was written for use by utilities, designers, suppliers, licensing authorities, and contractors who are involved in the modernization of I&C systems with software based systems. Special thanks are due to J. Naser of Electric Power Research Institute (EPRI), USA, who chaired the meetings, and to B. Wahlstrom of the Technical Research Centre of Finland who, along with J. Naser, largely contributed to the report. The IAEA officer responsible for preparing this report was Ki Sig Kang of the Division of Nuclear Power.

EDITORIAL NOTE

The use of particular designations of countries or territories does not imply any judgement by the publisher, the IAEA, as to the legal status of such countries or territories, of their authorities and institutions or of the delimitation of their boundaries.

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

1.1. BACKGROUND

The introduction of software based instrumentation and control (I&C) systems for use in nuclear power plants (NPPs) has raised many issues of safety and economics. One increasingly important issue is the need for engineering solutions to support the cost effective assessment and deployment of software based I&C systems. The IAEA has played a significant supportive role in the modernization of I&C systems in nuclear power plants by addressing and resolving a number of the issues arising from the introduction of software based I&C systems through its recent efforts of upgrading their safety standards, producing technical reports, and organizing specialists meetings. In order to address the need for the engineering solutions mentioned above, additional research and development work is required.

The introduction of software based I&C systems to nuclear power plants is driven partly by the lack of a viable technical alternative to address I&C system obsolescence issues to support the plant for its economic life, and partly by the cost and functionality advantages software based I&C systems appear to offer compared with analog and other hardware-based systems. The adoption of software based systems has not been as widespread or as free from difficulty as expected. There are a number of reasons for the lack of widespread implementation and less than expected benefits so far from new software based systems. These reasons include evaluation and acceptance costs and risks. Some of the reasons for this are the failure to produce a correct requirements specification, the failure to appreciate the rigor required in the safety demonstration, and the lack of good engineering solutions to facilitate implementation and evaluation of software based I&C systems.

In many cases, the root cause has been a failure to efficiently provide the evidence and to complete the range of evaluation and acceptance exercises, which are not just limited to safety concerns but also includes reliability and availability that are necessary for a successful project. This Coordinated Research Project (CRP) addressed the issue of good engineering solutions to improve cost effective assessment and deployment of software based I&C systems.

The reasons for the need of software based I&C systems in nuclear power plants include:

- Existing analog systems are becoming obsolete, spare parts are not available, and many suppliers have withdrawn from the nuclear industry;
- Software based I&C system technology is now the main stream I&C technology for process and other industries; therefore, analog technology supply has become only a niche market;
- The demand by the nuclear industry for new I&C systems is too small for it to support suppliers providing analog systems solely for nuclear use;
- Software based systems will be more cost effective over the lifetime of the system than alternative technologies, particularly when long term support is required; and

- Software based systems are the only means of providing the additional functionality and operational improvements being sought by the utilities for more effective operation.

The economic issues are particularly important when considering a system modernization in today's increasingly competitive and cost conscious deregulated electricity market. However, even in regulated markets it is important to minimize costs.

Despite the perceived benefits and the pressures driving the adoption of software based technology, there continues to be significant hesitation by the utilities to introduce software based I&C systems because of the financial and regulatory risks involved in their implementation. These risks arise from a lack of knowledge about and uncertainty in the consequences of adopting a new technology by both the implementers and the licensing authorities. For example, software based I&C systems introduce new situations characterized by factors that include:

- The highly integrated functionality which requires a more rigorous specification as compared to the partitioned functionality found in analog systems;
- The uncertainty of the content of the test plans required to complete the acceptance of the validation of software based systems;
- The unique properties of the software based technology arising from the digitization of input and variable space;
- The means of providing an acceptable demonstration of adequate system dependability and software correctness; and
- The management of the different operational and safety issues that arise when introducing a new technology.

A major element required to overcome these real and perceived difficulties is the production of engineering solutions for the development of a cost effective means of identifying and evaluating the options available, and of the production of the I&C system and necessary evidence to support the evaluation of its fitness for the purpose intended in the nuclear power plant.

There are several subject areas for which engineering solutions are required and will be discussed. These include:

- Establishing good project management solutions to support cost effective assessment and deployment;
- Establishing the requirements and, in particular, the form and content of the requirements specification documents to ensure that the requirements are correct and complete and that the unique properties of digital systems and technology have been considered when defining the functional requirements and interfaces;
- Identifying the strategy to be adopted, e.g., to make use of a commercially available or commercial off the shelf system (COTS) products, to have a custom built system, or some other of the wide variety of options available;
- Generically pre-qualifying systems and tools for multiple applications;

- Obtaining generic acceptance by the regulator for assessment methods and processes;
- Taking advantage of safety and reliability enhancements of the system, e.g., with the use of good software design practices, on-line monitoring and diagnostics, inclusion of fault detection features, separation, redundancy, diversity, and fault tolerant designs ;
- Identifying the choice of method or combination of methods to be used to support the cost effective assessment, e.g., ACEs (abnormal conditions and events), FMEA (failure modes and effects analysis), PSA (probabilistic safety assessment), and tools to support these assessments;
- Identifying effective verification and validation methods and tools; and
- Discussing related licensing issues and experiences along with guidelines developed to support cost effective assessment.

An important part of the CRP activities has been the evaluation of the benefits obtained by using the engineering solutions for the activities including:

- Evaluating and supporting claims that can be made on the basis of operating experience, principally determining what experience is relevant;
- Evaluating the level of effort that should be invested and its dependence on the safety category of the system;
- Use of on-line diagnostics and monitoring and other capabilities made possible by software based technology, which can be far more extensive than in systems built using analog technology; and
- Tool support for the design and evaluation processes.

Two other aspects are important. First is the dependence of the rigor of the processes and demonstrations on the safety and economic significance of the system (i.e., a graded approach), and second is the potential impact of regulatory involvement on the whole process. These, in turn, are influenced by the high speed of development of software based technology for I&C systems.

There continues to be uncertainty with both suppliers and utilities as to the regulatory requirements to be met when licensing a software based I&C system. Even when regulatory requirements are available, there remains uncertainty and inconsistency in their interpretation. Although not part of the CRP, since regulations are country dependent, stabilization of the licensing environment in each country is necessary for widespread implementation of software based systems. The engineering solutions to improve the evaluation process, when generally accepted, will help the realization of this stabilization.

1.2. OVERALL OBJECTIVE

The overall objective is to facilitate the cost effective assessment of software based I&C systems in nuclear power plants. This is necessary to address obsolescence issues, to introduce new beneficial functionality, and to improve overall performance. Taking into account the general development of I&C systems in non-nuclear industries in which the change in technology from analogue (hard-wired) to digital (software based) techniques has occurred

several years ago, the same technology is being introduced now into more and more nuclear power plants for both safety and high safety-relevant applications.

To successfully implement software based I&C systems, it is necessary that the safe operation of such software based systems in nuclear power plants has to be proven during the licensing process in order to gain the acceptance of licensing authorities, as well as acceptance in the plant itself. Suitable assessment methods in order to demonstrate safe and reliable functionality of the system, in particular for software related functionality, have to be applied. Efficient methods to meet the required high quality of the assessment within acceptable costs are needed. The engineering solutions of such assessment methods evaluated or developed in this CRP will contribute to this overall objective. The content of this report is based on presently applied methods, the results of direct research under CRP, and by the analysis and evaluation of other relevant information and experience.

1.3. SPECIFIC RESEARCH OBJECTIVE

The specific research objective is of this CRP is to assess engineering solutions which will cost effectively produce the information required to facilitate evaluations and acceptance of software based I&C systems. This is being done to assure that the software based I&C systems are of acceptable quality for use in safety and safety related applications in nuclear power plants, but is also relevant for not important to safety systems that still need to have high reliability and availability for economic reasons. These solutions cover two basic topic areas. First are the solutions that improve the assessment methods themselves to improve the quality of the assessment and/or to reduce the cost of the assessment. Second, are solutions that can be done in earlier phases of the project, or in earlier projects, to improve the quality and/or to reduce the costs of the assessment.

The objective will be reached through coordinated research in the following areas:

- Project management;
- Requirement specification;
- Systems explicitly designed — under nuclear power plant licensing requirements;
- Commercially available or COTS products for nuclear power plant applications
- Generic pre-qualification of platforms, devices, applications, and tools; and generic pre-acceptance of methods and processes to support cost effective assessment;
- Safety and reliability enhancements;
- Verification and validation methods/techniques for assessment; and
- Licensing issues.

1.4. RELATIONSHIP TO OTHER WORK

The IAEA has performed work in several areas related to the modernization of I&C systems in nuclear power plants. This work includes the CRP described in this report as well as subjects such as:

- Modernization strategies and guidelines
- Requirements specifications;
- Architecture and basic technical solutions for software based I&C systems;
- Verification and validation methods applicable to software based systems;
- Software anomalies;
- Safety and reliability assessment;
- Quality assurance; and
- Management.

These topics are discussed in a collection of IAEA reports [10, 14].

The recommendations presented in this report are to be read in combination with the other IAEA publications that give more detailed guidance in these topics. This report does not repeat general information provided by the reference reports but provides strategy and methodology recommendations for cost effective assessment of software based I&C system.

1.5. SCOPE AND USERS OF THIS PUBLICATION

This report describes approaches for the use of cost effective solutions to achieve high quality assessment of software based I&C system. These systems include safety applications, safety related applications, and not important to safety applications. It is obvious that assessment is required for safety and safety related systems to obtain the high quality required by safety regulations. However, from operation and economic points of view, high reliability is needed for many not important to safety systems as well, which leads to additional assessment activities. The utility may decide to do more assessment work than what is required by safety regulations for safety and safety related systems due to other requirements, such as economic ones requiring very high reliability and availability.

The same type of assessment strategies, methods, and techniques used for safety and safety related systems can be used for not important to safety systems, as desired by the utility. This has nothing to do with regulatory requirements. High reliability of these not important to safety systems is essential to ensure high availability of the nuclear power plant and to minimize the challenges to safety systems in the plant. High availability is essential in many cases for the economic viability of the plant. In fact, experience has shown that the economic impact of some not important to safety systems under normal operating conditions is substantially higher than that of safety and safety related systems. It is for this reason that the utility may elect to do a higher level of acceptance than is normally considered adequate for not important to safety systems. Therefore, the scope of the assessments in this report includes, but is not limited to, safety and reliability assessments.

The following life-cycle phases are considered to be in the scope of this project:

- Feasibility study;
- Specification of requirements;

- Computer system design (excluding the detailed design study);
- Qualification;
- Verification and validation; and
- Licensing.

The scope of assessments in this report includes assessment of the product (system) and the assessment of the processes used to develop, implement, operate, and maintain the product. The definition or description of the processes themselves that are used for system development, implementation, operation, maintenance, and modification activities are out of the scope of this report.

The scope of this report includes both the assessment of the software aspects of software based systems and the assessment of the integrated software and hardware of the system. This integrated assessment is essential for the determination of the acceptability of the system for its intended application.

An important issue for software based I&C systems and their associated interfaces is human factors and their assessment. It is important to do this assessment; however, this is not in the scope of this report and will be left to other sources.

This report is oriented to the following categories of users:

- Utilities or owner/operators;
- Independent experts;
- Designers;
- Suppliers;
- Third party implementers; and
- Licensing authorities.

The term independent experts refer to independent organizations or individuals who are brought in to perform or evaluate an assessment from an independent point of view. The term designers include the full range of designers for methods, tools, platforms, systems, etc. The term third party implementers refers to an organization or individual, that is neither the supplier or the user of the system, but is responsible for qualification and implementation of the system.

1.6. COST EFFECTIVE ASSESSMENT

The term assessment is used here to mean a process, which is performed as a basis for acceptance or rejection of a system. For a safety or safety based system the safety of the system is the main target for the assessment. However, economic assessment issues are also appropriate.

The term cost effective as used here means that the cost of the assessment should be as low as possible while assuring that the objective of the assessment is fulfilled. For a safety assessment, this means that the assessed system fulfils the requirements of the required safety level and, when appropriate, the economic requirements.

To be cost effective for assessment means that the cost of the assessment process itself is minimized. It also means that the development of the system and other activities should be performed in a manner that facilitates the assessment, and thereby limits the assessment costs. This report addresses both of these two basic aspects for cost effective assessment.

To obtain a cost effective assessment, the assessment should be performed in all life-cycle phases of the system development, from concept to system installation and maintenance plans. In this way it will be possible to make sure that failures made in earlier phases are revealed in that phase, when it is less costly to correct, rather than in an assessment done in later phases. This prevents the necessity of the development having to be redone from the earlier phase, which would increase both the development and assessment costs.

1.7. IMPORTANCE OF ASSESSMENT BY OTHERS

The activities required for assessment of a system may be able to be reduced based on relevant assessment work done by others. This relevant assessment work can come in eight areas. These areas fall into two major groups based on whether the work was done in the home country of the proposed assessment or not. These areas are described as:

- Work done under the regulatory requirements of the home country
  - Generic methods and processes formally approved by the home country licensing authority
  - Generic qualification or specific qualification of equipment, systems, and applications formally approved by the home country licensing authority
  - Industry consensus generic methods and processes developed consistent with regulatory requirements but not formally approved by the home country licensing authority
  - Generic qualification or specific qualification of equipment, systems, and applications in a manner consistent with the regulatory requirements but not formally approved by the home country licensing authority
- Work done under the regulatory requirements of a different country
  - Generic methods and processes formally approved by the licensing authority in a different country
  - Generic qualification or specific qualification of equipment, systems, and applications formally approved by the licensing authority in a different country
  - Industry consensus generic methods and processes developed consistent with regulatory requirements but not formally approved by the licensing authority in a different country

– Generic qualification or specific qualification of equipment, systems, and applications in a manner consistent with the regulatory requirements but not formally approved by the licensing authority in a different country

In each of these areas, being familiar with what has been done and taking advantage of it, can lead to significant reductions in the amount of assessment work required and hence make the assessment more cost effective. This familiarity will also help determine the level of risk in gaining acceptance, as well as give an indication of how much more work will be required. In the first case where the method or process has already been formally accepted by the licensing authority in the home country, all that should be needed for the use of that method or process to be acceptable to the licensing authority is to demonstrate that the method or process was used correctly. In the second case where the equipment, system, or application has already been formally accepted by the licensing authority in the home country, all that should be needed is to gain acceptance of the equipment, system, or application in the plant is to demonstrate that it has been implemented correctly and that it is being used in a consistent manner as was approved by the licensing authority.

For the next case, where the consensus methods or processes were developed consistent with the regulations in the home country and the technical basis for them has been documented, this technical basis should be used to play a major part in gaining acceptance of the use of the method or process. Additional work will most likely be required to gain acceptance of the use of the method or process but that should be considerably less than required without the already done technical basis. Similarly, for the next case where a piece of equipment, system, or application has gone the through qualification steps and the results are documented, this qualification activity and documentation should be the cornerstone of the licensing case for regulatory acceptance. Again, additional effort will probably be required, but that should be considerably less than required if starting without the previous qualification activities.

The next four cases can be considered together even though it would be reasonable to expect that something formally accepted by a licensing authority in a different country would have more complete information and require less work than for the case where it has not been formally accepted. In these four cases, the first thing to do is become familiar with what has been done and what the regulations are that apply to the work. Again this should give a good idea of the level of risk for gaining acceptance. Next, would be to determine how the work done applies to the regulations in the home country. This will help determine which of the activities already done are relevant, and which additional activities are required. The previous work and its justification can be used to support the licensing case. Again it is expected that if there is relevant previous work, it will reduce the overall assessment effort required to gain regulatory acceptance.

1.8. CONTENT OF REMAINING SECTIONS

A brief definition of the contents of the following sections is given here. Section 2 discusses project management as it applies to assessment. In particular, it considers the necessary management activities to support successful assessment activities. Section 3 discusses the requirements specification of software based systems. The accuracy of these requirements is essential for the success of the system and have a major impact on the amount of assessment work that must be done or re-done after problems are fixed. It discusses tools to

help make sure that the requirements are consistent, accurate, etc. Section 4 discusses systems that are designed explicitly for use in nuclear power plants under the nuclear power plant licensing requirements. It includes the assessment necessary for the qualification and acceptance of this category of systems. Section 5 discusses the use of commercially available or COTS hardware and software (products) for applications in nuclear power plants. It describes the special assessment activities needed to bring the acceptance level of these products up to what is acceptable in the nuclear industry. Section 6 discusses the concepts of pre-qualification of platforms, devices, applications and tools to reduce the assessment activities when a system is implemented in a nuclear power plant. This section also discusses the pre-acceptance by the licensing authority of methods and processes for cost effective assessment. Section 7 discusses safety and reliability enhancements that can be made to the processes and systems to reduce assessment activities. Section 8 gives a detailed description of techniques and tools to perform the verification and validation of software based systems. Section 9 discusses the impact that regulations have on assessment and the good practices to minimize the problems in satisfactorily meeting the regulations. Section 10 gives the conclusions and recommendations of this report.

2. PROJECT MANAGEMENT

2.1. INTRODUCTION

Good, knowledgeable, and actively involved management of I&C modernization activities, including cost effective assessment, is essential for the success of the project so that it achieves all of the desired benefits in the most cost effective manner possible. When the plans are put together for a modernization project, the plans for the assessment work and acceptance criteria must be an integral part of those plans. The desire is, of course, to be able to do the assessment activities in as cost effective manner as possible, while meeting the desired acceptance criteria for the new I&C system. This desire is applicable independent of whether the utility, the contractor, or a third party (or any combination of these) does the assessment work in the project. Note also that this could all be done in one organization and the end user could be considered the utility or customer and the developer and/or the assessor could be considered the contractor.

It is important that the management plan defines clearly, completely, and unambiguously the assessment requirements and acceptance criteria at the beginning of the project. These requirements and criteria should be agreed upon in advance within the utility. When the system to be assessed has to be licensed, then agreement on the regulatory requirements should be reached together with the licensing authority. These agreements will minimize the likelihood of expensive re-assessment activities to satisfy all of the stakeholders of the system, including the licensing authority when appropriate. The management plan should include a quality assurance plan and configuration control plan.

It should be noted that the utility can always do more assessment than the minimum level for safety required by the licensing authority for safety and safety related systems, or by standards or good practices for not important to safety systems. This additional assessment work may be desired to satisfy other requirements such as economic requirements, the desire to reduce exposure to increased scrutiny by the licensing authority due to bad performance of a system, etc.

The assessment plan must cover all aspects of the life-cycle of the system since it is important that each life-cycle activity is done correctly. Experience has revealed that the sooner an error found, the easier and cheaper it is to be corrected. For example, an error in the requirements specification will cause errors in the design, which will cause errors in the development, which will cause errors in the actual system. If this error is not discovered until assessment on the system as part of, for example, the site acceptance tests (SATs), the design will need to be fixed and a certain level of design, development, and implementation work on the system, as well as additional assessment activities, will have to be done. This can be extremely costly. The more effective approach would have been to discover the error during or, at worst, at the end of the requirements specification cycle. Then the error could have been fixed before starting the next phase of the life-cycle. To be able to do this requires assessment activities as often as reasonable. Therefore, the management plan for the assessment work, to be cost effective , must do all it can to maximize the likelihood of discovering errors as quickly as possible after they are made.

Interactions with the licensing authority is critical for cost effective assessment of safety and safety related systems. Management needs to set up interactions with the licensing authority as early in the modernization project as feasible to discuss the proposed new system. It is important at this time to make sure that there is an agreement on the relevant regulations and acceptance criteria that apply to the proposed system. This is also the time to identify any concerns or issues the licensing authority has about the proposed work. Management also needs to make sure that there are regular meetings with the licensing authority throughout the project to keep the licensing authority aware of what is being done on the project and to continue to identify any issues or concerns the licensing authority may bring up. It is also a good idea to provide the licensing authority with draft documents supporting the licensing case during the project. This way they will be familiar with them and can give feedback on them. It is necessary to develop all of the documentation for the licensing case in a comprehensive and clear manner to minimize the need for additional requests for information from the licensing authority. The importance thing is to do everything possible to minimize the likelihood of any surprises when the request for approval is made to the licensing authority.

2.2. CONSIDERATIONS TO SUPPORT COST EFFECTIVE ASSESSMENT

An important consideration for cost effective assessment, and cost effective implementation, of a system is to look for ways to minimize the assessment activities. One way is to look for a system that already exists and that has already been qualified to the regulatory requirements. If this system can be used as is, that is great and minimal assessment is required. However, most likely the system will need to be modified. Still the management plan should take as much advantage of the existing system and its qualification as possible. This would reduce the development work and the assessment work, making both more cost effective . Even if the system can be used as is, but it was built under the licensing requirements of another country, there will still need to be additional assessment activities to meet the licensing requirements of the country that it will be implemented in. Again, it is highly recommended to take advantage of the development and qualification work done to previously to minimize the work required to satisfy the licensing authority. Another benefit of using or starting with an existing system that has been qualified is that the risk that new system can be qualified to meet licensing requirements is reduced.

In the situation that a system has to be developed completely, or an application has to be developed completely on an existing platform, there are a couple of approaches that can help the development and assessment activities to make them more cost effective. The first is to look for modules that already exist that can be used to make up parts of the new system. Here the management plan should take advantage of the qualification work and operating experience that already exists to minimize the new assessment work. The second is to build the new system in a modular form so that the modules can be tested separately and confidence with them can be gained before they are integrated into the whole system where the testing is more complex. This does not remove the need for the assessment of the integrated system but reduces the effort required. Building the system up by modules and assessing the modules has the side benefit of building confidence in the system as it is being built and is an on-going process that continuously increases the confidence throughout the project.

2.3. MANAGEMENT PLAN

The management plan describes actions performed within the assessment process to ensure that the results are able to achieve all specifications and requirements of the contract between contractor and utility (or internal assessment to achieve requirements of internal assessment project). Two important portions of the management plan are the quality assurance plan and the configuration control plan. The main management plan objectives are:

- To meet project objectives and specifications and requirements of the project;
- To assure that the project organization and role of individuals is doable and effective to achieve all project objectives within conditions given in the contract;
- To assure that the scope of all activities within the project (including reviewing, testing, training etc.) is complete and unambiguous to achieve desired project results;
- To ensure technical and professional qualification of the assessment team;
- To define activities to assure project meets agreed upon criteria for completion;
- To define project management activities and management plan control from the start to the end of the project; and
- To ensure quality control and configuration control within the project.

To achieve these objectives, the management plant should include especially:

- Project scope and objectives, including the scope which describes the management process and interface with the customer;
- Project organization including contractor and customer personnel involved within the project and their responsibilities which are documented and clearly understood;
- Project realization process giving the description of the analysis and assessment process in detail, reviews, support to the customer, anomalies categorization, findings reports, findings dispositions, etc.;
- Training of people involved within the project to ensure that the staff has the appropriate level of training and experience for the task they need to do and suitable evidence of the qualifications of the staff should be available for customer inspection;

- Procedures to achieve the acceptance criteria (team selection and qualification acceptance, contractor review of completed work, customer audits, review and audits of completed analysis);
- Project schedule and resources;
- Resource planning (to prevent non-productive effect, to assure cost effective assessment);
- Progress reporting;
- Technical deliverables to the customer;
- Metrics (related to project and staff productivity, evaluation, and monitoring);
- Risk management (resulting from project risk evaluation performed by contractor);
- Progress meeting of the contractor with customer;
- Change control (procedures for rework and extra work analysis);
- Technical interface between customer and contractor;
- Confidentiality procedures (for control of confidential information);
- Quality control as defined in the quality assurance plan;
- Configuration control as defined in the configuration control plan; and
- Contract close out procedure.

2.4. QUALITY ASSURANCE PLAN

The quality assurance plan (QAP) describes the activities the project team needs to do to ensure the quality of the project. The main QAP objectives are:

- To meet customers technical and quality requirements in the contract specification;
- To assure that quality assurance, audits, reviews and inspections will be performed according to the contractor's quality program and procedures and to applicable codes or standards;
- To assure that all significant deviations from the specification, procedures etc. will be discussed with the customer before applying; and
- To ensure that the QAP will be correctly applied.

With respect to these objectives the QAP should include:

- Contractor's company quality program;
- Contractor organization and personnel responsibility (in case of subcontractor its organization and personnel responsibilities);
- Contract review;
- Design control (in case of software development contract);
- Document preparation and data control;
- Purchasing (in case that the contractor is expected to buy any products);

- Control of products supplied to customer;
- Product identification and traceability;
- Assessment process control;
- Inspection and testing process;
- Control, inspection of measuring and testing devices;
- Control of non-conforming products (such as software files from customer, tools for assessment, incorrect outputs from contractor, documentation, etc.)
- Corrective and preventive actions;
- Handling, storage, packaging, preservation and delivery processes;
- Control of quality records (contract documentation, records on executed audits, record on documentation development, non-conforming documentation, documentation on corrective actions, personnel cards, etc.);
- Internal quality assessment and audits (planning, performance, and assessment);
- Training (assessment, team qualifications, in case of subcontractor verification and technical acceptance of its team for the contractor); and
- Service and maintenance procedures (if it is expected in the contract with customer).

2.5. CONFIGURATION CONTROL PLAN

The configuration control plan describes how are changes to any part of the system are documented and communicated with project team members correctly and in a timely manner. The configuration control plan includes:

- Consequences of proposed changes and definition of action to be performed;
- Procedures for documentation identification and documentation control; and
- Procedures for performing the assessment analysis work (computer configuration used to develop the system, progress records development, review reports, etc.)

3. REQUIREMENTS SPECIFICATIONS

3.1. INTRODUCTION

The accuracy of the specification of system requirements of a digital I&C system is of prime importance to the acceptance and success of the system. Experience has shown that a significant portion of the problems with software based I&C systems can be traced back to incorrect, incomplete, or ambiguous requirements for the system. When the software based I&C system is a replacement for an existing analog system, it is a mistake to just take the requirements specification for the old system and use it for the new digital I&C system, since the latter system has characteristics different from the former. The analog system functional requirements specification can be used as a starting point, but the digital characteristics must be taken into account when preparing the new requirements specification.

It is important to make sure that the requirements specification is accurate, consistent, complete, unambiguous, and robust. Software based systems bring with them unique opportunities and unique concerns compared to analog systems. Therefore, it is important to make sure that these differences are taken into account when the requirements specification is being developed

The system requirements specification needs to completely define the functions of the system and the qualification and test requirements, including acceptance criteria. The requirements specification must also address system issues including defining the external and internal interfaces, hardware-software interactions, and timing, for example.

The software requirement specification (SRS) is an important part of the overall system requirements and is used in design, implementation, project monitoring, verification and validation phases of the life-cycle. However, the members of the project team often do not have the same background and; therefore, do not tend to describe software requirements the same way. For effective communication between the utilities, licensing authorities, customer, developer, and qualifier, good requirements practices should be carefully considered when writing an SRS. Good requirements should be a) correct, b) unambiguous, c) complete, d) consistent, e) ranked for importance and/or stability, f) verifiable, g) modifiable, h) traceable, and g) understandable.

The requirements can be classified by the system structures which make up the software based I&C system, as follows:

- System requirements;
- Hardware requirements;
- Software requirements; and
- Human factors (HF) and human-machine interface (HMI) requirements

By subject area, requirements can be classified as follows:

- Safety requirements;
  - Application of single failure criteria;
  - Application of redundancy criteria;
  - Application of independence criteria;
  - Application of diversity criteria;
- Reliability and availability requirements;
- Level of automation requirements;
- Security requirements including protection against unauthorized access;
- Timing and accuracy requirements;
- Environmental requirements (EMI, RFI, Seismic, power supply interface with other system);
- Documentation requirements;

- Maintainability requirements;
- Requirements for system operation;
- Who's requirements to follow such as utility requirements, e.g., EPRI Utility Requirement's Document [15];
- Regulatory requirements (for the country in which the system will be implemented); and
- Requirements for standards and guidelines.

System requirements also can be classified as follow:

- Requirements which can not changed (usually licensing requirements; however, there may be different approaches to satisfying the requirements with some more cost effective than others); and
- Requirements which can be changed and cost effective assessment may be a factor for choosing which of these requirements should be changed.

To the utilities, independent assessors and licensing authorities, good requirements should provide several specific benefits, such as the following:

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do;
- Reduce the development effort;
- Provide a basis for estimating costs and schedules; and
- Provide a baseline for validation and verification.
- Serve as a basis for enhancement.

The requirement specification may be written by one or more representatives of the supplier, one or more representatives of the customer, or by both. The SRS writer should avoid placing either design or project requirements in the SRS. A SRS should establish the requirements for a software based system. A Software Requirements Safety Analysis (SRSA) should confirm that the requirements do not pose a system hazard.

The SRS can be viewed as a bridge, or connection, between the overall system design and the software. The design of the system can be expected to impose certain requirements on the software. The SRS can be viewed as a translation of these requirements into a language understandable by software engineers, and defines the specific software requirements necessary for achieving the desired behavior of the software based system.

There are two safety issues involved in the SRS. First, it is necessary to ensure that all system hazards that the software is expected to handle are indeed covered by the SRS. Second, it is necessary to ensure that the SRS does not add additional hazards to the system. The SRSA is one means of analyzing these safety concerns.

To avoid unnecessary cost and the engineering effort in assessment, the determination of the assessment activities should take into account the safety classification of system. Once the

safety classification of the system is determined, then the minimum set of assessment activities can be defined.

Requirements considered in this report can be classified from several kinds of viewpoints. Good classification of requirements can reduce the implementation and assessment cost of the software based I&C systems. That is, the requirements must be well classified by the safety class of the target system, by system structures, by subject areas, and by whom they are required. First of all the requirements can be classified by criticality level as follows (using the IAEA classifications as shown in Figure [3-1], 16):

- Requirements for software based safety I&C systems
  - Requirements for new software based I&C systems
  - Requirements for upgrades using software based I&C systems
- Requirements for software based safety related I&C systems
  - Requirements for new software based I&C systems
  - Requirements for upgrades using software based I&C systems
- Requirements for software based not important to safety I&C systems
  - Requirements for new software based I&C systems
  - Requirements for upgrades using software based I&C systems

An important factor that supports being cost effective when implementing and assessing the requirements is the complete separation of these three sets of requirements. Each has a different level of minimum requirements for the system and the assessment of the system. There may be other reasons, as mentioned in Section 3, that would necessitate more than the minimum level of requirements.

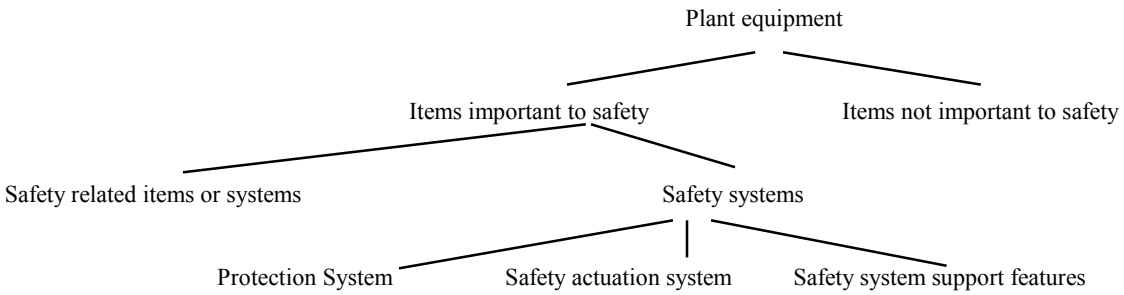
It is very important to note that the system requirements must be high quality requirements for every system. The same is true for the software requirement that are defined by the system requirements. The purpose of this using this software classification in relationship to the assessment requirements is:

- to help to evaluate the system software in sufficient detail and ensure an adequate level of confidence commensurate with the system's (or software's) importance to both safety and availability; and
- to provide a basis on which a cost effective V&V process can be established that uses the starting point as the minimum level of activities based on the software classification.

Appendix A gives another example of software classification which is based safety integrity levels. It then identifies cost effective assessment by defining the appropriate V&V for each level.

3.2. REQUIREMENTS SPECIFICATION ACCEPTANCE CRITERIA

The acceptance criteria for requirements specifications can be divided into two sets: functional characteristics and process characteristics, as shown in the following Table 3-1. Not all characteristics occur for every design output in the software life-cycle. The SRS that exhibits the functional characteristics and the software development process characteristics listed in Table 4-1 should be produced.



Examples:

Reactor control systems	Initiation I&C for:	I&C for:	Actuation I&C for
Plant control systems	Reactor trip	Emergency power supply	Reactor trip
Control room I&C	Emergency core cooling		Emergency core cooling
Fire detection and extinguishing I&C	Decay heat removal		Decay heat removal
Radiation monitoring	Confinement isolation		Confinement isolation
Communication equipment	Containment spray		Containment spray
Fuel handling and storage I&C	Containment heat removal		Containment heat removal

I&C associated with operation of the safety systems

I&C for monitoring the state of the safety systems

Access control systems

Figure 3-1. Examples of I&C systems important to safety.

Table 3-1. Acceptance Criteria Characteristics [17]

Functional characteristics	Development process characteristics
Accuracy	Completeness
Functionality	Consistency
Reliability	Correctness
Robustness	Clarity
Safety	Traceability
Security	Unambiguity
Timing	Verifiability

For example, Reg. Guide 1.172 [18], "Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants," which endorses IEEE Std 830 [19], "IEEE Recommended Practice for Software Requirements Specifications," describes an acceptable approach for describing software requirements.

Appendix D gives a discussion and checklist for each of the functional characteristics and development process characteristics given in the table above.

3.3. REQUIREMENTS MODELING

Models can be used to acquire understanding of the SRS, and to demonstrate that the SRS has been correctly constructed to satisfy the system. Such models abstract particular aspects of the requirements for more intense examination. Different models use different abstractions, and thus yield somewhat different sorts of information about the SRS. The following types of models are frequently used. Details on them can be found in books on software engineering.

- Structured Analysis and Structured Design Methodology
- Object Oriented Analysis and Object Oriented Design
- Component Based Modeling – Unified Modeling Language (UML)

The UML can be used to create models combining several of these different techniques. A valuable use of UML modeling is the activity of creating the model. If this can be done, and the result is sensible, then the portion of the requirements included in the model is likely to be correct. In any case, the action of creating the model is very helpful in understanding the requirements.

The requirements review process can benefit from the modeling of the SRS. Experience has shown that a UML model is appropriate. Creating such a model without adequate tool support is difficult, so if a UML model is desired, a tool should be acquired.

3.3.1. Prototyping

The prototyping life-cycle shown in Figure 3-2 is an approach that uses an operational system to help determine requirements. In a prototyping model, some system capability is built with minimum formality and control to be run for or by the user to obtain his input, so that requirements can be determined accurately. Several successive prototypes will usually be built. The amount of requirements analysis that precedes prototyping depends on the specifics of the problem. It is normally recommended that the prototype should be used only to help generate a valid set of requirements; after the requirements are available, they should be documented, and development should proceed as in the previously described management model. Prototyping is being used frequently during the requirements portion of a project. Many commercially available tools and methods are available to help support prototyping.

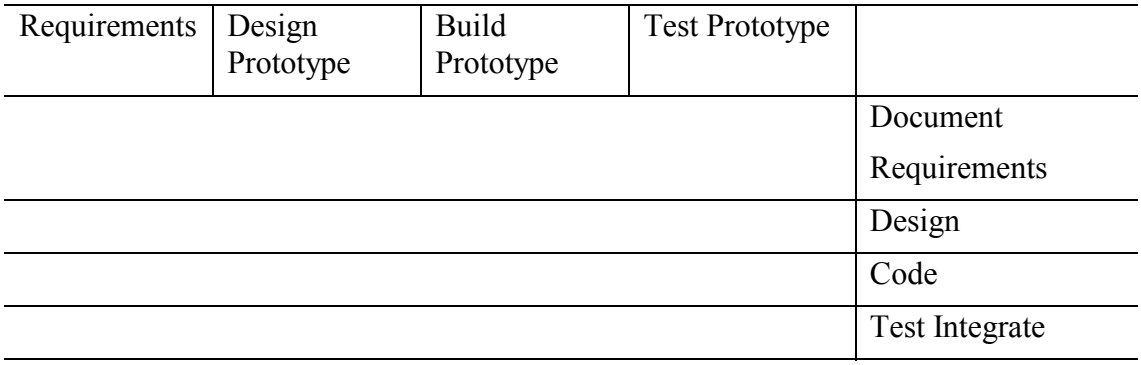


Figure 3-2. The prototyping life-cycle model.

For a reactor safety system, prototyping is likely to be most useful in modeling data communications, particularly if a potential problem is anticipated. This might be the case where many separate computers share a single network. A prototype of the network can be constructed in order to ensure that there will be sufficient network capacity even under the worst case conditions.

3.3.2. Formal methods

Mathematical analysis of requirements can be quite helpful in determining if requirements have been stated with precision and in discovering subtle inconsistencies. There are three primary methods in use. They are the Vienna Definition Method (VDM) and Parnas (rational design process, integrated approach). If formal methods are used, experience indicates that the third is most desirable. This has been used on reactor safety systems at Darlington Nuclear Power Plant with some success.

The Darlington Nuclear Power Plant's effort was extremely expensive, but this was at least partially a result of imposition of the formal method after the code was finished. This has been found to be generally true of formal methods. They need to be used initially and then followed throughout the development; otherwise they become too expensive and time consuming to be worthwhile. Perhaps because of the after the fact imposition, Parnas added many features to his method that appear unnecessary. It is believed that a simplified version of this could prove helpful in verifying the functional characteristics of the requirements, if a formal verification is used.

The functions required for reactor safety systems are often described in terms of Boolean equations before the software requirements are written. If this should occur, these equations are themselves formal mathematical descriptions, and nothing further should be necessary. The Boolean equations themselves, of course, are subject for formal verification.

A primary value of translating a requirements specification into any formal language is whether or not it can be done. If it can, then the requirements are probably correct. If not, or if there is difficulty, then the requirements need some additional work.

Formal methods are not being recommended to be used for requirements analysis since there is not convincing evidence that the resulting increase in safety is worth the cost when compared to other methods of analysis.

3.4. SOFTWARE REQUIREMENT SPECIFICATION REVIEW TECHNIQUES

There is no single assessment method that can, by itself, provide an adequate level of confidence in the SRS or the SRSA. Therefore, it is recommended that two or more methods be used, where the methods are selected to compensate for each other's weaknesses.

One method that should always be used is that of requirements reviews and requirements safety reviews. Other possible methods are:

- Reviews;
- Metrics collection and analysis;
- Requirements testing (e.g., statechart, dynamic data flow, etc.); and
- Hazards and operability (HAZOP) analysis.

The results of all these analyses can be combined into a single assessment of the quality and safety of the SRS. In general, this will be a qualitative result; in a few cases, a quantitative result may be possible.

3.4.1. Reviews

The SRS review includes questions about safety. The first set of questions can be used to examine the SRS for desirable safety properties. The second set of questions, on the SRSA, can be used to examine a safety analysis that has been carried out on the SRS. Thus, two levels of analysis are possible. In a specific case, some choice should be made as to which sets of questions are to be used, based on the actual circumstances.

3.4.2. Review data collection and analysis

It is suggested that a small amount of metric information be collected during the requirements review activity. This will serve primarily as a baseline for further work, and can be useful in assessing the increasing capabilities of the reviewers and in comparing the efforts of different sets of verifiers. The following three primitive measures can be collected.

- Size of the requirements, which may be calculated using full function point analysis, number of pages, or any other method which can be used consistently;
- The number and criticality of defects found by each assessment activity; and
- The number of staff hours spent on each assessment activity.

From the above it is possible to calculate several derived measures, such as defect density and review efficiency. Defect density is defined as the number of defects (of each level of criticality) per unit of software size. For example, if function points are used as a size measure, then a derived measure could be number of safety-critical defects per function point. There is some ambiguity in interpretation of defect density. A low number might mean a good requirements specification, or a poor review team. On the other hand, a high number always means a poor specification. Consequently, some care is required in interpreting this number.

Review efficiency is defined to be the number of staff hours spent in reviews per defect found. There is also some ambiguity here, in that a high number might mean few defects so that a long time is required to find each, a poorly written specification so that it takes a long time to understand the specification well enough to identify defects or a poor or inefficient review team.

It is important to focus on the insights gained from collecting metrics, not on the numbers. Early collection of metric information, particularly the derived measures, are useful mostly to establish a baseline from which to determine future process improvement or the lack

thereof. Effort is required in order to penetrate behind the numbers in order to discover what they mean. Without a commitment to making this effort, metric collection is pointless and should be omitted.

3.4.3. Requirements validation

At the end of validation, the implementation of the software requirements will be validated by some type of testing. This can be very difficult if the requirements are not written so as to be testable. Therefore, as part of the requirements verification activity, one or more specific test cases should be written for each software requirement. This test should be able to objectively determine whether the requirement has been met or not. That is, it should be possible to write a test driver that will execute the software for each requirement, and compare the result with a pre-determined answer.

The generation of validation test requirements is independent of whether the software is developed in house by the vendor, whether pre-developed software is used, or whether COTS software is used. The validation process is much the same in concept, though the test driver will be different in each case. It doesn't really matter if the tests generated through this activity are the actual tests used later on, though, of course, it is hoped that most of them are usable. The important thing is to determine whether such test cases can, in fact, be developed for the requirements specification.

At this point, the test designer should not worry about practicalities such as the cost of carrying out the tests, whether some tests are redundant, or about detailed test case definitions. For example, a test to ensure that a specified degree of reliability has been met may be very expensive to run, but the definition of such a test itself can reveal faults in the specification. Tests can be generated in concept for functional requirements (accuracy, functionality, reliability, robustness, safety, security, and timing).

3.4.4. Hazards and operability (HAZOP) analysis

Hazard analysis is used extensively in reactor designs, and can be extended to the software design elements with some care. Little extensive experience with software hazard analysis has been reported in the literature. Software requirements hazard analysis investigates the impact of the SRS on system hazards. One approach is to ask HAZOP-type questions about the requirements document, as the first step in the software hazard analysis. For example, the following questions should be asked about sensor input:

- What should the software do if the sensor is stuck at all zeros?
- What should the software do if the sensor is stuck at all ones?
- What should the software do if the sensor is stuck somewhere else?
- What should the software do if the sensor is below the minimum allowed range?
- What should the software do if the sensor is above the maximum allowed range?
- What should the software do if the sensor is within range, but wrong?
- What should the software do if the physical units are wrong?
- What should the software do if the sensor data has a wrong data type or data size?

Similar types of questions should be asked about other functional properties of the software requirements.

3.5. TOOL SUPPORT

The importance of managing the requirements of an I&C development project effectively cannot be overstated. Requirements management is a relatively new term. It used to be called “requirement engineering”

Requirements management is the process of establishing a common understanding of application requirements and managing changes to those requirements. These changes must be managed throughout the development cycle to ensure that the application will meet end-user needs. To accurately identify, track, manage and communicate the changing requirements of a project, organizations need an effective requirements management process. A conventional requirements management process includes capturing information either in documents or in a database. Organizations that practice careful requirements elicitation typically end up with a set of documents that capture the needs of varied end users. However, organizations are often limited in their ability to relate information in one document with information in other documents. In addition, these documents are often difficult to edit and distribute to the entire project team when changes are made. Therefore, the end results of using documents to capture requirement information are often delayed, extended, over-budget, or canceled projects

The challenge facing many projects is how to combine the convenience of capturing requirements in a document with the power of data handling available from a database. To meet this challenge, projects need an easy to use central repository. Such a repository would allow project teams to:

- Define and organize requirements more completely through attributes;
- Track the relationships between requirements;
- Share requirements information with the entire team; and
- Communicate changes to the requirements easily.

Most commercially available requirements management tools do essentially this. They capture information and put it into documents. Most people today manage requirements using word processors, or maybe spreadsheets, or a desktop database, or any combination of the three. But, there are a lot of shortcomings in trying to do it this way. The major shortcomings are as follows:

- Incomplete requirements and specifications;
- Changing requirements and specifications; and
- Lack of user input.

If commercially available tools and methods are selected and used correctly, they can provide considerable cost benefits. Proper use of commercially available tools and methods is an important part of requirement specification. Selection of tools and methods requires investigation of the applicability and compatibility with each country’s practices and needs.

3.6. BENEFITS OF REQUIREMENTS MANAGEMENT

Requirements management brings with it several benefits that will support cost effective development of requirements. These benefits include:

- Promotes complete, clear, and correct definition of application requirements;
- Enables the project team to fully understand and meet the needs of customers the first time;
- Enables earlier identification of requirement deficiencies, ambiguities, and errors;
- Enables analysts to assign attributes to requirements for safety classification and prioritization;
- Allows the project team to easily identify requirements within the project scope;
- Allows customers to review requirements and ensure that their needs were understood and documented correctly;
- Allows collaborative discussion to refine and prioritize requirements more completely;
- Uses traceability to show how changing requirements will affect other requirements;
- Allows project managers to easily identify extraneous or missing requirements through the traceability matrix;
- Allows project scope to be managed more effectively, reducing feature creep and rework;
- Provides the means to more accurately estimate required time and work estimates;
- Improves communication between team members and customers so customer needs can be met the first time;
- Allows project managers to implement a requirements management process more efficiently and effectively, through customizable support for any process;
- Allows project teams to quickly identify requirements within the project scope;
- Helps customers and team members define and understand requirements more quickly to establish and maintain a more accurate project schedule; and
- Allows project teams to meet the customers needs the first time, reducing re-work.

4. SYSTEMS EXPLICITLY DESIGNED UNDER NUCLEAR POWER PLANT LICENSING REQUIREMENTS

4.1. INTRODUCTION

When an I&C system is being planned for a modernization project or for a new plant, the solution will most likely be using today's state of the art technology. That is, a digital system using software to realize all data processing and logic calculations to get the desired functionality. The main decision of the project management; therefore, will be focused on the question whether the new system should be based on a product for which the software has been designed explicitly for nuclear power plant (NPP) application, i.e., developed for the application under the licensing requirements of that country; or should it be based on a system

available on the market. This latter falls into two categories. The first is a system that has been used successfully in other industries and is broadly available at a lower cost. However, additional qualification and documentation has to be done to meet the licensing requirements required for nuclear power plant systems. This additional work can potentially lead to substantial costs.

The second is a system that has been developed explicitly for nuclear power plant applications under the licensing requirements of a different country (where these licensing requirements are different than those of the home country). There will still need to be additional qualification and documentation to fill the gaps between the licensing requirements in the home country and the licensing requirements of the country in which the system was developed. However, the additional work should be considerably less, and less costly, than for a system not developed under any program to satisfy licensing requirements. The systems in these two categories will be considered in this report to be commercially available (or COTS) products and will be discussed in Sections 5 and 6.

This Section will discuss only software based I&C systems that are explicitly designed under the licensing requirements of the country for which the application will be used. Herein, in principle, two choices are possible and the influences on the resulting efforts for the assessment have to be estimated as the basis for the overall decision. The two choices are:

- Software explicitly designed for the specific application (functionality), i.e., in case of a particular safety system or even the entire reactor protection system, a homogenous software package designed for this functionality, or
- Software explicitly designed as re-usable software components, which are configurable for the plant-specific requirements and any safety system architecture.

Both options can have their justification. Therefore, both have to be carefully investigated with respect to consequences to the total cost, i.e., the hardware/software and assessment costs necessary to realize a system coping with the licensing requirements. In the following sections both options are briefly discussed.

4.2. SYSTEMS USING SOFTWARE EXPLICITLY DESIGNED FOR SPECIFIC APPLICATIONS

4.2.1. Main system characteristics

This type of software is newly designed and validated for a particular application. The software is homogeneously written and explicitly designed to fulfill all relevant tasks. Design and validation of this type of new software for a specific functionality are performed in agreement with the requirements found as consensus or in agreement with the requirements of already existing software-related standards, see [9, 13, 20,–25]. In addition, the software and overall system must satisfy the requirements of the licensing authority of the country in which it is to be used.

An important approach for systems that are explicitly designed for nuclear power plant applications is the reuse, as much as possible, of software modules that have been used in

other nuclear power plant systems where these modules have already been tested and qualified. Of course, it is essential to ensure that the modules will be used in a consistent manner as they had been tested and qualified. These modules could be used for the operating system of the platform, functions, tools, interface displays, etc. These re-use modules can also be used for the applications that are to be implemented on generically pre-qualified platforms which will be discussed in Section 6.

Writing modular software is a very effective approach for new systems. This approach makes the software less complicated so that it is easier to design and develop. It allows each module to be extensively tested individually before it is integrated into the overall system. This will build up confidence in the system as it is being developed and will allow more cost effective assessments of the quality of the system. It also puts the software in a form that allows it to be easily reused for other applications. Therefore, modular programming should be used for explicitly designed software and for applications that are built on COTS products, which will be discussed in Section 5.

4.2.2. Motivations to apply explicitly designed software

Considerable effort has been spent in the past to develop a criteria-based system to define categories of graded safety relevance for I&C systems important to safety in NPPs [22]. In case of lower graded safety requirements, e.g. safety categories B or C of IEC 61226 the efforts for assessment of the whole applied software (i.e. application software and operating system software) can be reduced to the needs of the relevant safety category [22, 23]. Therefore, if homogeneous explicitly designed software for a specific safety functionality can be assigned to a category with reduced requirements and since it does not have to consider the interface relationships of a modularized software structure, the effort for assessment might be reduced down to a lower level of effort.

It is possible to use parts of COTS software in the development of explicitly designed software. An approach on how to do this and which features or additional measures have to be considered when applying it in different safety categories has been investigated recently. A scheme of a guideline for the qualification of pre-developed software for safety-critical I&C application in NPPs has been presented at the OECD/NEA workshop meeting in Hluboka, Sept. 2001 [26] and will be published in the proceedings issued by NEA/CNSI.

For the highest safety category, explicitly designed software can be developed. In this case, experience has shown that for a completely new safety system, which is applicable to many different functions in the nuclear power plant, the assessment efforts are extremely extensive and costly. Since other nuclear power plants cannot use the system in the identical manner, a similar extensive level of assessment efforts are needed for use in a different nuclear power plant.

Concerning new explicitly designed software, it can be stated that only if there are many cases for re-use of the development and assessment procedures can such a kind of new explicitly designed system be recommended.

4.3. QUALIFICATION OF HARDWARE- AND SOFTWARE COMPONENTS

Listed below are the types of qualification required for a product to be used for nuclear power plant applications. The amount of qualification work in each is dependent on the safety and other requirements for the system.

- Type testing (assessment) of hardware components (e.g.,: environmental conditions, EMI, seismic, long term tests, etc.);
- Type testing (assessment) of software components (constructive and analytical methods of V&V procedures);
- Integration test of hardware/software in a representative system: assessment of product-specific features, list of open tasks to be assessed during the plant-specific assessment; and
- Certification of module assessments and integrated system assessment.

In addition, application-specific qualification is also needed. Listed below are the types of application-specific qualification needed. Again, the amount of qualification work in each is dependent on the safety and other requirements for the system.

- Verification of specification (always necessary, graphical representation useful);
- Compliance assessment of documents;
- Application code verification, e.g., by using verification methods or tools;
- Test field functionality simulation, wide-range stressing including fault conditions, factory acceptance tests;
- On-site acceptance tests;
- Commissioning tests; and
- Up-date of the central hardware/software configuration management system with the plant-specific data.

These qualification activities apply to all nuclear power plant systems, not just those that explicitly designed.

5. COMMERCIALLY AVAILABLE (COTS) PRODUCTS

5.1. INTRODUCTION

A commercially available (frequently called commercial off the shelf or COTS) system or piece of equipment is one that has not been developed and documented under the approved approach by the licensing authorities for nuclear power plant applications. In this Section this system or equipment will be referred to as product for simplicity.

The advantages of using a COTS product are that the COTS supplier is more likely to continue to support the product and provide spare parts for a longer period of time because the

customer base is larger than is typical for nuclear specific system. On the other hand, the product has not been designed, developed, tested, etc. to nuclear power plant licensing requirements. Nor has its documentation been done to nuclear power plant licensing requirements.

A second category of COTS product is when that product has been built under an approved approach by the licensing authorities of a different country and this approach is different than the one approved by the licensing authorities of the country in which the product will be implemented. Additional work will need to be done to make the product satisfy the licensing requirements of the country that the product will be used in. However, the amount of additional work is substantially less than required for product in the first category, and will cost less. In addition, the risk that the product cannot be licensed is much less.

The plant applications developed with the COTS product are expected to be cheaper over the lifetime of the application than those developed as a custom designed (explicitly designed) nuclear plant application. The COTS product has a wider experience base. The disadvantage, especially for safety and safety related systems, is that it may be difficult to get enough information to make a convincing acceptance argument to the utility or to the licensing authority for the COTS-based application. It may also be difficult to demonstrate the relevance of the prior experience of the COTS product to the application of interest since it was not for a nuclear power plant application. A good method for gaining an acceptable level of confidence in the COTS product for nuclear power plant applications is needed, especially for safety and safety related systems.

Although much of the attention has been put on the assessment of safety and safety related systems, the assessment of COTS-based not important to safety systems is also critical. In many cases for not important to safety systems, high reliability is needed to make sure that the system does not cause challenges to the plant's safety systems. High reliability and high availability of the system is also critical for economic reasons.

The COTS-based systems must meet the same acceptance levels, based on the application of the system, as are required for other nuclear power plant systems. There has been some work done, as demonstrated in reference [26], that indicates that there would be a cost advantage for the assessment of COTS-based systems if the number of safety categories were increased. This has not been accepted by licensing authorities at this time.

5.2. ISSUES WITH COTS-BASED SYSTEMS

There are some aspects of COTS-based systems that should be kept in mind when they are being considered for nuclear power plant applications. Some of these are related to the qualification and acceptance of the system as will be discussed in detail in Section 5-3. Others are related to cautions for using this system for nuclear power plant applications.

In many situations, the COTS-based system may have to be treated as a "black box" without the possibility of obtaining information about its internal structure. It is very important to attempt to work with the supplier to be able to obtain information about the internal workings of the system. They may have documentation that they are willing to show the utility or qualifier, at least under a proprietary agreement. They may be willing to allow the

utility or qualifier to talk with the designers and developers to explore the internals of the system. Especially for systems where there is a need for high reliability or high safety requirements, it is important to attempt to learn as much as possible about the internals of the system.

Concerns about configuration control and error reporting are another area where it is important to put together some type of agreement with the supplier. Otherwise it will be difficult to justify using the system for nuclear power plant applications. This will also impact maintenance of the system since replacements or spare parts may be needed. It will be important to know that the replacements or spare parts are functionally equivalent and will behave in the same manner if they are not identical. This can easily be a concern since commercial suppliers frequently improve their products.

Another area to be aware of and handle appropriately is that the COTS-based system will most likely have extra features/capabilities in it that are not needed for the nuclear power plant application. In this case, it is important to understand all of the possible features/capabilities that the COTS-based system has in it. Then it is critical to make sure that the extra features/capabilities are turned off or removed so that they do not cause unexpected behavior in the application. Any extra failure modes that they can cause should also be evaluated.

5.3. QUALIFICATION AND ACCEPTANCE OF COTS-BASED SYSTEMS

As mentioned above, there are definite advantages for using proven and widely applied COTS products for nuclear power plant applications. This section discusses the qualification and acceptance of an existing COTS product so that it can be used for nuclear power plant applications including safety applications and safety related applications. Although there are well established minimum levels of acceptance for safety and safety related systems by the licensing authority, there are not minimum levels of acceptance established for not important to safety applications. However, due to the important economic impact and/or potential for safety system challenges by some not important to safety systems, the utility may choose to set its own acceptance level for the not important to safety system. This acceptance level has nothing to do with regulations, but rather is a utility defined requirement.

For COTS products, controlled and monitored development processes and documentation are not performed to the level required by the nuclear industry. It is also very difficult to reproduce the processes and documentation afterwards. Because of lack of exact formality and documentation meeting nuclear standards, assuring correctness and identifying failure modes are difficult. Suitable strategies and methods for acceptance of existing COTS products and applications based on them must be developed.

5.3.1. Overall steps for acceptance of applications based on COTS products

Before a software based I&C system built using a COTS product (hardware and/or software) can be used for nuclear power plant applications, the COTS-based system must be qualified and accepted. A high level description of the process for doing this is given here.

The first step is the selection of the COTS product to be used for applications in the plant. After this product is selected, a qualification program must be defined to qualify the product for nuclear power plant application. This program has all of the same aspects of a typical qualification program including seismic testing, environmental testing, EMI/RFI testing, software evaluation, system testing, etc. This qualification may be done by the supplier, the utility, or some other third party. After the qualification activities are done, an evaluation is done to determine if the performance of the product is acceptable for use in nuclear power plants.

If the qualification for nuclear applications is successful, the processes must be put into place for maintaining the qualification, for configuration control, and for error reporting in the case of safety and safety related systems. A process needs to be put into place for revising the qualification if changes are made to the product that has been qualified. An organization must have specific responsibility for the maintenance of the qualification, configuration control, and error reporting. For not important to safety systems, the utility can identify what, if anything, is required to handle the above topics.

When a nuclear power plant buys one of these nuclear qualified COTS products, there is a dedication process to make sure that the product purchased is the same as the product that was qualified. This will also include the evaluation of any changes from the product that was qualified and are supposed to be compatible with the existing qualification. After the dedication is successfully completed, the product is determined to be acceptable for use at the specific plant.

The next step is to qualify the plant application that is to be implemented on the qualified product. This qualification of the final system is essential to demonstrate that the system is acceptable for use at the power plant.

5.3.2. Basis for qualification of commercially available software based systems

Processes for evaluating commercially available hardware products for use in nuclear plant safety related applications have been well developed for many years. The plant safety systems were originally designed and built in strict accordance with nuclear standards. As the plants aged, it became necessary to replace worn or obsolete equipment, but in many cases the original suppliers of the nuclear qualified equipment no longer maintained nuclear quality assurance programs. However, equivalent commercially available equipments were available, and processes have been developed to evaluate and accept such equipment for nuclear safety applications. For example, the process used in the United States is based on verifying selected characteristics of the equipment and generating reasonable assurance that the equipment will perform its safety function. The approach uses special tests, vendor audits, etc., to confirm that the commercially available equipments have adequate quality, and this approach has been very successful for mechanical and electronic equipment. The approach is described in detail in EPRI NP-5652 and EPRI TR-102260 [27, 28]. However, this approach was not developed with software based systems in mind, and additional work was needed to clarify and extend the approach.

Use of digital technology in safety and safety related applications raises new design and licensing questions, regardless of whether the system was developed under a nuclear quality

assurance program or as a general use commercially available product. Issues include the use of software and the potential for common cause failure resulting from software errors, the effect of electromagnetic interference (EMI) on software based systems, and the use and control of equipment for configuring software based systems. The most notable of these concerns is the potential for software errors that could lead to common cause failures of redundant trains in the safety system.

It is important for the industry and the licensing authority to agree on a framework for addressing these digital issues in safety and safety related systems. An example of one such framework is the one used in the United States of America given in EPRI TR-102348 [29]. This framework emphasizes consideration of the effects of potential failures in determining system adequacy. It recommends using attributes of the software development process, such as evidence of a systematic verification and validation (V&V) program with supporting documentation, to help confirm software quality. This framework discusses the need to obtain an adequate level of assurance for a commercially available system with software that was not developed with all the desired nuclear pedigree, but it does not get into the details of how to achieve this.

Once a general licensing framework is established, it is advantageous to develop guidance based on that framework to help utility engineers with digital upgrades that involve commercially available products. An example of a guideline providing this guidance in the United States is given in EPRI TR-106439 [30]. This guidance provides a specific framework for the treatment of commercially available digital products. Again it is important for the industry and licensing authority to agree on this framework. For the example guideline above, a second report, EPRI TR-107339 [31], was developed. This report gives more detailed information on how to conduct the evaluations described in the guideline. This type of more detailed information is very helpful when the utility engineer is ready to gain acceptance of a COTS-based system.

5.3.3. Qualification of commercially available software based systems

The objective of the qualification of commercially available software based systems is to obtain a level of confidence in the system that is comparable to a system that was developed explicitly for nuclear applications using the approved nuclear development and qualification process. Since the commercially available system was not developed and qualified under an approved program, alternative methods must be used to obtain an adequate level of confidence.

As an example of a method that can be used, the one mentioned above in reference [30] is discussed in general here. The following provides an overview of the approach taken in this guideline for addressing digital issues within the established commercially available equipment qualification and acceptance process, as shown in Figure 5-1. This section introduces the basic problem faced when applying a commercially available software based system in a safety application; that is, obtaining reasonable assurance that the system will perform its intended safety function. (It should be noted that in the United States the category safety includes both safety and safety related categories of this report. Therefore, this example refers to only safety but is applicable to safety and safety-related.) This section also describes the processes that are currently used in the United States for design and licensing of software

based systems and for performing qualification and acceptance of commercially available systems. Although this is explicitly targeted at safety applications (including safety-related), the concept is the same for not important to safety applications. The difference is that the level of acceptance for the not important to safety application is defined by the utility rather than by regulatory requirements. It should also be noted that the utility could always do more than the minimum regulatory requirements for safety and safety related systems if it desires to do so for other reasons such as economic ones.

The following questions are answered for this example:

- What are the differences between nuclear and commercially available software based systems that affect the level of assurance for their use in safety applications? What supplemental activities would be necessary with a commercially available software based system to obtain acceptance equivalence with a system developed under a 10 CFR 50, Appendix B, [32] quality assurance program?
- What standards and guidelines are used for design and licensing of a nuclear software based system, giving assurance that it is adequate for safety applications?
- What process and methods are used by the industry in procuring and qualifying and accepting commercially available products for safety applications?

5.3.3.1. The problem: obtaining an adequate level of assurance with commercially available software based products

As stated in the United States Code of Federal Regulation (CFR)10 CFR Part 21 [33], the goal of qualification and acceptance is to "provide reasonable assurance that a commercial grade item...will perform its intended safety function and, in this respect, is deemed equivalent to an item designed and manufactured under a 10 CFR Part 50, Appendix B, quality assurance program." (The term commercial grade refers to commercially available in this report.) Thus the judgment that an adequate level of assurance has been reached is based on achieving equivalency to nuclear systems (i.e., systems developed under an Appendix B program).

Figure 5-2 contrasts the assurance-building elements used for a nuclear system with those used to establish equivalent assurance for a commercially available software based system. The relative contributions of the various elements shown in the bars of Figure 5-2 are chosen somewhat arbitrarily to illustrate the basic concept. In practice, the contributions can vary widely depending on the particular application, vendor, and product being evaluated. Again, it should be remembered that this same approach and the contributions of the various elements applies to not important to safety applications. The difference is that the acceptance level is defined by the utility and not by regulatory requirements.

5.3.3.1.1. Assurance for nuclear systems

The bar on the left side of Figure 5-2 addresses a system that has been developed specifically for nuclear service. In this case, a significant part of the assurance comes from the use of an approved vendor who has a 10 CFR 50 Appendix B quality assurance program. However, this is not sufficient by itself to reach the needed level of assurance. The utility reviews the design of the system, and the vendor's development process and quality assurance

program. For software based systems, this includes evaluating the vendor's programs for software configuration control, verification and validation, and testing.

Standards such as IEEE 7-4.3.2-1993 [34], ASME NQA-1a Subpart 2.7 [35], and other software engineering standards and guides typically are consulted. The guidance in EPRI -TR-102348 [29] is used in addressing software based system issues and to support licensing, including the 10 CFR 50.59 evaluation [36]. Failure analysis techniques are used to identify the important failure modes for the system, and to examine the system design and the vendor's process for addressing potential failure modes and abnormal conditions or events (ACEs), as discussed in EPRI TR-102348 [29] and IEEE 7-4.3.2 [34]. If the system has been applied previously (it is not the first of its kind), its operating experience may be reviewed to determine whether it has been operating satisfactory. The utility may perform reviews of the vendor's design and quality assurance (QA) practices. Finally, when the system is received, the utility performs receipt inspections and acceptance tests, follows its own quality assurance program and quality control (QC) practices in configuring and installing the system, and performs further testing after installation to ensure that the system is operating satisfactorily and will perform its safety function. This entire process is documented and retained in plant records.

5.3.3.1.2. Equivalent assurance for commercially available systems

The right hand bar in Figure 5-2 illustrates how an equivalent level of assurance can be achieved with a commercially available system. The level of assurance must be at or above the level reached for the nuclear system. With a commercially available product, the process begins with the vendor's commercial practices for product design, development and quality assurance. Since the vendor does not have a 10 CFR 50 Appendix B quality assurance program, the process that was followed in development and verification of the product may not have included all of the elements of an Appendix B program, and documentation of the process may be lacking. The vendor's commercial practices may follow established commercial quality standards (e.g., ISO-9000 [37]), the elements of which are similar to a 10 CFR 50 Appendix B program. The utility may request that some additional activities be undertaken (e.g., additional testing or documentation). However, because nuclear power is a small part of most commercial vendors' markets, additional vendor efforts are likely to be quite limited.

For a commercially available system, documented operating history of the system can be an important factor in providing confidence in the system. This experience may have been gained through applications in industries other than the nuclear power industry, and may represent a much larger experience base than could be obtained with a system used only in nuclear applications. It is here that one can take advantage of the field experience and product shakeout that has occurred with widely used, mature commercially available systems. However, the experience must be shown to be relevant to the planned nuclear applications, in addition to being sufficient in terms of the number of units and length of time in service, and it must be successful experience.

Additional activities will be required by the dedicator to reach an adequate level of assurance for a commercially available system. An example would be additional testing needed to supplement the vendor's tests and build confidence in the system and its functionality, or to examine its response to specific conditions or abnormal events. Additional

reviews or analyses may be needed (e.g., review of the system design and analysis of its failure modes), depending on the extent of reviews and verifications performed by the vendor during the system development. Additional documentation may need to be produced, for example, in areas where it is evident that some process steps were performed by the vendor but not adequately documented. It is important to note that these supplemental activities by themselves do not add to or improve the quality of the commercially available system. Their purpose is to help confirm and document the commercially available system's quality.

The last element shown on the bar for reaching the needed level of assurance for a commercially available system is the utility's final acceptance and installation testing, and quality control during installation. Again, the entire process is documented and retained in plant records.

5.3.3.1.3. Supplemental effort and cost

For commercially available systems the vendor's activities will usually contribute a smaller portion of the assurance as compared to that for nuclear systems. As a result, the efforts by the utility or dedicator must provide a larger portion of the assurance. The amount of supplemental activity required and the associated cost can vary widely, depending on:

- The safety significance and economic risk associated with the specific application (this sets the overall level of assurance needed)
- How rigorous are the vendor's development and quality assurance practices
- The maturity of the commercially available system
- The complexity of the system, the more complex the system, the greater the effort to develop adequate confidence it will meet the requirements of the application, particularly with regard to potential failure modes.

The utility must, on a case-by-case basis, estimate how much will have to be done to supplement the vendor's process and documentation, and then determine the cost effective ness of pursuing qualification and acceptance (as opposed to buying from a supplier with an Appendix B program, if there is one). Also, there are cost trade-offs involved in choosing between commercially available systems. It may be more cost effective to select a somewhat higher priced system if the vendor of that system has a better process and will require less costly supplemental activities by the utility.

5.3.3.1.4. Demonstrating versus adding quality

It is important to make one point clear. The efforts performed by the utility or a third party do not add system quality. They seek to help confirm that the commercially available system already has adequate quality. If a product has a critical shortcoming, adequate qualification and acceptance may not be possible at any cost.

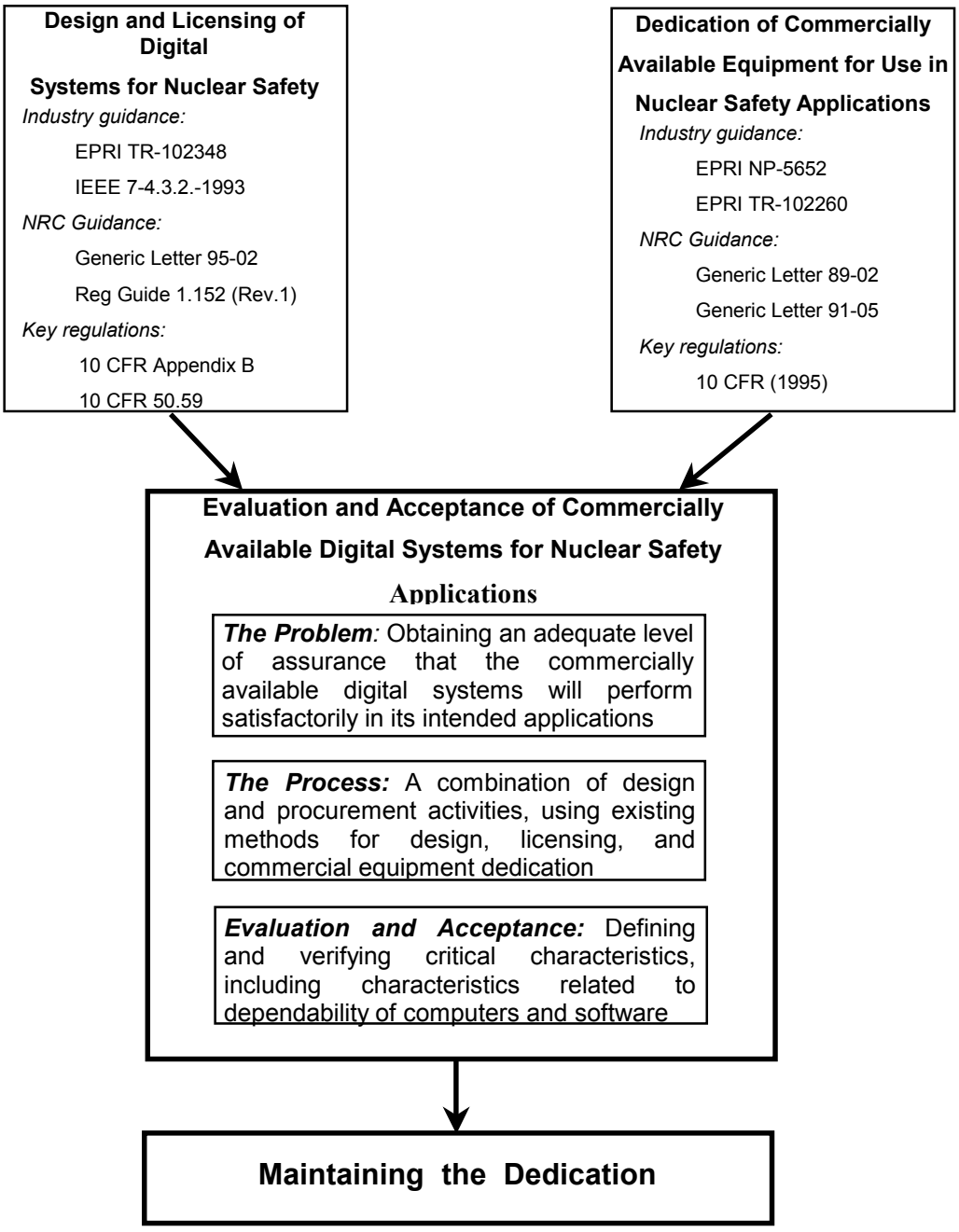


Figure 5-1. Overview of approach for addressing software based issues for COTS qualification and acceptance [30].

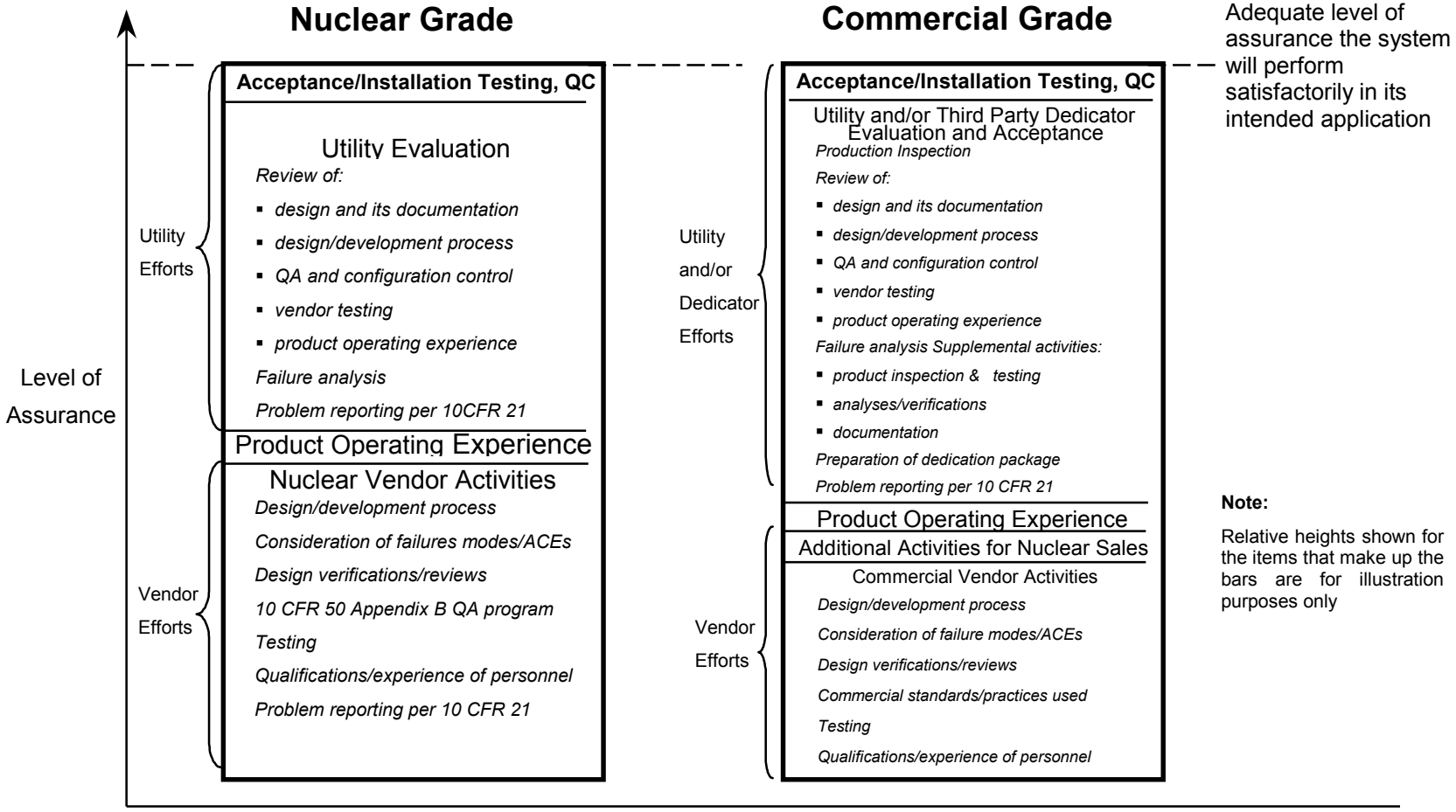


Figure 5-2. Equipment assurance for nuclear grade and commercial grade digital equipment [31].

5.3.3.2. Existing guidance on design and licensing of software based systems

An example of the existing guidance is the guidance that is currently available in the United States for design and licensing of software based systems for safety applications. Some of the key reports are listed below. These industry and United States Nuclear Regulatory (NRC) reports address the issues and concerns that have been raised with the use of software based equipment in safety applications.

Table 5-1. Comparisons between industry guidance and NRC guidance for licensing of software based systems

Industry Guidance	NRC Guidance
EPRI TR-102348, "Guideline on Licensing Digital Upgrades" [29]	Generic Letter 95-02, endorsing EPRI TR-102348 [38]
IEEE 7-4.3.2-1993, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations" [34]	Regulatory Guide 1.152 Rev. 1, endorsing IEEE 7-4.3.2-1993 [39]

The licensing guideline in EPRI TR-102348 [29] emphasizes the use of failure analysis and examination of system-level effects to assess the significance of failures in software based systems. This remains an important focus when evaluating commercially available software based systems.

IEEE 7-4.3.2 [34] and the guideline in EPRI TR-102348 [29] both address dedication of commercially available software based systems, emphasizing that the fundamental requirement is to obtain an adequate level of confidence in the commercially available system. As stated in the guideline in EPRI TR-102348 [29] and reinforced in US NRC Generic Letter 95-02 [38], this typically involves making an engineering judgment which needs to be documented.

Appendix D of IEEE 7-4.3.2 [34] provides additional guidance on commercially available product qualification and acceptance. It discusses the definition of functional and performance requirements, requirements related to behavior under abnormal conditions and events (ACEs), and verification of these for both hardware and software. It also discusses the need to evaluate the commercially available software development process and the operating experience of the commercially available system to obtain adequate confidence in the system being dedicated. NUREG/CR-6421 [40] discusses a standards-based approach for the evaluation of commercially available software based systems. However, none of these documents describes the relationship between the existing methods for commercially available product dedication and the issues that should be addressed for software based systems.

The guideline in EPRI TR-106439 [30] is intended to help fill that gap, showing how software based issues can be addressed within the established commercially available product qualification and acceptance process. In doing so, the guideline focuses primarily on digital-specific issues and criteria. Guidance for other types of systems is provided in other referenced reports.

5.3.3.3. Existing guidance on commercially available product dedication

An example of the existing guidance is the guidance on commercially available product qualification and acceptance that has been developed and used by utilities in the United States for a number of years. Key reports they have used are listed below. These guidelines have been applied successfully in qualification and acceptance of many different kinds of commercially available systems for nuclear safety applications.

Table 5-2. Comparisons between industry guidance and NRC guidance for commercially available product dedication

Industry Guidance	NRC Guidance
EPRI NP-5652, "Utilization of Commercial Grade Items in Nuclear Safety Related Applications" [27]	Generic Letter 89-02, conditional endorsing EPRI NP-5652 [41]
EPRI NP-6406, "Guidelines for the Technical Evaluation of Replacement Items for Nuclear Power Plants" [42]	Generic Letter 91-05, providing additional NRC guidance [43]
EPRI TR-102260, "Supplemental Guidance for the Application of EPRI Report NP-5652" [28]	

NP-5652 [27] is the primary source of guidance on commercially available product qualification and acceptance, and has formed the basis for many utilities' commercially available product qualification and acceptance programs, along with clarifying guidance provided by the NRC in Generic Letters 89-02 [41] and 91-05 [43]. EPRI NP-5652 [28] defines the basic process for commercially available product dedication: a technical evaluation, definition of "critical characteristics for acceptance," and use of any of four acceptance methods to verify the characteristics. The four methods are:

- Method 1 — Special tests and inspections;
- Method 2 — Commercial grade survey of supplier;
- Method 3 — Source verification; and
- Method 4 — Acceptable supplier/item performance record

EPRI NP-6406 [42], which gives Guidelines for the Technical Evaluation of Replacement Items for Nuclear Power Plants provides detailed guidance for technical evaluation of replacement products. This includes determining whether a replacement product is an equivalent or like-for-like replacement, or if it is sufficiently different that a design change is required. It also includes guidance on defining safety related functions and design requirements from which critical characteristics are identified.

EPRI TR-102260 [28] supplements EPRI NP-5652 [27], giving more clarification and guidance that builds on both EPRI NP-5652 [27] and EPRI NP-6406 [42].

Software based systems present new challenges in the qualification and acceptance of commercially available products. However, the same basic approach still applies. Key elements of the qualification and acceptance process are:

- An up-front technical evaluation to define the requirements for the system;
- Selecting from these a set of critical characteristics for acceptance; and
- Applying the methods described in EPRI NP-5652 [27] (as endorsed by US NRC Generic Letter 89-02 [41] and supplemented by US NRC Generic Letter 91-05 [43]) to verify the critical characteristics.

With software based systems, there are new critical characteristics and additional verification activities that need to be performed. For the qualification and acceptance to be successful, these activities must achieve the required level of assurance for the commercially available system, as shown in Figure 5-2. Typically, this requires the use of more than one of the methods described in EPRI NP-5652 [27] since no one method (e.g., testing per Method 1, or review of performance history per Method 4) will suffice by itself. For many software based systems, Methods 1, 2 and 4 will be needed.

6. GENERIC PRE-QUALIFICATION AND PRE-ACCEPTANCE TO SUPPORT COST EFFECTIVE ASSESSMENT

6.1. INTRODUCTION

Generic pre-qualification is a strategy that can be used very successfully to reduce the cost of the assessment of applications when they are implemented in a nuclear power plant. Generic pre-qualification is defined in this report as the qualification that is done for as wide a range of applications as possible before it is used for a specific application. For example in the traditional approach, when a specific application is implemented in the plant, it is necessary to qualify the entire application system at the time of implementing it. That is, if the system consists of a software based platform and the application software on the platform, it is necessary to qualify the entire system of both the platform and the application at the time the application is implemented. If the software based platform is generically qualified for a wide range of applications before it is used for a specific application, this is generic pre-qualification of the platform. Then, when an application is developed on this pre-qualified platform, it is only necessary to qualify the application on the platform without having to qualify the platform at that time, since it has already been qualified. This will reduce the assessment effort/cost and risk for the application.

Generic pre-qualification for a wide range of applications is usually more costly to do than is the qualification for a specific application. However, if the pre-qualified component is to be used for multiple applications or for multiple plants, the assessment of the applications will be more cost effective and will overshadow the additional costs of initial generic pre-qualification. There is another strong point for generic pre-qualification and obtaining the licensing authority's acceptance of the platform. It also reduces the risk for the acceptance of the application since part of the application by the licensing authority (e.g., the platform) has already been accepted. This is very significant since the platform on which the application is built is almost always more complex and difficult to qualify than is the application software on it.

Although generic pre-qualification is commonly thought of only for safety and safety related systems, there are advantages for not important to safety systems when additional qualification is required for the high reliability needed for economic reasons or to reduce the likelihood of challenging safety systems. The same arguments hold for the advantages of generic pre-qualification efforts on not important to safety systems so that they do not have to be repeated for different applications or plants. The only difference for not important to safety systems is that the qualification efforts are not required by regulations, but instead are required by the users and for the user defined acceptance levels.

Another important way to support cost effective assessment, and to reduce regulatory risk and cost, is to obtain pre-acceptance by the licensing authority of methods and processes to be used for assessment. Frequently the regulations tell what must be done, but they do not tell what is an acceptable way to do it. Or if the licensing authority gives guidance on an acceptable way to meet the regulations, it may not be the most cost effective approach. The advantage of having an agreement by the licensing authority on cost effective methods and processes to be used is that it makes clear what is to be done. When this agreement exists, there is no question that assessment work will need to be done over in a different way because the approach used turns out not to be acceptable to the licensing authority when it is submitted to him. In addition, it will help prevent unnecessary assessment activities that might be done because an agreement with the licensing authority on what is necessary to be done does not exist.

6.1.1. Types of components that can be pre-qualified

The following paragraphs describe some of the types of components that could be pre-qualified to support cost effective and reduced risk assessment. The concept of pre-acceptance by the licensing authority of methods and processes to be used for assessment and licensing is also discussed.

A generic pre-qualified platform is an equipment platform that is a basic system (consisting of hardware, operating system, functions, tools, etc.) on which applications are constructed and this platform is qualified for a wide range of applications before it is used for a specific application. These platforms can be custom designed platforms for nuclear power plant applications or they can be commercially available platforms. The advantages of generically pre-qualified platforms are derived from the fact that the platform is demonstrated to be acceptable for a wide range of applications, and is plant independent, before trying to implement a specific application. Consequentially, the risk of acceptance failure of a particular application using the pre-qualified platform is reduced since only the application being put on the platform has to be evaluated and accepted. This also makes the assessment of the application more cost effective since the platform portion does not need to be assessed for the specific application as long as that application falls within the range of applications for which the platform was pre-qualified. The disadvantage is that it is more difficult and costly to define and carry out the evaluation and acceptance of the platform for a wide range of applications compared to doing so for a specific application, but this disadvantage rapidly disappears if the platform is used for several applications in one or more plants.

Pre-qualified devices are simple software based devices such as single loop controllers, smart instruments, and recorders that are qualified for a range of applications in advance of their implementation for a specific application. The advantage is that the device is

demonstrated to be acceptable before the plant decides to implement it, so the cost and risk of the assessment and acceptance of the application of the device is minimized. The disadvantage is that there are no cost savings, and in fact would probably cost more, unless the device is used for more than one application in one or more plants or for an application in more than one plant and the qualification cost is shared.

Pre-qualified applications are plant applications that are qualified in advance of their implementation at a specific plant. The advantage is that the application is demonstrated to be acceptable before the plant decides to implement it so the cost and risk of the acceptance of the application is minimized. The disadvantage is that there are no cost savings unless the identical application, or at least sufficiently close to identical that the pre-qualification can be used with minimal extra effort, can be implemented in more than one plant.

Pre-qualified tools are software based tools, for developing or assessing applications in a more cost effective manner, that are qualified in advance of their use for a specific application. The advantage is that the tool is demonstrated to be acceptable before it is used to develop or assess an application so the cost and risk of the acceptance of the use of the tool for application development or assessment is minimized. The disadvantage is that there are no cost savings unless the tool can be used for multiple applications, or for identical applications, or at least sufficiently close to identical that the pre-qualification can be used with minimal extra effort, implemented in more than one plant.

6.2. GUIDELINE FOR GENERIC PRE-QUALIFICATION OF EQUIPMENT PLATFORMS AND CONFIGURABLE SYSTEMS

For complicated devices, such as programmable logic controller (PLC)-based or microprocessor-based platforms to be used for many safety applications, it may be advantageous to develop a specific guideline for the generic qualification of that type of equipment platform and get acceptance of the guideline by both the utility and licensing authority before the qualification is done. This is suggested since these platforms will be the keystone for many major I&C modifications in the plant and will require substantial qualification efforts. If an approach for the qualification, such as would be as specified in the guideline, is not previously agreed upon by the utility and licensing authority, there is the risk that the substantial qualification effort performed will not be successful. In that case, even more assessment work would be required and doing it later would be even more costly than if it had been done during the initial qualification activities.

For example, the generic qualification of commercially available PLC-based platforms for a wide range of nuclear power plant applications is an approach that can lead to the cost effective assessment and implementation of applications using these platforms in nuclear power plants. Traditionally, each time a new digital system is implemented for safety applications in nuclear power plants, the platform, on which the application is implemented, must go through evaluation and acceptance by the utility and licensing approval by the licensing authority. If the platform on which an application is implemented can be generically pre-qualified and accepted by the licensing authority for the range of applications covered by the qualification activity, then the cost and risk of implementing the application is reduced as long as the application is covered under the pre-qualified range of applications.

In this situation, the utility needs only to be concerned with the application software being put on the platform. This is an easier case for the utility for two reasons. First, there is no risk that the platform will not be able to be licensed, nor is the addition cost for the platform qualification required in the implementation project. Second, the utility usually has a better understanding of the application software than of the development and inner workings of the platform. This makes it easier and less costly for the utility to obtain regulatory acceptance of the application than it would be to gain the acceptance for both the platform and the application. Therefore, with the use of a pre-qualified platform, the assessment and acceptance is easier, less costly, and less risky.

The cost of the generic qualification of the platform for a wide range of applications is usually higher than the qualification required for using it for a specific application. However, if the platform is expected to be used for many applications by a utility or by many utilities sharing the cost of the generic pre-qualification, then the generic pre-qualification is a cost effective assessment approach. The pre-qualification has an additional benefit. It removes the risk of licensing the platform before the actual application is being implemented and also removes possible delays that could occur due to licensing delays on the platform. These delays could impact plant outage time and be very costly to the utilities. The supplier of the platform may also be willing to pay for the generic pre-qualification and licensing authority acceptance of the platform since this is a major advantage in selling the platform and for getting jobs to develop and implement applications in the plant.

An example of a guideline that was developed for the generic qualification of a platform is the one developed by EPRI and described in EPRI TR-107330 [44]. This guideline was submitted to the licensing authority (US NRC) for approval. After this guideline was approved by the US NRC, three platforms (Common Q System, Teleperm XS System, and Tricon System) were generically qualified and submitted to them for approval. These platforms and their qualification are described in three reports EPRI TR-110045, EPRI TR-114017, and EPRI 1000799 [45–47]. All three received favorable safety evaluation reports from the US NRC. Some other papers describing these and other systems and their qualification, both in the United States and in other countries, are given in [48–57].

6.3. PRE-QUALIFICATION OF SOFTWARE BASED DEVICES

For small component devices like single loop controllers, recorders and transmitters, requiring commercial suppliers to conform to nuclear industry standards is usually not a viable alternative since the profit on the device does not make it worthwhile for the supplier to go through the nuclear qualification effort. So a qualification and acceptance approach must make the most of what the suppliers have to offer, and where feasible, supplement it to obtain an acceptable solution. This would be the same for a utility user or for a third party that wants to qualify a device and sell it to nuclear utilities pre-qualified. The COTS qualification and acceptance framework described in an earlier Section provides a framework under which various attributes of a device are evaluated. The process should look at physical, performance, and dependability attributes. These include hardware and software design features, as well as operating history and the processes used by the commercial vendor for development, integration, testing, and configuration control. It tailors the scope and rigor of the evaluations based on the complexity and safety significance of the device and the application.

The framework provides only a general generic proposal and thus the utility or third party qualifier must ensure that the specific documentation of the details of the qualification and acceptance process and the identification and verification of specific critical characteristics for the commercially available product are available, including a combination of design verification information, performance testing and successful comparable operating history. The framework will still leave some questions unanswered until actual equipment and applications are worked on. These include the following questions. How should the safety-significance and complexity criteria be applied? How can the qualification and acceptance take credit for operating history that is obtained outside the nuclear industry? How can strengths in one area be used to compensate for weaknesses in another? How should engineering judgments be documented? So far work has been done in the USA to generically qualify two single loop controllers and a smart pressure transmitter. These results are given in the reports EPRI 1001094, EPRI 1001450, and EPRI 1001468 [58–60].

It is clear that while the approach in the framework can be used for generic qualification and acceptance over a range of applications, the utility will have to make sure that its application is covered by the range of applications in the generic qualification.

6.4. PRE-QUALIFICATION OF APPLICATIONS

The pre-qualification of an application will have to follow the same type of approach of qualification and acceptance that is done for software based platforms and devices. The pre-qualification of an application will only have a benefit if the application can be used in more than one plant will little or no modification. Due to significant plant differences that frequently exist, the applications for generic pre-qualification must be chosen carefully.

6.5. PRE-QUALIFICATION OF TOOLS

The pre-qualification of a tool will have to follow the same type of approach of qualification and acceptance that is done for software based platforms and devices. The pre-qualification of software based tools, for developing or assessing applications in a more cost effective manner, has the benefit that the cost and risk of the acceptance of the use of the tool for development or assessment is minimized. This is on top of the cost effective ness of using good tools for development or assessment. The pre-qualification of the tool only has benefit if the tool can be used for multiple applications at one or more plants.

6.6. GUIDANCE ON THE USE OF PRE-QUALIFIED PLATFORMS AND DEVICES FOR COST EFFECTIVE ASSESSMENT

In order to maximize the cost effective ness of the assessment of an application when a generic pre-qualified platform or device is used, it is necessary to take advantage of the qualification activities performed during generic pre-qualification. It is desirable to have a thorough understanding of what is needed to qualify the application on the pre-qualified platform or device. A guideline, such as the one in EPRI 1001045 [61], could be developed explicitly to define the activities required for the application qualification and indicate how to determine what part has already been done as part of the generic pre-qualification. This guideline will support cost effective assessment in two ways. First, it will help assure that the all of the necessary activities for acceptance are performed so that additional qualification

work will not have to be done later when it is more expensive. This would be more expensive due to the need to re-establish test configurations, pay for test personnel, and so on. Second, it will point out how to take advantage of the work done during the pre-qualification to reduce the amount of assessment work required for the specific application.

The guideline on the use of pre-qualified digital platforms for nuclear power plant applications in EPRI 1001045 [61] established a checklist for use in applying the guidance given in that report. This checklist is provided in Appendix E.

6.7. PRE-ACCEPTANCE OF METHODS AND PROCESSES

The development of methods and processes by industry to meet regulatory requirements may be desirable for three reasons. One, it can add clarification to the regulations so that it is easier to understand how to meet the regulations. Second, it can define a method or process to meet the regulations where the regulations only tell the levels of acceptance that must be achieved but do not give guidance on how to achieve them. Third, it can define a more cost effective approach than approaches indicated by regulatory guidance. Successfully obtaining the licensing authority's pre-acceptance of these methods and processes will support more cost effective assessment. It will do so by defining an acceptable way so that when the method or process is used, unnecessary work is not done. Also there is no risk that the method or approach will be unacceptable to the licensing authority causing assessment re-work.

A very effective approach for developing and gaining acceptance of methods and processes is to put together industry groups to define the method or process. This industry group is more focused and has a shorter time objective than standard committees since it is not developing a standard. In addition to having industry members in the group, it is important, when possible, to include the licensing authority in the group. If that is not possible, then it is important to have frequent meetings with the licensing authority to discuss the developing method or process to get an understanding of the licensing authority's issues and concerns to make sure that they are addressed. This industry-wide approach has three major advantages. First, the method or process is most likely of higher quality and addresses the needs of the entire industry better than if it is written by a single organization. Second, early and often interactions with the licensing authority makes sure that the licensing authority's issues and concerns are taken into consideration when the method or process is being developed and increases the likelihood of the acceptance by the licensing authority when it is submitted to the licensing authority for formal evaluation and approval. Third, working as an industry group is a more effective way to interact with the licensing authority than is a single organization.

Some of the successful examples of this industry group approach in the United States, which have been mentioned earlier in this report, are the "Guidelines on Licensing Digital Upgrades" given in EPRI TR-102348 [29], the "Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications" given in EPRI TR-106439 [30], and "Generic Requirements Specification for Qualifying a Commercially Available PLC for Safety related Applications in Nuclear Power Plants" given in EPRI TR-107330 [44].

7. SAFETY AND RELIABILITY ENHANCEMENTS

7.1. INTRODUCTION

Apart from good engineering practices, there are various methods, which can be used during design and implementation of software based I&C systems to enhance their reliability and thereby also the safety of the nuclear power plants where they are installed. These methods include, but are not restricted to, the following:

- good software design practices,
- on-line monitoring and diagnosis of the I&C system,
- inclusion of fault detection and identification features in the software,
- use of separation, redundancy and diversity,
- fault tolerant design.

These methods are described somewhat more detail below.

In using the methods it has to be understood that some of them may apply only to a certain phase in the life-cycle of the software based I&C. The credit that may be taken from the application of a specific method should also be seen in a larger context, when the confidence in the I&C is established.

7.2. GOOD SOFTWARE DESIGN PRACTICES

One practice that have been used in many I&C projects both in nuclear power plants and the conventional industry is to separate between the software used by a common platform and the software used to adapt that platform to a specific application. This separation between system and application software should at least be logical and can in practice be realized in different ways. One possibility is for example to define the application as a sequential execution of system modules, which then could be implemented as a data string giving reference to the modules to be executed. This solution has an additional benefit of achieving a complete separation between the system and the application software.

Another good design principle is to build the real time database of plant variables to be accessed with standardized and well-tested mechanisms. This principle also has the benefit of achieving a clear separation between software and data. These general principles are described in more detail below.

7.2.1. System programming

System programming is concerned with the establishment of the basic functions of the hardware and software platform. This phase may be completed even years before a specific I&C application project is started. Because the platform is going to be used for several perhaps slightly varying applications care should be to make it generic enough for its intended purpose. It is also important that a thorough V&V process is undertaken to ensure that the platform has the intended quality. It is necessary to document the V&V process accurately

enough to make it possible to at a later instant check the validity of certain assumptions in the confidence building process when the platform is used for a specific application.

The system programming phase is similar to any real time computer application with the only exception that also this phase should be scrutinized by a licensing authority. An extensive use of good software engineering practices has a potential of ensuring a high stability of the platform and thus also reliability of the software. The accumulated experience in software based systems has generated many good practices. These practices include, but are not restricted to, the following:

- use of structured programming,
- use of high level programming languages,
- restricting the complexity of program modules.

When the same hardware and software platform is being used for several applications it may be cost effective to invest more efforts in its V&V, because these costs can then be shared between the applications.

7.2.2. Application programming

Adapting the hardware and software platform to a specific application can be done in the same way as the system programming by using an application library for the functions and some configuration tool to define specific parameters. More often however, a specific application language is designed as a part of the development of the platform. This application language is then adapted for the I&C engineers to ease the task of defining the required functions.

There have been efforts to make the application oriented programming functional and thereby at least to some extent independent of the hardware and software platform to be used. Some vendors have used graphical software packages both to enhance programming productivity and to decrease the likelihood of errors in the resulting software. If the application program consists of modules that are very simple it may even be argued that they are deterministically correct, i.e. it is not necessary to consider the possibility of an error in those modules. If possible it is also a good practice to separate the application program from the system program for example by using separate PROM's for that purpose.

7.2.3. Real time database

A clear separation of the real time database from other variables and data has the potential of structuring the software to make it more understandable. This practice will also make the collection of historical data very straightforward. The real time database may be distributed to several computers, but to consider it as a logical entity makes it easier to handle. It is also possible to use well-proven data base techniques to ensure that any modifications are consistently brought into the application software.

Dividing the real time database into logical parts to reflect the safety classification of variables can also ease the V&V and therefore also the licensing. In this case the mechanisms

for changing data in the real time database should be accurately specified to make it very unlikely that an application program belonging to a lower safety class can change data belonging to a higher safety class.

7.3. ON-LINE MONITORING AND DIAGNOSIS OF THE I&C SYSTEM

On-line monitoring and diagnosis are straightforward methods to increase the reliability of software based and also analog I&C. The benefit of the software based systems in this context is that it is far easier to integrate these functions to be a standard feature of the I&C system. This can be used both for the detection of failure in function and the occurrence of unintended functions. One measure of the success of the on-line monitoring and diagnosis is the ratio of revealed to unrevealed faults. These desirable features are included at the cost of increased complexity and consequently make the evaluation of the design and acceptance of the system more difficult.

On-line monitoring and diagnosis methods include, but are not restricted to, the following:

- the use of watch dogs to signal problems with hardware modules and when designed to do so initiate a switch over to a standby module,
- consistency checks of signals coming from the process at a transmitter level,
- signal the break down of communication buses and initiate a transfer to a redundant one,
- signal time outs if single computers have not completed the execution of pending programs in one time slot.

It is also possible to include on-line monitoring and diagnosis at a higher level by introducing proper application programming in the I&C system. This could for example include consistency checks between redundant sensors or a combination several physical variables.

7.4. INCLUSION OF FAULT DETECTION FEATURES IN THE SOFTWARE

The inclusion of fault detection features in software design and particularly in the design of high integrity applications is a long established practice.

Fault detection in software development includes, but are not restricted to, the following practices:

- have a general fault handling policy for all modules and if necessary alarm or mark values as doubtful,
- a check of input variable validity is done whenever a software module is entered and output variable validity when the modules is left,
- for applicable parts calculate checksums to check that memory content is not to be changed,
- ensure that inadvertent user inputs is signaled and does not cause unwanted functions.

It is good practice to use methods such as fault tree analysis and failure mode and effect analysis as laid out in Appendix B.1 to identify possible faults and build in barriers for their detection and confinement.

7.5. USE OF SEPARATION, REDUNDANCY AND DIVERSITY

Separation, redundancy and diversity are used as general principles to improve reliability of equipment and thereby safety of nuclear power plants. The separation principle is used to protect from a propagation of some fault that has occurred. Redundancy again is intended to introduce failure tolerance either by using a voting logic for actions or by a switch over to standby functions. Diversity is used to abate the possibility of common cause failures (CCF). The separation, redundancy and diversity principles can be used both for hardware and software to increase the reliability of the I&C and therefore also the confidence that it is fit for its purpose.

A failure occurs whenever a system is not able to perform its intended function or executes an unintended function. Such failure is in software based I&C typically caused by a errors in the code. The separation, redundancy and diversity principles can also be used to protect the code, but the exact way in which this is done depends however very much on the application. Sometimes it is possible to use a backward recovery based error processing to trace and correct design faults.

7.5.1. Challenge of common cause failures

In using the separation, redundancy and diversity principles a common aim is to avoid common cause failures (CCF). There is always some subtle mechanism that may be argued to introduce a threat that two or more redundant channels will fail simultaneously and therefore CCFs pose the largest challenge in the design of high integrity software. Some national licensing authorities have even required that simultaneous fault should be assumed with probability one in doing the analysis of I&C functions that depend on software.

It is actually prudent to assume that there are mechanisms, which may lead to CCFs, because in distributed software based I&C system there is usually some commonality. This commonality could be connected to for instance data sharing at some level, the same software modules in redundant channels or the same specifications for diverse software modules.

The next section briefly discusses how protection can be built against common cause failures in software based I&C systems. An assessment of adequacy of diversity methods can be found in [62]. A more comprehensive treatment of possibilities and restrictions in the use of diversity to achieve reliability of high integrity systems can be found in Appendix C. Basically the challenge of CCFs can be approached by the following principles:

- Designing a system of simple components that are simple to design and require minimal verification and validation.
- Designing a system with a high degree of independence between functions so that when failures occur the effects are confined.
- Designing software using methods that further simplify verification and validation.

- Developing diversity strategies that allow demonstration of high functional reliability for critical functions without extremely high levels of verification and validation.
- Using standardized system components such that all I&C functions can be performed by configuring one on a very few types of standard devices into a system.

7.5.2. Protection against common cause failures

Common cause failures occur when two or more software components fail in exactly the same way and at the same time. Common cause failures occur when there exists at least one input combination for which the outputs of the two versions are erroneous, and the outputs are identical for all possible input combinations. Protection against CCFs can be achieved only when the causation mechanisms are identified and broken by a suitable barrier. Such barriers can be technical or administrative. A technical barrier could be built into the code to make it likely that a failure will not propagate to cause the assumed down stream consequences. An administrative barrier could be built into the software design practices or the quality system to make it likely that the possibility for various CCFs have been minimized or that their consequences are confined not to cause a simultaneous fault of redundant channels.

In assessing various mechanisms for CCFs the following six categories of dependent failures may be distinguished [62]:

(1) Shared equipment dependencies where one system is a support system for others or a component is shared by several systems. Failure of the support system leads directly to complete or partial failure of all the supported systems.

(2) Functional dependencies where the operation or non-operation of a system affects the ability of another system to be operated and may therefore cause its unavailability.

(3) Common cause failures where the availability of support or mitigating systems is affected by an initiating event. This includes major energetic external events but also equipment related transients.

(4) Physical interaction failures where the environmental effects caused by a failure cause other systems to fail.

(5) Human interaction dependencies where an operator error affects the operation of more than one system or component.

(6) Common cause failures (CCF) where two or more identical or similar components fail at the same time because of some common cause not covered by explicit modeling of the types of dependencies given above.

In recognition of the possibility of CCFs it is a good design practice to be careful in the design of common software modules, architectures, algorithms, development method, tools, implementation methods, staffing and management. It is also important to have a prudent approach to requirements to make them properly understood and correctly transformed to the software specification resulting in a decreased risk for software CCFs. This is important, because deficiencies in software can be due to incorrect, incomplete, and inaccurate or misunderstood software requirements. Design errors or software faults can be present in diverse versions, due to common human factors such as training, organization, thinking processes and design approaches.

There is no reason to expect that the use of different simple development tools or methods will reduce significantly the risk for CCFs even in multiple-version software components. All evidence points to the fact that independently developed software that uses different programmers, programming languages, and algorithms but computes the same function i.e. satisfies the same functional requirements, cannot be assumed to fail in an independent fashion.

7.6. FAULT TOLERANT DESIGNS

Suggestions have also been forwarded to reach high integrity by the use of fault tolerant designs. The perhaps most important step in this direction has been to implement software based I&C in a modular fashion with distributed functionality and local intelligence. Using a suitable hardware and software platform it should also be relatively easy to introduce true fault tolerance in a cost effective manner. The amount of credit that can be given to such solutions may however be more disputable.

To achieve a high dependability for programmable I&C systems in nuclear power plants, especially in safety critical applications, the extensive application of fault avoidance and fault removal techniques and methods are important. Fault avoidance should also include techniques that are used to minimize the number of faults in the system during the early phases of its production. Fault removal comprises activities like testing, verification and validation that are applied to detect and remove faults from the final product.

How efficient the fault avoidance and removal measures even may be, they are not sufficient to guarantee the required high dependability of a software based I&C system. The hardware will always be prone to random failures, and in practice it is considered to be nearly impossible to produce totally faultless software. Equally important, therefore, is to include in the system fault tolerance mechanisms and structures both for hardware failures and for residual software faults. For safety systems the single-failure-criterion is set, i.e. the system is required to be designed so that no single failure is to prevent the safety system actuation when needed, nor shall a single failure cause a spurious activation.

7.7. ON THE USE OF THE METHODS

It is a part of sound design and implementation practices of software based I&C systems to utilize various methods to increase reliability and safety. The methods to be used will however vary considerably depending on the specific module or function in consideration. It will also vary with the lifecycle of the I&C. Useful parameters can be deduced from available experience and can help in selecting methods, tools and languages to b e used.

One of the cornerstones in building fault-tolerant systems is the principles used to handle CCFs. Qualitative reliability assessment can help in approaching software development issues, such as the structure of the software process and the results of software testing. The N-version programming does not seem to be a viable way to approach the software CCFs, because there are too much remaining uncertainties even if systematic design methods have been used. This implies that the only remaining strategy is to minimize the probability of CCFs in basically two ways, firstly to increase the quality of software by

reducing the possibility of mistakes and inconsistencies, and secondly to eliminate the potential causes of related faults.

In a consideration of the possibility to use diversity to increase the reliability of software based I&C systems it seems fair to conclude that there is not enough evidence for the efficiency the approach. Instead it seems better to use a possible diverse version of the software as a test oracle for automated testing as outlined in section B.2.2.3 of the Appendix B. The use of hardware or functional diversity is a different matter, which has to be approached separately.

8. VERIFICATION AND VALIDATION

8.1. INTRODUCTION

Verification and validation (V&V) is the key in building confidence in software based I&C systems. Verification and validation can be performed both for smaller modules and for larger entities at various points in time for I&C project. Loosely verification can be said to be concerned with doing things the right way and validation with doing the right things. More stringently however the following definitions can be given:

- Verification is 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. Verification is performed throughout the software life-cycle to ensure a correct flow of information from the system requirements specifications phase through design to the operation, maintenance and modification phase.
- Validation is the testing and evaluation of the software based I&C system that aims to ensure compliance with functional, performance and interface requirements. Validation is performed to show that the system fulfils all its intentions and user needs. It is performed after each major integration phase and before the resulting system is put into service. If subsequent major changes are made to the software, then revalidation would be required before the system is returned to service.

Verification and validation will depend on a large number of evaluation methods, which are used to create evidence that a specific part of a software based system is of a high quality. This includes both an evaluation of the development process as the final product. The evaluation methods are generally of qualitative nature, although quantitative methods should be used where they are appropriate. The verification and validation should be based on all relevant evidences as documented at each stage of the system development. The confidence building process will be based on a variety of evaluation methods, including questionnaires, well-known risk analysis methods, document metrics, statistical methods etc. A set of these methods is described in Appendix B.

The verification and validation is often adapted to a lifecycle model to discuss requirements to be fulfilled and goals to be reached at the various phases in this model. This facilitates an in-depth discussions of the methods to be applied and what can be concluded as satisfactory evidence for confidence in the lifecycle phase. Finally at the end of each step there has to be an assessment, which combines the different evidences into a measure of a common

goal for the assessment at the stage of the development. This is described in more detail in Section 9.

8.2. THE APPLICATION OF VERIFICATION AND VALIDATION

The verification at a specific instant in the development of software based I&C systems means the check how requirements developed for earlier stages in the development of the system were transferred to specific designs solutions within the specific stage in consideration. These solutions could be reflected in either in the hardware or the software design. Finally, the verification aims to ensure that relevant requirements are fulfilled and that the evaluation activities by which this conclusion was reached are properly documented. This process is continued in stages to lead to larger entities of software to be further integrated in the system development process.

The validation process of software based I&C systems aims to ensure that preceding steps of verification and validation are integrated to give an overall assessment of the fitness for its purpose. The validation can be carried out for suitable large entities to establish confidence that the software design process can be continued with a reasonably small likelihood that major changes in the earlier established concepts would be required. Finally there should be a validation of the whole system to ensure that the integrated software system will both fulfill both design requirements and be applicable at real nuclear power plant.

The required range of validation process is generally dependent on the role of relevant I&C system or sub-system for safety. In software based plant safety critical control systems it is generally required to perform validation for the whole system (including hardware and software) and for the software besides that.

The software validation process there is usually a requirement to do a so called static analysis. To assure that this static analysis is cost effective and repeatable (to prove that the results do not depend on the analyst ability and approach for validation) special tools (also software based) are used.

The validation process of the whole system (hardware and software integrated) should include dynamic testing. This testing is based on comparison of outputs of the real system and a system modeled in parallel using the same requirements for its development and the same inputs. Effectiveness of the testing of different applications depends on the set of tests (inputs) which is developed for the testing.

The next section gives a brief survey of methods, which can be used to evaluate a programmable system in during all phases of its life-cycle. It includes all phases starting with the first concepts of the system and continues to the final installation of the system. The objective is to provide the developer with a set of methods, which can be used as an as early stage in the development as possible, to ensure that the development proceeds in a sound way.

8.3. OVERVIEW OF EVALUATION METHODS

Some methods and techniques which traditionally been applied in conventional risk analysis are also being increasingly used in risk analysis of software based systems and their

impact on the plant and the environment. These methods can be applied at the specification level as well as on the completed system. These methods and techniques are described and discussed more in detail in Appendix B.

Static analysis and dynamic testing are important evaluation methods used in the V&V process. Today there are various tools for static analysis, which can make the software analysis cost effective ly.

There are engineering solutions for dynamic testing process to demonstrate compliance with the specified system characteristics. A straightforward solution is to use simulation of the plant systems to which either simulated or real I&C is connected. There are described requirements for a simulator that is based on validated thermal hydraulic codes and takes into account the full plant system. Such a simulator could be a powerful tool for effective generation of test data for the complete I&C system. The analysis involves also the problem of defining and developing a complete set of input date (trajectories) which can be investigated by dynamic testing in reasonable time. Consequently the test program can not be truly exhaustive.

To direct the V&V efforts one can use a variety of software metrics, with the rationale that more complex software most likely would need more efforts before a confidence can be established. Some of these are described in the Appendix B, which also makes an evaluation of their validity as a measure of system reliability and other properties. One aspect is the complexity of a system, which also has an impact on its reliability. A procedure to measure this complexity therefore also is a method for the assessment of the system reliability. Complexity measures may also have impact on other properties of a system. A complexity measure of the specification may be used in an assessment of the system at an early development stage.

These methods can be used to form the basis for estimating the reliability of a piece of software to be used in a probabilistic safety assessment (PSA). There are various software reliability models which can help in this respect, but it is often in practice necessary to use expert judgement to establish a subjective belief that a specific software has a reliability that is better than some specified number.

One advantage of using pre/developed software is that there often exists experience from a variety of users of these systems. If this experience is documented, e.g. in the form of failure or other problem reports, this can be utilized in the assessment of these products. Other information of interest is the number of versions of the system, the number of installations, the spectrum of users, the relevance of previous use etc.

The final part of the Appendix B is devoted to the so called Bayesian Belief Net (BBN) methodology, which is suggested as providing a consistent way of combining evidence coming from many different sources.

8.4. VERIFICATION AND VALIDATION IN DIFFERENT LIFECYCLE PHASES

There exist various, although similar, lifecycle models, described in various software engineering guidelines, such as e.g. [63] or the IAEA safety guideline [13]. The philosophy

behind these models is very similar, although they may have different shapes, such as the 'waterfall model', the 'V-shaped model', the 'spiral model' etc. Common for these models is that they consist of a set of lifecycle phases, and the relations between them. These relations describe some ordering between the phases. This ordering is often, but not always, a chronological ordering. In each of these phases there is a certain goal one wants to achieve. These goals are subordinate a common goal for the completed system. A cost effective way is to evaluate whether the main goal is reached by the final product is to evaluate whether the sub goals are reached in each phase of the development. In this way one can prevent large effort in correcting the final program if it does not fulfill its goal.

Table 8-1. Goal in each phase

Phase	Goal
Concept	*Preparedness* on safety and computer aspects.
Scope Overall requirements/ Scope of risk analysis	*Completeness* in preparation to initiate development
Hazard and risk analysis	*Safety* of concept
Requirement specifications	Dependable specifications
Project planning	High quality plans
Overall system design	Dependable design methodology
System realization Detailed design Programming/ System integration	Dependable system
System validation	A safety validated system
System installation	A safe system in operation

As an example we will use a lifecycle model of waterfall type model, based on the one given in IEC-61508. The main goal is here *safety*. At each of the phases there should be a check that an intended safety goal is reached.

A set of lifecycle phases is given in Column 1 of Table 1. Column 2 of that table lists the safety goals one wants to obtain at each phase, and which is the basis for an evaluation of the development at that stage.

Notice that this list contains few references to software. The intention is that the milestones should be connected to the system development as a whole, and that particular software aspects should be treated as details in this development. However, particular emphasis is put on software aspects.

A short summary of these phases:

The *concept* phase occurs when a company seriously considers to install a (computer based) system, or replace an already existing system with a computer based one. At this stage there is no detailed specification of the target system, only a more general description of what

it is intended to do. The phase goal *preparedness* expresses the confidence at this stage that the company has properly considered the safety aspects of introducing a computer based safety system, and also that that they are experienced enough to be able to handle the safety aspects throughout the system development.

At the *overall requirements* phase there should be a set of functional requirements, i.e. a description of what the system is intended to do, as well as preparations for hazard and risk analysis. The phase goal *completeness* is a measure of how complete the preparations for the further development work are.

Based on the documentation from the two previous phases, a *hazard and risk analysis* should be performed. The objective is to determine the hazards and hazardous events the system may impose on the plant for all reasonably foreseeable circumstances including fault conditions and misuse. The phase goal *safety of concept* is a measure of the confidence that the specified system can be developed into a system which fulfil safety requirements, provided that necessary safety requirements are added and implemented into the system.

The next step is to make complete *requirement specifications,* including both the functional- and the safety requirements. The phase goal *dependable specification* is a measure of the degree to which the results from the safety analysis are incorporated and conform to the general safety and reliability requirements.

The assessment objective of the *project planning* phase is to show that the company can demonstrate that it has made development plans to fulfil the requirements identified at the previous phases. However, the project planning is difficult to place as a fixed step in the development process, and may be seen as a parallel activity to the actual development, and therefore assessed at various stages in this process.

At the *overall system design* phase the company shall document how the system shall be made to fulfil the requirements. Aspects to be evaluated in the assessment are design methodology, system architecture, the use of commercially available equipment (hardware and software), available design supporting tools, etc. The phase goal *dependable design methodology* is a measure on the confidence that the design fulfils the dependability requirements stated in previous phases.

The *system realization* phase is when the detailed design, programming and integration of the system are completed. Traditionally, it is often at this stage the safety assessment, and contact with regulating body, have started. The objective is to assess whether the requirements are fulfilled by the system, and to propose testing and other verification activities, which are required to fulfil the safety requirements.

At the *system validation* phase all the tests and other verification activities proposed have been completed, and the results evaluated. The phase goal *validated system* is an expression of the degree to which the dependability targets required for the system are actually reached.

At the *system installation* phase the system is installed at the plant and ready for use. Assuming that targets at the previous phases have been reached, the approval to use the system depends on an evaluation of the installation procedure and the site testing.

However, to ensure that this is a safe system also in future operation, a re-evaluation of the *maintenance and modification plan* is also required. This should also be approved before start up of the system.

8.5. V&V IN THE CONFIDENCE BUILDING PROCESS

The verification and validation is the most important part of the confidence building process and it should therefore be planned in detail before the project is started. It is also important that the V&V is given enough resources during the whole project. If the V&V for some specific part of software I&C systems is not carried out to the necessary standard it may even jeopardize the whole I&C project, because important information on some piece may be lost, which means that this part has to be scrapped.

The V&V process can be rather complex if the software based I&C is supposed to cover very many functions in several safety classes. The only way to handle this complexity is to ensure that a proper quality has been built in all earlier modules before they are used as stepping stones to implement the next level of functions. In the V&V process a special concern should be given to the management of changes. They should evidently be done according to the used quality assurance procedures, but it is also necessary to be aware of the fact that any change has the potential to introduce additional changes, which have to be handled properly.

The V&V efforts associated with specific modules should be in parity with their safety importance. In applying this general principle care should however be given to identify modules which have the potential to introduce CCFs. These modules should be given a higher importance that their direct safety importance may imply.

Finally it should be noted that confidence in a system could be achieved only if the final validation of the whole integrated system do not reveal any major problem. If that is the case it is very likely that large parts of the project of designing and implementing software based I&C have to be reopened without any considerations for project time schedule or costs.

9. THE LICENSING PROCESS

9.1. A BASIS FOR THE LICENSING

The licensing process is based on an authorization from the society through laws and decrees to ensure that the nuclear power plants fulfill necessary and sufficient conditions for safety. The licensing of software based I&C systems is according to this view related to the functional importance of the I&C both in ensuring that safety functions are introduced in a timely manner and that no unintended functions can lead to uncontrolled safety threats.

According general safety principles an attention to objects, components, systems and functions should be given, which corresponds to their safety importance. This general principle is reflected in the classification systems as used at the nuclear power plants to govern the work processes by which the objects, components, systems and functions are

designed, operated and maintained. This principle is also reflected in the so-called risk based regulation as adopted by many licensing authorities in the world.

Other high level safety principles are the defense in depth and the single failure criterion. This general safety principles are reflected in an overall design philosophy for each plant, which typically is documented in the safety analysis report (SAR). Furthermore the general safety principles are reflected in specifying requirements and recommendations for the I&C. The licensing process is therefore to a large extent concerned with a collection of evidence that these requirements have been fulfilled and that the recommendations have been followed in the design of the I&C.

These requirements and principles for the I&C are further broken down into more detailed requirements, which are connected to the special considerations that digital I&C brings into place. In general this can be viewed of a system of requirements defined on different hierarchical levels, which will be verified to be fulfilled during the V&V process. A final step in the licensing process is to collect all the evidence available for a final statement on the acceptability of the proposed solutions.

In establishing the basis for licensing and the licensing requirements to be applied it should be observed that very detailed and prescriptive requirements may tend to transfer parts of the safety responsibility from the licensee to the licensor. In the licensing it is also important to separate between design requirements and design targets.

9.2. DETERMINE IF PRIOR REGULATORY APPROVAL IS REQUIRED

The Utility must have a firm knowledge and understanding of the applicable laws and decrees to be able to define all activities, documents and materials requiring approval of the Licensing Authority prior the start of the proper software based I&C systems. Typical practices call for a prior notice to the licensing authority on all modification on safety or safety related systems. Due to the special characteristics of digital I&C it may however be prudent to discuss also standard components to be installed in non-safety system if there is a danger that CCFs at the same time may render several components or systems non-functional.

Agreed practices usually define the necessary documentation to be submitted to the licensing authority and in the required form and level of detail. It is evidently in the interest of the utility to ensure that the submitted documentation has a high quality and that any additional material needed do not cause unexpected delays. Finally it evidently necessary to ensure that proposed solutions are realistic, not to cause a rejection of the approval after considerable investments for the unacceptable route of a software based I&C system has been made.

In considering projects to design and implement software based I&C at nuclear power plants it is mostly easy to identify two general groups, one where it is clear that an approval from the licensing authority is needed and the other where it is clear that it is not needed. Between those groups there is however projects of smaller importance, but where it still may be prudent to at least notify the licensing authority when plans are materializing. For all projects it is usually a cost effective policy to establish an early contact between the utility and

the licensing authority to establish the requirements base and to agree on details of the licensing process.

9.3. DEFINITION OF A REQUIREMENTS BASE

Digital I&C licensing requirements in various Member States are different due to specific national requirements, but there are fundamental principles that should be adopted commonly by all users as minimum requirements. For example, all digital I&C systems for safety or safety related applications shall be developed under an approved Quality Assurance (QA) Program.

An important part in the definition of the requirements base is to agree on the standards, which are intended to be used for use of software based I&C system. In this field absolute agreement and solution between the utility and the licensing authority should be reached. When the utility and the licensing authority succeed to set-up a close co-operation in defining standards and design principles, it is possible to reach very good results and to shorten the time necessary for licensing process for software based I&C system.

Such co-operation is profitable also for the licensing authority, as it can enter to the I&C modernization process from the early beginning in an active way and to get background for safety evaluation of the software based I&C system. Different forms of interaction may be agreed, ranging from the participation of the Licensing Authority in the discussions, audits at the suppliers, to submission of specially generated documentation, such as Topical reports. Especially engineering solutions of the following areas should be included into the interactive process:

- Defense in depth concept
- Diversity
- Redundancy
- CCF (common cause failure)
- EMC (electromagnetic compatibility)
- Level of automation
- V&V (verification and validation)
- HMI (human machine interface)
- Testing

Also following must be discussed and determined as minimum with the Licensing Authority prior to the beginning of the works:

- Definition of the applicable standards for the safety analysis generation
- Determination of the responsibility for the overall concept and integration
- Determination of the principles and form of elaboration
- Division of responsibilities between the compilers of the individual parts and assignment of a coordinator
- Creation of control mechanisms and tools
- Creation of a detailed time schedule

The establishing licensing requirements for a project lasting over several years it may be necessary not to consider only present, but also future and evolving technology as additional products become available. The requirement may include a cost/benefit analysis to be used in optimizing the distribution of resources in development and testing of the system. Environmental effects such as electrical interference, seismic events, and other factors must be accounted for in development of licensing requirements for digital I&C systems in NPPs.

Advances in the digital I&C technology can occur with such rapidity that product life-cycles can often be shorter than the time required for the licensing and /or certification of the equipment for nuclear application. To date, the regulatory review process for digital I&C upgrades has largely proceeded on a case by case basis. These kinds of case by case process for licensing are sometimes cost effective and reduce the licensing duration, Although this case by case process may have certain benefits — particularly where technology is rapidly evolving and neither the industry not the regulator has extensive experience that could, in turn, be the basis for establishing generally applicable regulatory requirements for digital upgrades, a number of concern have been raised about this process. [65]

- First, the lack of clearly defined regulatory standards for digital upgrades can make it difficult for a utility to evaluate the acceptability of a particular digital upgrade and to gauge the level of effort necessary to obtain regulatory approval.
- Second, the lack of such standards can lead to inconsistent regulatory reviews that are sometimes heavily influenced by the individual reviewer.
- Third, the case by case approach to evaluating digital upgrades has proven to be a time consuming and in some cases, a resources-intensive process, both for the industry and the staff.
- Finally there is a concern that the licensing authority has not implemented a clear, consistency policy.

Indeed, digital systems have the potential for improved capabilities (e.g., fault tolerance, self-testing, signal validation, process system diagnostics) that could form the basis for entirely new approaches to achieve the required reliabilities. Because of such potential advantages, and because of the general shift to digital systems and waning vendor support for analog systems, the nuclear power industry expects substantial replacement of existing, aging analog systems with digital I&C technology. [64]

9.4. STEPS IN THE CONFIDENCE BUILDING PROCESS

The major challenge with modern I&C is the new design methods for SW to reach the quality required and special requirements for the nuclear industry. Due to very rapid development, design methods and technical solutions have not yet stabilized and a learning process seems therefore still required to decrease the costs of SW development.

Another characteristics of digital technology is that the possibility of CCFs from specifications, design process, etc. are difficult to exclude and that failures can have largely unpredictable consequences. To cope with these issues a large number of guiding documents have been developed, which should be properly reflected in the design and implementation

process of the software base I&C system. Sometimes it has shown to be difficult for utilities, vendors and regulators to select the documents to be applied in the licensing process.

The introduction and use of digital systems so far has brought important experience to be utilized. For example, on the basis of recent nuclear power plant experience with several digital I&C retrofits, the USNRC has identified the following challenges in the application of digital I&C systems [65].

- Common-cause failure in software
- Commercial dedication of hardware and software
- Possible lack of on-site plant experience with the new technology and systems
- Configuration management
- Increased complexity leading to possible programming errors and incorrect outputs
- Reliability of standard software tools
- Environmental sensitivity: electromagnetic or radio-frequency interference, temperature, power quality, grounding, smoke
- Effects on plant margin of safety

The challenges in the licensing process can be met by bringing up argumentation in response to certain important characteristics of software base I&C systems. Some of these issues are [65]:

The single failure criterion. If it is possible to argue based on a deterministic reasoning that a single failure of the I&C cannot lead to disastrous consequences for the plant, it may be enough to ensure acceptability for some applications. This would imply a consideration of detectable and identifiable failures together with their direct consequences. In using this argument it is usually necessary to consider implemented redundancy and independence of system parts.

Common cause failures. are failures, which have the potential to at the same time influence several redundant components or functions. A CCF implies that a believed independence between redundant or diverse components or functions is destroyed by some mechanisms which impairs the redundancy or diversity. If there are design principles or practices, which makes the argument believable that important classes of dependence have been eliminated, these may exempt from the need for an in-depth analysis of the possibility for CCFs.

Environmental conditions. It is important to place due consideration on all possible environmental conditions in which the I&C is supposed to function. This implies a consideration of seismic events, electromagnetic compatibility, and other environmental conditions.

Self-checking and -testing. Software based I&C has excellent possibilities for including various self-checking and –testing functions. If such functions have been included it can be argued that the likelihood that the I&C will not function or has the possibility to exhibit unintended functions. To approach this challenge the design requirements could include various schemes for self-supervision of the I&C system, such as internal self-testing, live signaling and, identifying unauthorized software modifications.

Human machine interface. It is important to consider all users of the software based I&C system to ensure that the possibility of human errors is minimized. This also includes necessary authorization procedures to give users access to intended system resources. The considerations could also include ensuring that manual initiation of safety functions always is allowed and that the conditions for overriding of automatic safety functions are accurately defined.

Verification and validation activities for the software based I&C systems should be planned thoroughly in advance, carried out according to the plan, and documented accordingly for all phases of the design and implementation. It is important that the relationships between requirements and design solutions are traceable with a proper reference to equipment classification and applied engineering procedures and tools. It is the aim of the V&V process to ensure that characteristics are consistent with system requirements. In a way one may say that the V&V aims at ensuring that the software base I&C fulfills all reasonable requirements on accuracy, completeness, traceability and robustness. If these four attributes are addressed in a proper way it should be possible also to keep the costs connected to the licensing process on a reasonable level.

In the confidence building process it is important to note that not all functional characteristics of the software based I&C may be exercised and tested during the design and implementation process. There is therefore always the possibility that some unwanted characteristics are observed later in the life-cycle. When this occurs it is necessary to go back to the original requirements, specifications and detailed design identify the cause for the observed deviations. When necessary the modifications to the I&C should be treated in the same manner as the original design and implementation of the I&C system.

9.5. DEVELOPMENTS IN LICENSING REQUIREMENTS

In recent years with the move toward digital I&C systems, greater attention has focused on developing a regulatory framework for the review and approval of such systems. Because the general design criteria for I&C systems provide high level guidance, there is considerable latitude in how these requirements are interpreted and applied. As a result, the evolution of the regulatory framework for digital I&C system has proceed on the case by case basis, as the regulatory body has reviewed utility-specific proposed applications of digital technology for I&C functions, without a clear view to existing regulatory guidance applied to large scale, safety grade systems (such as emergency core cooling system).

In the United States for example, in addition to the broad substantive standards contained in Title 10CFR part 50, appendix A, the USNRC has established a process for individual utility licensees to evaluate plant specific modification that they may wish to make and in particular, defining when such changes can be made without prior regulatory authority approval. This process, which is codified in 10CFR 50.50 and covers changes in plant hardware and procedures, as well as any new plant tests or experiments, requires individual rector licenses to assess the impact of any such proposed plant changes pursuant to several specific criteria [64]. It is necessary to use the endorsed documents. USNRC endorsed the approach taken in EPRI TR 102348 published in December 1993 by Generic letter 95-02 with two exceptions for digital I&C upgrade system approach. Because of the issue of case by case licensing the use of the endorsed approach shall be cost effective.

In Europe the present diversity of national requirements within software based I&C has been addressed in an attempt to find a common basis for licensing [25]. Other efforts to reach a better harmonization of requirements to place on new plants to be built have been initiated by nuclear utilities in their efforts to write standardized utility requirements.

9.6. PREREQUISITES FOR A SMOOTH LICENSING PROCESS

It is important to agree clear co-operation rules with the licensing authority to ensure a timely process and to avoid unnecessary cost for the licensing of software based I&C system. It is the necessary condition for good co-operation and it provides the prerequisites for the designer or utility to obtain necessary statements and permissions in time. It is the interest of the Utility to openly provide all necessary information for the licensing authority. Only the atmosphere of confidence and full frankness provides the prerequisites for the seamless progress of the licensing process without unnecessary delays and unexpected costs for additional documentation or even for additional modifications of the equipment.

The licensing process should run in parallel with the verification and validation process, which is the process where it is ensured that all requirement and criteria expected before implementation have been correctly done during implementation, and that the process and results meet all criteria. This is the way to prove and achieve enough confidence about the qualities such as the safety and reliability of the product and the work processes that have been used to create the product. In this prospect it is natural that licensing authority is mostly interested in and concerned with the V&V process and its documentation. The licensing authority makes his decision depending on the collected evidence as collected in the safety case or the safety analysis report. Cost effective ness of the licensing process implies both timeliness in the completion of single tasks in the V&V process and an efficient use of available resources in these tasks. Finally a successful completion of the licensing process relies on an mutual understanding of important characteristics to ensure between the licensing authority and the utility.

10. CONCLUSIONS AND RECOMMENDATIONS

This document has approached the challenge of finding cost effective assessment of software based I&C systems in nuclear power plants. Cost effective ness can basically be found in two ways:

(1) Solutions that improve the assessment methods or assessment process to enhance the quality of the assessment and/or to reduce the cost of the assessment;

(2) Solutions that can be done in earlier phases of the project, or in earlier projects, to reduce the amount of assessment required and/or to reduce the costs of the assessment, such as generic pre-qualification of a platform for an application or taking advantage of assessment activities done by others.

There already exists considerable experience from the use of software based I&C systems for safety and safety related applications in nuclear power plants. This experience makes it appropriate to conclude that there should not be any obstacles to use this technology even for the most demanding applications. Experience with various methods and tools by which the assessment to meet safety, reliability, and economic requirements can be supported,

also gives confidence that the assessment can be done in a cost effective way. This generally optimistic statement has to be qualified with the requirement that sound engineering methods are used in all phases of the design and implementation process of the software based I&C systems.

The major recommendations from this report are listed below. They are grouped in terms of their commonality. There are not in any order of importance. All of these recommendations are considered important to cost effective assessment of software based I&C systems in nuclear power plants. The recommendations are given here:

(1) Project planning management

- A clear, correct, and comprehensive plan for the assessment activities with well-defined acceptance criteria is needed at the beginning of the project.
- The level of rigor of the assessment processes and demonstrations should depend on the safety and economic significance of the system.
- Before the start of the project agree on standards, quality assurance (QA) and quality control (QC) procedures, change management practices, configuration control and documentation practices.

(2) Establish requirements specification

- Develop a clear, comprehensive, and unambiguous requirements specification for the system.
- Assessment of not important to safety systems can be necessary when high reliability is needed for economic reasons or to ensure that the system does not challenge safety systems
- Document the requirements specification to facilitate communication amongst project participants, vendors and the licensing authority.
- It is important that consistency and completeness of the requirements specification can be ensured. Formal and semiformal methods can often help in reaching this end.

(3) Pre-qualification

- Establish methods or approaches for the generic pre-qualification of equipment, tools, and applications and for obtaining regulatory approval of them.
- Work with industry to obtain consensus methods and approaches, and to do generic pre-qualification, and interact with the licensing authority as a group.

(4) Good assessment practices

- Learn about and use related assessment activities and licensing authority system acceptances in the home country (country in which the system is being implemented) and in other countries and make use of the information as much as possible.

- Obtain regulatory agreement on generic assessment processes for COTS-based systems to achieve the same level of assurance as systems designed under nuclear specific requirements.
- Use computerized tools to support the design, implementation, assessment, and licensing processes to be more efficient and free of errors, to make results repeatable, and to improve documentation.
- The computerized tools used should be verified and validated to a reasonable level.
- Evaluate the degree to which tests of the software cover the applicable states and inputs.

(5) The assessment process

- In each life cycle phase set up the acceptance criteria to be achieved in that phase, assess that phase, and evaluate whether the acceptance criteria is met before one starts the next phase.
- The level of rigor of assessment processes should depend on the safety and economic significance of the system.
- When applicable do a hazard and risk analysis.

(6) Regulatory interface

- Obtain a thorough understanding of regulatory requirements before the start of a software based I&C project.
- Interact with the licensing authority early and often about the assessment activities and results when regulatory approval is required.

REFERENCES

[1] INTERNATIONAL ATOMIC ENERGY AGENCY, Safety Assessment of Computerized Control and Protection Systems, IAEA-TECDOC-780, Vienna (1994).

[2] INTERNATIONAL ATOMIC ENERGY AGENCY, Reliability of Computerized Safety Systems at Nuclear Power Plants, IAEA-TECDOC-790, Vienna (1995).

[3] INTERNATIONAL ATOMIC ENERGY AGENCY, Advanced Control Systems to Improve Nuclear Power Plant Reliability and Efficiency, IAEA-TECDOC-952, Vienna (1997).

[4] INTERNATIONAL ATOMIC ENERGY AGENCY, Modernization of Instrumentation and Control in Nuclear Power Plants, IAEA-TECDOC-1016, Vienna (1998).

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

[6] INTERNATIONAL ATOMIC ENERGY AGENCY, Effective Handling of Software Anomalies in Computer Based Systems at Nuclear Power Plants, IAEA-TECDOC-1140, Vienna (2000).

[7] 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).

[8] INTERNATIONAL ATOMIC ENERGY AGENCY, Software for Computers in Safety Systems of Nuclear Power Plants: A Guidebook, Technical Reports Series No. 367, IAEA, Vienna (1994).

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

[10] INTERNATIONAL ATOMIC ENERGY AGENCY, Modern Instrumentation and Control for Nuclear Power Plants: A Guidebook, Technical Reports Series No. 387, IAEA, Vienna (1999).

[11] INTERNATIONAL ATOMIC ENERGY AGENCY, Safety Issues for Advanced Protection, Control and Human-Machine Interface Systems in Operating Nuclear Power Plants, Safety Reports Series No. 6, IAEA, Vienna (1998).

[12] 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).

[13] INTERNATIONAL ATOMIC ENERGY AGENCY, Software for Computer Based Systems Important to Safety in Nuclear Power Plants, Safety Standards Series No. NS-G-1.1, IAEA, Vienna (2000).

[14] INTERNATIONAL ATOMIC ENERGY AGENCY, Information Integration in Control Rooms and Technical Offices in Nuclear Power Plant, IAEA-TECDOC-1252, Vienna (2001).

[15] ELECTRIC POWER RESEARCH INSTITUTE, Advanced Light Water Reactor Utility Requirements Document, EPRI NP-6780 (1990).

[16] INTERNATIONAL ATOMIC ENERGY AGENCY, I&C Systems Important to Safety in Nuclear Power Plants, Safety Standards Series Working ID NS. 252, IAEA, Vienna (1999).

[17] US NUCLEAR REGULATORY COMMISSION, NUREG-0800, "Guidance on Software Reviews for Digital Computer based I&C Systems", USNRC Standard Review

Plan for the Review of Safety Analysis Reports for Nuclear Power Plants, Chapter 7, “Instrumentation and Controls" Branch Technical Position HICB-14 (1997).
[18] US NUCLEAR REGULATORY COMMISSION, Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants, USNRC Regulatory Guide 1.172 (1997).
[19] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, “IEEE Recommended Practice for Software Requirements Specifications”, IEEE Std 830, IEEE, Piscataway, NJ (1993).
[20] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Software for Computers in Safety Systems of Nuclear Power Plants, Standard No. 880, Geneva (1986).
[21] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Software for computers important to safety for nuclear power plants – Part 2: Software aspects of defense against common cause failures, use of software tools and of pre-developed software,Standard No. 60880-2, Geneva (2000).
[22] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear power plants – Instrumentation and control systems important to safety – Categorization of safety functions, Standard No. 61226, Geneva (1993).
[23] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear power plants – Instrumentation and control systems important to safety – General requirements for systems, Standard No. 61513, Geneva (2001).
[24] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Software for I&C systems of safety class 2 & 3 ,Draft Standard No. 62138, Geneva (2001).
[25] European Commission, Common position of European nuclear regulators for the licensing of safety-critical software for nuclear reactors, EUR 19265 EN, Brussels (2000).
[26] M. Kersken: Qualification of Pre-Developed Software for Safety-Critical I&C Application in NPPs, Proc. of NEA/CNRA/CSNI Workshop on Licensing and Operating Experience of Computer based I&C Systems, Hluboka, Czech Republic, Sep 25 – 27, 2001.
[27] ELECTRIC POWER RESEARCH INSTITUTE, Guideline for the Utilization of Commercial Grade Items in Nuclear Safety Related Applications (NCIG-07), EPRI NP-5652 (1988).
[28] ELECTRIC POWER RESEARCH INSTITUTE, Supplemental Guidance for the Application of EPRI NP-5652 on the Utilization of Commercial Grade Items, EPRI TR-102260 (1994).
[29] ELECTRIC POWER RESEARCH INSTITUTE, Guidelines on Licensing Digital Upgrades, EPRI TR-102348 (1993).
[30] ELECTRIC POWER RESEARCH INSTITUTE, Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications, EPRI TR-106439 (1996).
[31] ELECTRIC POWER RESEARCH INSTITUTE, Evaluating Commercial Digital Equipment for High-Integrity Applications: A Supplement to EPRI Report TR-106439, EPRI TR-107339 (1997).
[32] NATIONAL ARCHIVES AND RECORDS ADMINISTRATION, Title 10 of the Code of Federal Regulations., Part 50, Appendix B, “Quality Assurance Criteria for Nuclear Power Plants and Fuel Processing Plants” Office of the Federal Register (1975).
[33] NATIONAL ARCHIVES AND RECORDS ADMINISTRATION, Title 10 of the Code of Federal Regulations, Part 21, “Reporting of Defects and Noncompliance”, Office of the Federal Register (1995).

[34] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, “Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations”, IEEE 7-4.3.2-1993, IEEE, Piscataway, NJ (1993).
[35] AMERICAN SOCIETY OF MECHANICAL ENGINEERS, "Quality Assurance Requirements of Computer Software for Nuclear Facility Applications", ASME NQA-1a (1989).
[36] NATIONAL ARCHIVES AND RECORDS ADMINISTRATION, Title 10 of the Code of Federal Regulations., Section 50.59, " Changes, Tests, and Experiments", Office of the Federal Register (2000).
[37] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, “Quality Management and Quality Assurance Standards - Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software”, ISO 9003-3-1991 (1991).
[38] US NUCLEAR REGULATORY COMMISSION, “Use of NUMARC/EPRI Report TR-102348. “Guideline on Licensing Digital Upgrades”, in Determining the Acceptability of Performing Analog-to-Digital Replacements Under 10 CFR 50.59”, USNRC Generic Letter 95-02 (1995).
[39] US NUCLEAR REGULATORY COMMISSION, Criteria for Digital Computers in Safety Systems of Nuclear Power Plants, USNRC Regulatory Guide 1.152 (1996).
[40] US NUCLEAR REGULATORY COMMISSION, “A Proposed Acceptance Process for Commercial-Off the shelf (COTS) Software in Reactor Applications”, USNRC NUREG/CR-6421, (1996).
[41] US NUCLEAR REGULATORY COMMISSION, “Actions to Improve the Detection of Counterfeit and Fraudulently Marketed Products”, USNRC Generic Letter 89-02 (1989).
[42] ELECTRIC POWER RESEARCH INSTITUTE, “Guidelines for the Technical Evaluation of Replacement Items for Nuclear Power Plants”, EPRI NP-6406, 1989.
[43] US NUCLEAR REGULATORY COMMISSION, “Licensee Commercial-Grade Procurement and Dedication Programs”, USNRC Generic Letter 91-05, (1991).
[44] ELECTRIC POWER RESEARCH INSTITUTE, Generic Requirements Specification for Qualifying a Commercially Available PLC for Safety related Applications in Nuclear Power Plants, EPRI TR-107330 (1996).
[45] ELECTRIC POWER RESEARCH INSTITUTE, Generic Qualification of the ABB Common Qualified PLC-Based Platform for Safety related Applications, EPRI TR-110045 (1999).
[46] ELECTRIC POWER RESEARCH INSTITUTE, Qualification of Siemens Power Corporation TELEPERM XS Safety System: Compliance with EPRI TR-107330 “Generic Requirements Specification for Qualifying a Commercially Available PLC for Safety related Applications in Nuclear Power Plants”, EPRI TR-114017 (1999).
[47] ELECTRIC POWER RESEARCH INSTITUTE, Generic Qualification of the Triconex Corporation TRICON Triple Modular Redundant Programmable Logic controller System for Safety related Applications in Nuclear Power Plants, EPRI 1000799 (2000).
[48] W. Bastl, J. Brummer, E. Hoffmann, D. Wach: Safety Review of the Concept of Digital I&C System Important to Safety of Siemens/KWU, GRS-A-1921, July 1992 (in German).
[49] W. Bastl, D. Wach: Qualification of an Advanced Digital I&C Safety System, ANS Topical Meeting NPIC & HMIT, Penn. State University, USA, May 1996.
[50] D. Wach, B. Mulka, G. Schnuerer: Experience with Safety Assessment of Digital Up-grading of I&C in VVER Type Reactors, ENS International Topical Meeting on VVER Instrumentation and Control, Prague, Czech Republic, April 21-24,1997.

[51] W. Bastl, H.-W. Bock: German Qualification and Assessment of Digital I&C, Reliability Engineering and System Safety 59, (1998), 163 pp.
[52] A. Graf: Advanced Digital I&C Systems and their Application in Various Reactor Types, Trans. of ECN '98, Nice, France, Oct 25 – 28 1998.
[53] D. Wach: Independent Generic and Plant-Specific System Qualification – A methodology for Cost effective Assessment of Software based I&C Systems Important to Safety, IAEA SPM on Modernization of Nuclear Power Plant Instrumentation and Control Systems, Buenos Aires, Argentina, June 1 – 3, 1999.
[54] E. Lee, M. Waterman, H. Li: Safety Evaluation by the Office of NRC, Siemens Power Corporation Topical Report EMF-21(NP), TELEPERM XS: A Digital Reactor Protection System, Project No 702, May 2000.
[55] A. Lindner, D. Wach: Experiences Gained from Independent Assessment in Licensing of Advanced I&C Systems in NPPs, ANS Int. Topical Meeting on Nuclear Plant Instrumentation and Human-Machine Interface (NPIC&HMIT 2000), Washington DC, USA, Nov 2000.
[56] SPIN-3 system of Schneider Group, presented at the ANS Int. Topical Meeting NPIC&HMI 2000, Washington DC, USA, Nov 2000.
[57] Common-Q system of Westinghouse/ABB, presented at the ANS Int. Topical Meeting NPIC&HMI, Washington DC, USA, Nov 2000.
[58] ELECTRIC POWER RESEARCH INSTITUTE, Generic Qualification of the Honeywell UDC 3300 Single Loop Controller for Nuclear Safety Applications, EPRI 1001094 (2000).
[59] ELECTRIC POWER RESEARCH INSTITUTE, Generic Qualification of the Bailey-Fischer & Porter 53s16000 Single Loop Controller for Nuclear Applications, EPRI 1001450 (2001).
[60] ELECTRIC POWER RESEARCH INSTITUTE, Generic Qualification of the Rosemount 3051N Pressure Transmitter, 1001468 (2001).
[61] ELECTRIC POWER RESEARCH INSTITUTE, Guideline on the Use of Pre-Qualified Digital Platforms for Safety and Non-Safety Applications in Nuclear Power Plants, EPRI 1001045 (2000).
[62] Korhonen, J., Pulkkinen, U., and Haapanen, P. Diversity requirements for safety critical software based automation systems. STUK-YTO-TR 142. Helsinki 1998. 48 pp.
[63] INTERNATIONAL ELECTROTECHNICAL COMMISSION Functional safety of electrical/electronic/ programmable electronic safety related systems, Standard No. 61508 part 1 – 7, Geneva (1998-2000).
[64] NATIONAL RESEARCH COUNCIL, Digital Instrumentation and Control Systems in Nuclear Power Plants, National Academy Press, Washington D.C. (1997)
[65] Mauck, J., Regulating Digital Upgrades, Presentation to the Committee on Applications of Digital Instrumentation and Control Systems to Nuclear Power Plant Operations and Safety, Washington D.C., January 31, 1995.

ABBREVIATIONS

ACE	Abnormal Condition And Event
ASIC	Application Specific Integrated Circuits
BBN	Bayesian Belief Net/Network
BTP	Branch Technical Position
CFR	Code of Federal Regulation
CDL	Controller and Data Logger
CPU	Central Processing Unit
CCB	Configuration Control Board
CCF	Common Cause Failure
DT	Design Team
EMC	Electromagnetic Compatibility
EMI	Electro-magnetic Interference
EPRI	Electric Power Research Institute
FAT	Factory Acceptance Tests
FMEA	Failure Mode and Effect Analysis
FMECA	Failure Mode Effect and Criticality Analysis
FTA	Fault Tree Analysis
HAZOP	Hazard and Operability analysis
HDL	Hardware Description Language
HMI	Human-Machine Interface
IEEE	Institute of Electrical And Electronics Engineers
ISO	International Organization For Standardization
I&C	Instrumentation and Control
MSC	Message Sequence Chart
PROM	Programmable Read Only Memory
QA	Quality Assurance
QAP	Quality Assurance Plan
QC	Quality Control
PDF	Probability Distribution Function
PHA	Preliminary Hazard Analysis
PLC	Programmable Logic Controller
PRA	Probabilistic Reliability Assessment
PRPS	Primary Reactor Protection System
PSA	Probabilistic Safety Assessment
RFI	Radio Frequency Interference

RTL	Register Transfer Level
RTM	Requirement Traceability Matrix
SAR	Safety Analysis Report
SAT	Site Acceptance Tests
SDD	Software Design Description
SER	Safety Evaluation Reports
SEU	Single Event Upset
SHA	System Hazard Analysis
SIL	Software Integrity Level
SRP	Safety Review Plan
SRS	Software Requirement Specification
SRSA	Software Requirements Safety Analysis
UML	Unified Modeling Language
V&V	Verification and Validation
VT	V&V Team
VDM	Vienna Definition Method

APPENDIX A

Software classification methods used in Republic of Korea

This appendix gives a different classification of software which is used in Korea. Based on this classification, a graded approach for V&V is given to support cost effective assessment.

1. SOFTWARE CLASSIFICATION FOR COST EFFECTIVE ASSESSMENT

Software Integrity Level means the level of software that determines how rigorous the requirements for engineering, verifying and validating the software will be, and how restrictive design constraints and practices in the software life-cycle activities will be. In the system (or project) specific software development plan, the Software Integrity Level (SIL) of all involved software should be identified. SIL indicates the relative degree of rigor (in the application of standards, methods, and practices) required for the software development activities. The higher the SIL (i.e., the smaller the SIL number) is, the more rigorous will be the development activities and the more restrictive will be the engineering practices. For lower SIL software (i.e., higher SIL number), some of the requirements and recommended engineering practice are relaxed or left to the discretion of the developer or the individual utility. This is referred to as grading of activities or graded approach.

SIL is determined by two principal factors as shown in following table[A-1]; the system class and the software failure impact type. The system class follows the I&C system classification criteria defined by IAEA. The software failure impact type is determined by the criteria defined below.

Software Failure Impact Type	Items Important to Safety		Non-Safety System
	Safety	Safety-Related	Items Not Important to Safety
Type I	1	2	3
Type II	1	2	3
Type III	2	3	4

Type I: Any software whose failure immediately compromises or challenges plant protection functions or plant safety. Such failures, in general, require or result in an immediate reactor shutdown or an immediate reactor trip.

Type II: Any software whose failure does not immediately compromise or challenge plant protection functions, or plant safety, or nominal plant operation; but may compromise or challenge them after a period of time. Such failures, in general, result in an eventual, but not immediate, reduction of nuclear power generation; or an eventual, but not immediate, shutdown of nuclear power generation. The eventual reduction or shutdown of nuclear power generation may be based on an allowable time interval (“completion time”) in which to complete a corrective action (in order to correct a certain undesired condition that was caused by the software failure) before initiating the power reduction or power termination; in accordance with the plant technical operating procedures.

Type III: Any software whose failure does not cause or result in any shutdown or reduction of nuclear power generation; be it immediate (upon occurrence of the software error) or eventual (after a period of time from when the software error occurred). Such failures, in general, do not compromise or challenge plant protection functions, or plant safety, or nominal plant operation.

As a result of this classification, all I&C system software is identified as belonging to one of the following classes:

- SIL 1
- SIL 2
- SIL 3, and
- General Purpose (GP)

GP software is software that performs some functions other than that described in the previous classifications. This GP software includes tools that are used to develop software in the other classifications, but is not installed as part of the on-line plant operation.

Based on the SIL, the V&V planner can select for each category an appropriate set of V&V activities for each phase and describe those in the system (or project) specific plan. The IAEA assigns safety classification to the software residing within the protection system. There exists; however, some software that supports either directly or indirectly the primary function(s) of the protection system. It might not be necessary or beneficial to apply the most rigorous standards and engineering practice to this kind of support software, as long as they are well isolated and their failure(s) does not influence the execution of the primary function(s).

Higher SIL has been assigned to the software whose failure can lead to any form of power reduction. This is because the loss resulting from the incident, such as reactor shutdown or power reduction due to the software fault(s), can easily override the cost savings realized by a lower SIL designation (or classification).

Specific parts of the software in a single system may be assigned to different classes. Each separately designated part of the software must have an assigned software integrity level. The classification also makes two dimensional distinctions regarding methods applied to each of the following categories of I&C system software:

- Original(or newly developed software), developed for I&C Systems;
- Existing, to be modified; and
- Existing, not to be modified

Every I&C system software item is assigned to one of the above categories. Software in several categories may be included in each I&C system. For example, a typical computer system may rely on:

- An operating system from a commercial supplier that is existing and is to be used as is (existing not to be modified);
- Some residual code to be updated from a previous project (existing, to be modified); and
- New algorithms (originally developed).

Software to be developed and used for some projects shall be placed into the following IAEA software classes according to the classification criteria defined in Software Integrity Level (SIL) Table:

(a) In-System Software

- Safety (with SIL 1 or 2)
- Safety related (with SIL 2 and 3)
- Not Important to Safety (with SIL 2 and 3)

(b) General Purpose Software

The software designer responsible for developing I&C systems software should document the appropriate class for each software module in its Software Requirements Specification (SRS) and Software Design Description (SDD).

In addition, the existing software must be evaluated by the verifier (or safety analyzer) to meet the following criteria:

- Existing commercial off the shelf(COTS) software may be used for "Importance to Safety" (Safety, Safety-related) software if it is qualified in accordance with the following COTS qualification described in Section 5;
- Existing NPP non-commercial software that has been actively used in a nuclear power plant may be used for the safety class of software, provided it has been maintained under an acceptable quality plan with an active program for problem reporting and corrective action reporting to the I&C systems project during the software life-cycle. This software shall also have adequate design documentation, user documentation, and well commented source code. This software shall have been verified and validated under another program that is judged by the verifier to be acceptable.
- Other existing non-commercial software may be used if a review by the software designer and verifier concludes that it well organized and has adequate design documentation, user documentation, and source code commentary.

For tools (usually used for V&V) classified as General Purpose, there exists a contradiction between standards. IEEE Std 7-4.3.2 (5.3.3) and IEC 880 (8) about whether V&V tasks associated with software tools are not required, provided the software produced using the tools is subjected to V&V activities. In this respect, it is reasonable to settle the discrepancy between these standards by recommending the same level of V&V rigor only if the outputs produced by the tools becomes part of the operational system. In this case, the V&V rigor should be proportional to the software integrity level (SIL) defined by the criteria in the previous SIL Table.

2. COST EFFECTIVE V&V FOR EACH CLASSIFIED SIL

2.1. SOFTWARE REQUIREMENTS PHASE

The following table illustrates a typical grading of activities associated with the Software Requirements Phase, in accordance with the Software Integrity Level (SIL) which is assigned to the software.

SIL	Types of Review	Level of Effort	Initial Test Plan Development
1	Documentation (SRS) Review Meetings (a)	Independent Review Thread Path Audit Maintain RTM (c) Structured Reviews Use of Checklists	Yes
2	Documentation (SRS) Review Meetings (a)	Thread Path Audit Maintain RTM (c) Structured Reviews Use of Checklists	Yes
3	Documentation (SRS)	One level of independent review (b) Use of good QA practices Traceability (d) Use of Checklists (e)	Yes

Notes:

(a) Review meetings to be conducted as required.
(b) Performed by an independent reviewer. Can be from same Software Design Team(DT). Evidence of review can be via reviewer signature on document cover sheet noting "Independent Reviewer".
(c) Formal traceability via Requirements Traceability Matrix (RTM).
(d) For non-safety(Not Important to Safety) application class software, verification of traceability may be performed within software design team with documented evidence of trace process. For other software classes the RTM is to be formally maintained.
(e) As applicable.
(f) SRS = Software Requirements Specification.
(g) RTM = Requirements Traceability Matrix.

Requirements	Document A Rev. 0	Rev. 1	Document B Rev. 0	Document C Rev. 0
Requirement A	Section No.	Section No.	Section No.	Section No.
Sub Requirement 1				
Sub Requirement 2				
Sub Requirement A				
Requirement B				
Sub Requirement 1				
Requirement C				
Requirement D				

2.2. SOFTWARE ARCHITECTURE DESIGN PHASE

The following table illustrates a typical grading of activities associated with the Software Design Phase, in accordance with the Software Integrity Level (SIL) which is assigned to the software.

SIL	Types of Review	Level of Effort	Prototyping
1	Preliminary Design (g) Documentation (SDD) Review Meetings Critical Design Review (a)	Independent Review Thread Path Audit Maintain RTM (c) Structured Reviews Use of Checklists	Limited SW configuration management Adhere to Coding Standards Document prototype design Informal review within DT
2	Preliminary Design (g) Documentation (SDD) Review Meetings Critical Design Review (a)	Thread Path Audit Maintain RTM (c) Structured Reviews Use of Checklists	Limited SW configuration management Adhere to Coding Standards Document prototype design Informal review within DT
3	Documentation (SDD) Review Meetings (b)	One level of independent review (f) Use of good QA practice Traceability (d) Use of Checklists (e)	No SW configuration management Coding Standards not required Document prototype design Informal review within DT

Notes:

(a) The Critical Design Review should consider the interfaces to software of lower Software Integrity Levels (SIL).
(b) If required.
(c) Formal traceability via Requirements Traceability Matrix (RTM).
(d) For non-safety application class software, verification of traceability may be performed within DT with documented evidence of trace process. For other software classes the RTM is to be formally maintained.
(e) As applicable.
(f) Performed by an independent reviewer. Can be from same DT. Evidence of review can be via reviewer signature on document cover sheet noting "Independent Reviewer".
(g) A preliminary SW design is reviewed prior to generating the SDD.
(h) SDD = Software Design Description.
(i) RTM = Requirements Traceability Matrix.

2.3. SOFTWARE IMPLEMENTATION PHASE

The following table illustrates a typical grading of activities associated with the Software Implementation Phase, in accordance with the Software Integrity Level (SIL) which is assigned to the software.

SIL	Types of Review	Level of Effort	Testing Activities
1	Documentation (a) Review Meetings	Independent Review Code Inspections Thread Path Audit Maintain RTM (c) Structured Reviews Use of Checklists	Module test verification by VT (g) Prepare FAT procedures

SIL	Types of Review	Level of Effort	Testing Activities
2	Documentation (a) Review Meetings	Code Inspections Thread Path Audit Maintain RTM (c) Structured Reviews Use of Checklists	Module test verification by VT (g) Prepare FAT procedures
3	Documentation (a) Review Meetings (b)	One level of independent review (d) Use of good QA practice Traceability (e) Use of Checklists (f)	Module test verification by DT (g) Prepare FAT procedures

Notes:

(a) Documentation consists of code listings for: (1) originally developed software; and for (2) existing software and modifications to existing software.
(b) As required.
(c) Formal traceability via Requirements Traceability Matrix (RTM).
(d) Performed by an independent reviewer. Can be from same DT.
(e) For non-safety application class software, verification of traceability may be performed within DT with documented evidence of trace process. For other software classes the RTM is to be formally maintained.
(f) As applicable.
(g) Module test verification is performed by either the DT or the V&V Team (VT) depending upon SIL - as noted in above table.
(h) RTM = Requirements Traceability Matrix

2.4. TESTING PHASE

The following table illustrates a typical grading of activities associated with the Software Testing Phase, in accordance with the Software Integrity Level (SIL) which is assigned to the software.

SIL	Type of Testing to be Considered	Test Execution	Preparation of Final SVVR for delivered software
1	Static Testing Dynamic Testing (d) Random Testing (e) Testing of SW interface to lower SIL SW (a)	Independent Review System Test = VT/DT (b) Integration Test = VT	VT/DT
2	Static Testing Dynamic Testing Testing of SW interface to lower SIL SW (a)	System Test = VT/DT (b) Integration Test = VT	VT/DT
3	Based on specific application	System Test = DT/VT © Integration Test = VT	DT/VT (h)

Notes:

(a) Testing of all SW interfaces to lower SIL software to verify that a SW failure in the lower SIL software will not negatively impact software of the higher SIL.
(b) VT had lead responsibility, DT supports or reviews. In certain areas which VT does not have sufficient knowledge, then DT can perform with VT review and concurrence.
(c) DT has lead responsibility, VT supports or reviews. However, VT still responsible to verify that a SW failure in the lower SIL software will not negatively impact software of a higher SIL (as part of testing for SIL 1 and SIL 2 software).
(d) Inputs or test states vary with time.

(e) Random combinations of input/state conditions.
(f) VT = V&V Team.
(g) DT = Design Team.
(h) VT still prepares results of testing interfaces between software of different SIL's.
(i) SVVR = Software V&V Report.

2.5. SITE INSTALLATION AND CHECKOUT PHASE

The following table illustrates a typical grading of activities associated with the Software Testing Phase, in accordance with the Software Integrity Level (SIL) which is assigned to the software.

SIL	Site Installation Activity
1	Independent Review Installation Procedure SW Integrity Test (b) Site Acceptance Procedure (a)
2	Installation Procedure SW Integrity Test (b) Site Acceptance Procedure (a)
3	Installation Procedure SW Integrity Test (b, c)

Notes:

(a) The Site Acceptance Procedure is a subset of the System/Integration Acceptance Test Procedures which are performed prior to shipment.
(b) This test is conducted to assure the integrity of the delivered software. It may be combined with, or done in conjunction with, or as part of the Site Acceptance Test, as appropriate.
(c) Test conducted as applicable.

REFERENCE

[A-1] Yi, Woojune and Kim, Jang-Yeol, Evaluation Method of Software Integrity Level Used in KNGR I&C System, Nuclear I&C Group, Nuclear Lab., KEPRI.

APPENDIX B

Functional and process characteristics

This appendix gives each characteristic topic, a discussion of the topic as given in the United States Nuclear Regulatory (USNRC) Branch Technical Position 14 (BTP-14) which is an Appendix of Chapter 7 of the Standard Review Plan (SRP) [17], and review questions that may be asked in order to determine whether the topic has been satisfied. Some additional thoughts are also added for some of them.

1. FUNCTIONAL CHARACTERISTICS

All characteristics should be consistent with the system requirements.

1.1. ACCURACY

Accuracy requirements should be provided for each input and each output variable. Accuracy requirements should be stated numerically, and appropriate physical units and error bounds should be supplied. Accuracy requirements should include a description of data type and data size for each input and output variable. [17]

Checklist	Yes	No	N/A
Does an accuracy requirement exist for each output variable that has a numerical value?			
Is each accuracy requirement stated quantitatively and consistent with system requirement?			
Are the physical units stated for each accuracy requirement?			
Does each accuracy requirement include permissible error bounds?			
Do all accuracy requirements include data type and data size information?			
Simple data types include integer, fixed point and floating point. These can be combined to create more complex data types — for example, vectors and matrices. Data size is generally given as the number of bits required to accurately represent the variable throughout its range.			
Does an accuracy requirement exist for each input variable that has a numerical value?			

1.2. FUNCTIONALITY

Functionality requires that the operations that must be performed for each mode of operation be completely specified. Functions should be specified in terms of inputs to the function, transformations to be carried out by the function, and outputs generated by the function. [17] Are termination requirements, such as power down, shut down, and error sequences, specified?

Does the Software Requirements Specification specify completely the functional requirements for all modes and states of operation identified in the System Design Description and Safety Analysis Report? These modes and states include refueling, system installation and commissioning, test, normal operation, startup and shutdown, off-normal operation, upset and emergency operation. Also they include things such as initial value of variables, startup sequences and power up sequences.

Checklist	Yes	No	N/A
Do functional requirements include starting conditions and system status at the initiation of each function?			
Do functional requirements specify the input and output variables required by each function?			
Do functional requirements include task sequences, actions and events required to carry out each function?			
Do functional requirements include termination conditions and system status at the conclusion of each function?			
Do functional requirements include directly or indirectly the relevance of each function to system reliability and safety?			
Does the Software Requirements Specification identify those variables in the physical environment that the software must monitor and / or control? Such as temperatures and pressures.			
Does the Software Requirements Specification define the required behavior of the controlled variables in terms of monitored variables with the use of mathematical functions? Monitored variables are those the software has to measure, and controlled variables are those the software is intended to control. The entire set of monitored variables must be covered by these mathematical functions			
Do functional requirements include the purpose of each function?			
Do functional requirements include trigger conditions which cause each function to operate?			
Are initialization requirements specified?			

Functional requirements should describe:

- How each function is initiated;
- The input and output variables required of the function;
- The task sequences, actions, and events required to carry out the function; and
- The termination conditions and system status at the conclusion of the function.

User interfaces should be fully described for each category of user.

Checklist	Yes	No	N/A
Are operator interfaces fully defined? Operator interface definitions can be given in terms of keyboard inputs; control panels; positioning and layout of controls and displays; human reaction and decision times; use of colors, bold face, underlining and blinking of displays; menu techniques;			
Does each functional requirement describe how the function is initiated? This includes the trigger conditions that cause the function to operate, the starting conditions at the initiation of the function and the required system status at the initiation of the function. The requirements should be written in such a way that a single requirement specifies no more than one function.			
Can the required safety related functions be correctly implemented within the existing, available resources? Resources include budget, schedule, manpower, equipment, software tools.			
Are the specified models, algorithms and numerical techniques practical and within the state of the art?			
Are the quality attributes specified for the software achievable both for each software unit and the complete integrated software system? Such as accuracy, adaptability, availability, clarity, completeness, consistency, correctness, deterministic timing, integrity, maintainability, modularity, reliability, robustness, safety, security, serviceability, simplicity, stability, testability, traceability, understandability, uniformity, usability and validity.			
Is the relationship between the monitored and input variables, and the relationship between the output and controlled variables, precisely described?			
Are error conditions described, including required corrective actions?			
Do requirements exist to permit the operator to verify that basic system functions are operating?			
Do requirements for the use of colors, positions of information on the display screen, icons, flashing signals and alerting signals follow a consistent scheme?			
Are the variables in the physical environment that the software must monitor and control completely specified? The required behavior of the controlled variables shall be specified in terms of the monitored variables with the use of mathematical functions.			
Is there a requirement that the computer system will report its			

own defects and failures to the operator?			
Are requirements for control panels and display layouts specified?			
Are procedures required for introducing, modifying and displaying parameters to the operator exactly defined?			
Are requirements for human reactions to software-generated messages specified, including the amount of time available for making decisions?			
Is there a requirement that manual interactions shall not delay basic safety actions beyond specified safe limits?			
Is each possible input from each sensor completely described? The description should be in terms of the appropriate items in the following list: type of sensor (analog, digital); possible range of values; units of measurement; resolution of measurement; error bounds on measurements for the range of measurement; instrument calibration; and conversion algorithms. Examples of conversion patterns include analog to digital and bit patterns to physical units.			
Does the Software Requirements Specification specify the behavior of the software for anomalous inputs? This includes inputs received before startup, inputs received after shutdown and inputs received when the computer is temporarily disconnected from the process. The purpose is to cover spurious inputs which appear to come from the process under computer control, but do not actually come from there.			
Does each functional requirement specify the input and output variables required by the function?			
Are the actions required of the computer system for error recovery completely described? The description should include the type of error, the procedure (if any) for notifying the operator of the error, and the means of restoring service. The recovery procedure should not compromise safety; simply halting is generally not a satisfactory response.			
Does each functional requirement specify task sequences, actions and events required to carry out the function?			
Does each functional requirement specify the termination conditions and system status at the conclusion of the function?			
Are all output variables from functions completely described? The description should be in terms of the appropriate items in the following list: data types, formats, physical units, accuracies, update intervals, valid ranges, available options, timing, frequency of occurrence, message error rates and types, alerting signals and method of access by the software.			

Is each possible output to each actuator completely described? The description should be in terms of the appropriate items in the following list: type of actuator (analog, digital); possible range of values and units; units of measurement; resolution of measurement; calibration requirements; and conversion algorithms.			
Is each category of operator specified, including expected experience level for each category? The experience mentioned in the question refers to computer experience, not reactor experience. Less experienced computer operators may require fuller explanations on screens, less technical help messages, etc. The expected experience level can affect screen design, so should be mentioned in the SRS.			
Does the Software Requirements Specification completely specify the software interfaces? This includes interfaces to hardware, predeveloped software, commercial off the shelf software and operators.			
Are the actions required of the computer system for which fail-safe action must be taken completely described?			
Are all the operating modes within which the software must perform listed and described?			
Does the Software Requirements Specification describe the operational environment within which the program must run?			
Does the Software Requirements Specification state what the software must not do? The SRS must not contain requirements unless they are imposed explicitly or implicitly by the System Design Description or the SAR. All requirements imposed by the System Design Description and the SAR that specify actions which the software must not do must be contained in the SRS. Any "requirements" that the software not do something which are not contained in higher level documents are unlikely to be detected by the auditors.			
Are all actions required of the computer system for every mode of operation completely described? This includes startup, shut down, initialization, termination, normal operation, off-normal operation, degraded operation, emergency operation, etc.			
Are all inputs variables to functions completely described? The description should be in terms of the appropriate items in the following list: data types, formats, physical units, accuracies, sampling intervals, valid ranges, available options, timing, frequency of occurrence, alerting signals and method of access by the software.			

1.3. RELIABILITY

Reliability should specify the factors required to establish the required reliability of the software system at time of delivery. Reliability requires that all requirements for fault tolerance and failure modes be fully specified for each operating mode. Software requirements for handling both hardware and software failures should be provided, including requirements for analysis of and recovery from computer system failures. Requirements for on-line in-service testing and diagnostics should be provided. [17]

Checklist	Yes	No	N/A
Are software reliability requirements derived from the reliability requirements of the System Design Description?			
Are software reliability requirements defined quantitatively? That is, in terms of failure rate or mean time to fail criteria.			
Are requirements for fault tolerance and graceful degradation defined?			
Are reliability and availability criteria given for each mode of operation?			
Does the Software Requirements Specification contain requirements for on-line in-service testing and diagnostics?			

1.4. ROBUSTNESS

Robustness is an important goal for safety systems. But, it is, perhaps, an emergent property, hard to pin down precisely in measurable and testable terms. At least there is no such definition in standard use.

Robustness requires that the behavior of the software in the presence of unexpected, incorrect, anomalous and improper (1) input, (2) hardware behavior, or (3) software behavior be fully specified. Of particular concern is the behavior of the software in the presence of unexpectedly high or low rates of message traffic. [17]

Checklist	Yes	No	N/A
Does the Software Requirements Specification specify the behavior of the software in the presence of unexpected rates for message traffic? This includes both unexpectedly high rates and unexpectedly low rates.			
Does the Software Requirements Specification specify the behavior of the software in the presence of unexpected, incorrect, anomalous and improper input data and other anomalous conditions? This includes unexpected data types, formats, physical units, accuracies, sampling intervals, ranges, options, timing, and frequency of occurrence.			

Are error conditions and actions to be taken on error conditions specific ?			
Does the Software Requirements Specification specify the behavior of the software in the presence of unexpected, incorrect, anomalous and improper hardware or software behavior? The following factors must be considered: failures shall be identified to a reasonable degree of detail and isolated to the most narrow environment; fail-safe output shall be guaranteed as far as possible; if such a guarantee cannot be given, system output shall violate only less essential safety requirements; the consequences of failures shall be minimized; remedial procedures, such as fall back, re-try, system recovery should be considered for inclusion; reconstruction of obliterated or incorrectly altered data may be tried; and information on failures shall be provided to the operating staff.			
Does the SRS require (1) the checking of status after exit from any procedure where the status is provided and (2) appropriate action if an incorrect status is detected?			

1.5. SAFETY

Like robustness, safety too is an emergent property, hard to define in pragmatic engineering terms. The working definition used this section encompasses three facts as shown in the checklist below.

Checklist	Yes	No	N/A
Have enough requirements been specified?			
Is each individual piece of output behavior specified in sufficient detail so that, when safety review of each requirement occurs, acceptable behavior can be distinguished from unacceptable behavior?			
Has each specified output been reviewed by a competent reviewer to determine if it is safe for this system operating in its specified environment? And if not, or not always, is it still an acceptable requirement?			

Safety requires that the software functions, operating procedures, input, and output be classified according to their importance to safety. Requirements important to safety should be identified as such in the SRS. The identification of safety items should include safety analysis report requirements, as well as abnormal conditions and events as described in Reg. Guide 1.152 [39] [17].

The following factors are relevant to avoiding common mode failures [5-1]: defense-in-depth, graceful degradation, management of failures in general, functional diversity and (if necessary) software diversity, spatial separation and modularization, decoupling, logical separation.

Checklist	Yes	No	N/A
Are the software conditions which can lead to a hazardous state identified in the Software Requirements Specification?			
Does the SRS specify the input conditions and the calculations necessary as a prelude to the software initiating protective actions?			
Does the Software Requirements Specification identify and define safe and unsafe (i.e., hazardous) reactor states?			
Are requirements for validity checks on operator and sensor inputs defined in the Software Requirements Specification?			
Does the Software Requirements Specification classify sensors and actuators I/O according to their safety criticality?			
Does the Software Requirements Specification specify software actions which are necessary to prevent plant damage, including the necessary calculations and their physical background?			
Does the Software Requirements Specification classify software functions according to their safety criticality?			
Does the Software Requirements Specification specify software actions which are necessary to carry out emergency shutdown of the reactor, including the necessary calculations and their physical background?			
Are software actions specified for potential common mode failures as required by the System Design Description and Safety Analysis Report?			

1.6. SECURITY

Security should specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure.

Security requires that security threats to the computer system be identified and classified according to severity and likelihood. Actions required of the software to detect, prevent, or mitigate such security threats should be specified, including access control restrictions. [17]

Checklist	Yes	No	N/A
Does the Software Requirements Specification impose requirements to prevent unauthorized personnel from interacting with the software system?			
Does the Software Requirements Specification impose access restrictions on operators, managers and other personnel?			
Are the security requirements, taken as a whole, mutually consistent?			

Are potential security threats to the computer system identified, classified according to severity and likelihood, and documented?			
Does the Software Requirements Specification impose requirements to prevent unauthorized changes to the software system?			
Does the Software Requirements Specification specify requirements address security threats?			

1.7. TIMING

Timing requires that functions that must operate within specific timing constraints be identified, and that timing criteria be specified for each. Timing criteria should be provided for each mode of operation. Timing requirements should distinguish between goals and requirements. Timing requirements should be stated in such a way that the time delay between stimulus and response for safety actions is deterministic under normal and anticipated failure conditions. [17]

Checklist	Yes	No	N/A
Does the Software Requirements Specification specify storage tolerances?			
Does the Software Requirements Specification specify the time-critical functions and the timing criteria for each? Timing criteria include minimum times, maximum times, sampling frequencies, time intervals and timing tolerances, as appropriate. Criteria may differ according to the different modes of operation			
Are volume and throughput expectations given for the software?			
Do memory size requirements state explicitly which are merely target figures or goals and which are absolutely necessary for the software system?			
Is the software system required to have deterministic timing?			
Do timing requirements state explicitly which are merely target figures or goals and which are absolutely necessary for the software system?			
Does the Software Requirements Specification specify timing tolerances?			
Are timing requirements specified for each mode of operation?			

2. PROCESS CHARACTERISTICS

2.1. COMPLETENESS

Two major aspects to completeness in requirements specifications are very important. They are the individual requirement's completeness and set completeness.

An SRS is complete if and only if, it includes a) all significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interface, b) the definition of the response of the software to all realizable classes of input data in all realizable classes of situations, and c) full labels and references to all figures tables, and diagrams in the SRS and definitions of all terms and units of measure.

Completeness requires that all actions required of the computer system be fully described for all operating modes and all possible values of input variables (for example, the complete span of instrument inputs or clock/calendar time). The SRS should describe any actions that the software is prohibited from executing. The operational environment within which the software will operate should be described. All variables in the physical environment that the software must monitor and control should be fully specified. [17]

Checklist	Yes	No	N/A
What is the minimum set of essential information required for the complete specification of an "individual [behavioral] requirement?			
When does the existence of one or more requirements imply the need for additional requirements?			

2.2. CONSISTENCY

Specified requirements must be shown to be fully deterministic and consistent with all specified constraints.

If an SRS does not agree with some higher level document, such as a system requirements specifications, then it is not correct. Consistency requires that the contents of the SRS be consistent with the safety system requirements, the safety system design, and documented descriptions and known properties of the operational environment within which the safety system software will operate. Individual requirements should not contradict other requirements. Timing requirements should be consistent with thermal hydraulic analyses performed in the system safety analysis. Uniform and consistent terminology, notation, and definitions should be used throughout the SRS. [17]

2.3. CORRECTNESS

Specified requirements must be shown to be fully deterministic (consistent) and consistent with all specified constraints An SRS is correct if and only if , every requirement stated therein is one that the software shall meet. Correctness requires that the description of actions required of the computer system have no faults and that no other requirements be stated.

Checklist	Yes	No	N/A
Are the models, algorithms and computational techniques specified in the Software Requirements Specification mathematically mutually compatible?			
Are the accuracies required of input, computational and output data mutually compatible?			
Are the individual requirements consistent with the System Design Description and the Safety Analysis Report?			
Are requirements for similar functions mutually consistent?			
Do requirements for the use of color, positions of information on display screens, icons, flashing signals and alerting signals follow a consistent scheme?			
Has the SRS been analyzed for internal contradictions and has this analysis been documented?			
Do models, algorithms, and numerical techniques specified in the Software Requirements Specification agree with standard references where applicable?			
Are input and output specifications in the Software Requirements Specification consistent with interface requirements imposed by the hardware or pre-developed software? Specifications include data type, data size, data rate, accuracy, error bounds, and physical units.			
Are the individual requirements consistent with documented descriptions and known properties of the operational environment into which the program must fit?			
Have uniform and consistent terminology and definitions been used throughout the Software Requirements Specification?			
What is the minimum set of essential information required for the complete specification of an “individual[behavioral] requirement ?			
When does the existence of one or more requirements imply the need for additional requirements ?			

The operational environment within which the software will operate should be accurately described. All variables in the physical environment that the software must monitor and control should be properly specified. Has an independent analysis of the algorithms specified in the SRS been done to verify the correctness of those algorithms? [17]

2.4. CLARITY

Clarity requires that the contents of the SRS be clear and understandable. [17] The SRS should differentiate between requirements placed on the software and other supplementary information, such as design constraints, hardware platforms, and coding standards. A precise

definition of each technical term should exist, either in the SRS or in a separate dictionary or glossary. Each requirement should be uniquely and completely defined in a single location in the SRS.

Checklist	Yes	No	N/A
Does the Software Requirements Specification differentiate between requirements placed on the software and other supplementary information? Such as design constraints, hardware platforms, and coding standards.			
Is the functional requirements portion of the Software Requirements Specification organized in such a way that the requirements for each operating mode are grouped together?			
Is each requirement uniquely and completely defined in a single location in the Software Requirements Specification?			
Does the Software Requirements Specification contain a precise definition of each technical term and mnemonic that occurs in the Software Requirements Specification? - Reference to a dictionary or glossary containing the required definitions will satisfy the intent of this question.			
Does the Software Requirements Specification conform to standards imposed by the reactor vendor or software developer?			
Does the Software Requirements Specification distinguish between requirements and design constraints?			
Is there written justification for each design and each implementation constraint contained in the Software Requirements Specification? - Factors which limit the designer's options include, but are not limited to, regulatory and other legal policies; hardware limitations; interfaces to other applications; audit functions; use of specific operating systems, compilers, languages, and database management systems; use of specific communications protocols; critical safety considerations; and critical security considerations.			

2.5. TRACEABILITY

An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. Backward traceability and forward traceability are recommended.

Traceability requires that a two-way trace exist between each requirement in the SRS, and the safety system requirements and design. [17]

Checklist	Yes	No	N/A
Can each requirement be traced backward to specific elements in the System Design Description or Safety Analysis Report? - The preferred technique is a requirements tracing matrix which identifies each requirement, the system component implementing the requirement, the system design element which generated the requirement, and the test used to confirm that the requirement has been met.			
Can each requirement be traced forward to specific tests or validation criteria which will be used to confirm that the requirement has been met?			
Can each requirement be traced forward to specific design elements?			

There should be a two-way trace between each requirement in the SRS and the software design, as well as a forward trace from each requirement in the SRS to the specific inspections, analyses, or tests used to confirm that the requirement has been met. (Test matrix)

2.6. UNAMBIGUITY

An SRS is unambiguous if and only if, every requirement stated therein has only one interpretation. Unambiguity requires that each requirement, and all requirements taken together, have one and only one interpretation. [17]

Checklist	Yes	No	N/A
Can every requirement be interpreted in one and only one way?			

2.7. VERIFIABILITY

An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite process with which a person or machine can check that the software product meets the requirement. Verifiability requires that it be possible to construct a specific analysis, review, or test to determine whether each requirement has been met. Are all requirements listed to be verified by one or more of the following: analysis, peers, and inspection test?

3. ATTRIBUTES OF A COST EFFECTIVE REQUIREMENTS SPECIFICATION

The attributes of a cost effective requirements specification have been determined based on engineering judgment. Among the most important attributes are the following:

- Clarity;
- Unambiguity;
- Completeness;

- Correctness;
- Understandability (this attribute is not described in above section);
- Consistency;
- Robustness; and
- Safety.

These eight attributes are so important that the verification and validation of requirements against these criteria must be carried out, to the degree possible and as quantitatively as possible.

APPENDIX C

Checklist for the use of pre-qualified digital platforms for safety and non-safety applications in NPPs

The guideline on the use of pre-qualified digital platforms for nuclear power plant applications in EPRI 1001045 [61] established a checklist for use in applying the guidance given in that report. It is important that this or any other checklist be used by qualified personnel who are knowledgeable in digital systems, in conjunction with the pertinent regulations, standards, and industry guidelines of their country. This checklist is given below.

Checklist for the Use of Pre-Qualified Digital Platforms for Safety and Non-Safety Applications in Nuclear Power Plants			
1. Modification Process			
1.1 Project Definition			
	1.1.1 Establish safety significance and complexity of the application		
	1.1.2 Review platform qualification report and any Safety Evaluation Reports (SERs) to determine extent of generic qualification activities and any open issues that must be addressed		
	1.1.3 Determine whether the planned application is covered by the generic qualification		
	1.1.4 Determine what application software will have to be developed and the approach that will be used for V&V and configuration management (e.g., programmatic requirements)		
	1.1.5 Determine whether qualified configuration tools, or strategies for use of non-qualified tools, have been covered by the generic qualification		
	1.1.6 Determine the impact of the change on the human-system interface		
	1.1.7 Determine licensing strategy		
	1.1.8 Determine impact on integration plan and long term strategy		
1.2 Detailed Requirements			
	1.2.1 Identify requirements for the platform and the application		
	1.2.2 Ensure that the requirements document reflects any limits of the generic qualification program that lead to specific requirements on the modification.		
1.3 Design and Implementation			
1.3.1 Comparing the Application to the Bounds of the Generic Qualification			
	1.3.1.1 Compare the plant-specific application and environment to the bounds of the generic qualification program		
		1.3.1.1.1 Check the qualification status of the equipment to be provided	

		1.3.1.1.2 Compare the planned hardware configuration to the generic qualification	
		1.3.1.1.3 Compare the application physical environment to the qualification envelope	
		1.3.1.1.4 Check that the planned functionality fits within the bounds of the qualification	
1.3.2 Extending the Bounds of the Qualification When Required			
	1.3.2.1 Determine what must be done to extend the qualification if required to meet specific application requirements or constraints		

1.3.3 Interfaces			
	1.3.3.1 Examine interfaces, including communication interfaces among qualified devices, to determine whether the generic qualification provides all the needed information or if additional work is required		
1.3.4 Application Software			
	1.3.4.1 Determine how the required application software will be developed, and ensure it receives adequate V&V and configuration management		
		1.3.4.1.1 Determine what applications will be involved in this modification, and what software will have to be developed to implement those applications	
		1.3.4.1.2 Determine how these applications will be developed, by whom, and under what QA program and procedures	
1.3.5 Use of Configuration Tools			
	1.3.5.1 Ensure that the planned use of configuration tools is qualified		
		1.3.5.1.1 Are configuration tools required?	
		1.3.5.1.2 Was the specific tool you intend to use qualified, or was a method for using the tool qualified, as part of the generic platform qualification?	
		1.3.5.1.3 If the tool you intend to use was not qualified, and no justification was developed for use of the tool with the qualified platform, have you scoped out and accounted for the cost and time to develop such a justification?	
1.3.6 Other Design and Implementation Issues			
	1.3.6.1 Ensure that operational and performance issues associated with digital systems are adequately addressed in the modification		
		1.3.6.1.1 Consider operation of the existing system and ensure operational features are transferred to the new implementation as appropriate	
		1.3.6.1.2 Determine which system performance requirements depend on the digital system and thus place requirements on the digital platform	

		1.3.6.1.3 Confirm that the required performance falls within the envelope of the generic qualification	
		1.3.6.1.4 Ensure digital system performance issues are addressed in the design	
	1.3.6.2 Determine the strategy for periodic testing of the digital equipment		
	1.3.6.3 Address human factors considerations in the design and configuration of the digital system		
1.4 Testing and Installation			
	1.4.1 Determine what tests should be run during Factory Acceptance Testing (FAT) and Site Acceptance Testing (SAT), taking advantage of what was done in generic qualification.		
2. Commercial Grade Dedication			
	2.1 If the digital device is to be procured by the utility as a commercial grade item, make use of the generic qualification results to simplify the process of evaluation and acceptance/dedication for the plant-specific application		
3. Failure/Hazards Analysis			
	3.1 Check the qualification report to ensure that failure modes of the qualified platform have been addressed by the generic qualification activities, along with effects of the environment on the platform		
4. Licensing			
	4.1 Ensure that the issue of software quality has been adequately addressed and documented, sufficient to meet NRC requirements and expectations per Chapter 7 of the Standard Review Plan		
	4.2 Determine whether changes are required to the plant Technical Specifications		
	4.3 Perform the 10 CFR 50.59 evaluation if required, making use of pertinent information from the generic qualification program		
	4.4 Ensure that the potential for software common mode failure has been adequately addressed and, where necessary, perform a diversity & defense-in-depth analysis		
	4.5 Ensure that human factors issues have been adequately addressed		
	4.6 Ensure that the required documentation and records are developed to support licensing		

APPENDIX D

The use of diversity as a mean to reach reliability

The structure to be chosen for an I&C system depends basically upon the structure of the process system to be controlled and the degree of dependability required. Redundancy, diversity, defense-in-depth as the basic means for building high integrity systems are applied both for hardwired and software based I&C systems but in the latter case a set of additional constraints have to be observed which are connected to the characteristic of programmable processors. In this context common cause failures (CCF), which may be introduced by software faults, are of specific concern. This section discusses in some detail diversity as an approach to abate CCFs and thereby to make the software based I&C more reliable.

1. INTRODUCTION

Difficulty in reliability assessment of diverse systems derives from the fact that one cannot claim independence of failures in diverse versions. Independence of failures means that the probability of the versions failing together on a demand is the product of their individual probabilities. Versions that are developed independently are said to hold best chances to achieve effective diversity.

Diversity may designate several concepts: process diversity, product diversity, fault diversity and failure diversity. Software developments produce process diversity, which causes the version products to be visibly different in their structure and internal operation. The versions will hopefully be less likely to contain identical faults and the probability that the versions will fail in the same way on the same demands is reduced.

It is a problem to determine how much the reliability of a diverse software system is increased compared to the reliability of the individual software. For redundant hardware systems, one assumes that the reliability is the product of the reliability for the individual channels. The basic assumption for this is that the failures occur randomly and independently in the different channels. This assumption may be valid for hardware errors, but the situation is much more complex where software failures are concerned. In this case there are two sources of randomness. One is the 'random' introduction of software faults during the software development. Another is the random distribution of input data, which are common to all the diverse versions.

Theoretical and empirical studies have showed that the assumption of independence is not generally valid, even if one assume that the faults were made completely randomly and independently in diverse versions. This is because of variance in the intensity of the distribution of the common input data, or the possibility of failure masking. In addition there is the possibility of common cause failures, either because of defects in the common specification or because of some common typical slips of the mind.

2. A MODEL OF DIVERSE SYSTEMS

A model of the influence of diversity on two versions of the same system can be constructed as indicated below. The probability of failure *P(AB)* of two version system is

$$P(AB) = P(B) \bullet P(A|B), \qquad (1)$$

where both versions fail simultaneously, and $P(A)$ and $P(B)$ are the failure probabilities of versions A and B, and $P(A|B)$ is the conditional probability of A failing when B has failed.

If complete independence of the failure behavior of the versions can be assumed:

$$P(A|B) = P(A) \qquad (2)$$

and the probability of the failure for the system:

$$P(AB) = P(A) \bullet P(B) \qquad (3)$$

A failure probability in the order of 10^{-3} can be achieved for software based technology with quite reasonable effort. If also the failure independence between two such diverse versions could be achieved, the system failure probability would be 10^{-6}, which would be sufficient for the most safety-critical applications in nuclear power plants.

Unfortunately, the assumption of the total independence of the failure behavior of diverse redundant channels is seldom fully justified. In the worst case the failure behavior can be totally dependent that is the conditional probability is

$$P(A|B) = 1 \qquad (4)$$

and the failure probability of the system

$$P(AB) = P(B) \qquad (5)$$

In this case no reduction in the failure probability of the system would be achieved by using redundancy. It may however, be a rather hard assumption to use that if one redundant channel will fail then the other will fail with probability one. A more deliberate investigation may suggest that certain failure modes are not dependent and the conditional probability should be something between 1 and 0. This means that there still may be a benefit of using redundant systems with the same software. To model the probability that the other redundant channel will fail when the first has failed a so-called β-factor has been defined. By connecting the β-factors to specific failure mechanisms and using probability calculus it is possible to make an assessment of the influence on $P(A|B)$ as above.

In many cases the objective is not to achieve probability in order of 10^{-6}, but of probability of about 10^{-4}. Single high quality software have difficulties to achieve even the latter probability, but using two diverse software components in a redundant configuration can often provide a way of achieving the required probability at reasonable cost.

3. DIFFERENT KINDS OF DIVERSITY

The diversity in the system can be realized by applying a proper combination of the three main design diversity principles:

1. hardware diversity
2. functional diversity
3. software diversity.

Some low degree of diversity can even be achieved by using independent and different sensors and asynchronous operation in different trains of a single variant. Due to the small differences in sensor signals the input sequences in sensor signals the input sequences to the redundant trains slightly deviate from each other which may be able to prevent the common residual faults to cause simultaneous failing of all trains.

The reliability gain that software diversity produces is difficult to assess. Diverse versions are not assumed to fail independently, and all other techniques for assessing reliability are no less problematic for multiple versions than for ordinary software. An important question is how best to achieve effective diversity.

3.1. HARDWARE DIVERSITY

There exists several principally different ways to apply hardware diversity to improve the fault-tolerance of a programmable system. Either one can use different hardware in the individual channels of a programmable system or one can equip the programmable system with an analog back up system.

One way to try to achieve fault tolerance in a redundant computer based system is to have computers from different manufacturers in the redundant channels of the system. Each of them is running its own object code translated from the same high level source code. Different computers can have different processors, operating systems, compilers and other software tools. Therefore they are thought to execute the same software differently enough to achieve resistance to CIF's. There are, however, some doubts about the efficiency of this approach [C-1]. For example, the operating system and other software tools can have the same origin, same hardware component is used by different manufacturers etc.

A somewhat more efficient way to achieve fault-tolerance through hardware diversity is to use different programmable system platforms from different manufacturers. The secondary system could possible be a simplified version implementing only the most critical tasks of the primary system. The selection of another system has the advantage of being of the same modern technology as the primary system, which may ease the maintenance of the system. On the other hand, the difference between the two systems is an implementation of random diversity, and its may be difficult to prove the degree of independence achieved between the systems.

The extreme alternative to protect against common cause failure propensity of software based systems would be the use of a conventional hardwired at least the most important

functions of the system. Since it is not easy to achieve the same functionality with the analog technology than with programmable digital technology, the use of analog back up usually contains also some functional diversity. In any case this probably is not a cost effective way. If the cause of the replacement of an existing analog system in an old nuclear power plant has been the obsoleteness of the old system and difficulties in its maintenance, much of the advantages in replacing it with a modern programmable system are lost, if one still has to construct also a new analog back up system.

3.2. FUNCTIONAL DIVERSITY

Functional diversity means that the system in concern contains at least two parallel redundant channels that perform completely different functions to satisfy the same high level safety and reliability goal. The crucial point here is that the requirement specifications of functional channels is different. In the case of software, functional diversity means that the behavioral requirements for the software are different. For example, one program may check to see whether two numbers are equal and another, functionally diverse, program might select the larger of two numbers.

At [D-1] is presented a model of functional diversity. It is showed that claims for independence between functionally diverse systems seem rather unrealistic. Instead, it seems likely that functionally diverse systems will exhibit positively correlated failures, and thus will be less reliable than an assumption of independence would suggest. The result does not, of course, suggest that functional diversity is not worthwhile; instead, it places upon the evaluator of such a system the responsibility to estimate the degree of dependence so as to evaluate the reliability of the system.

3.3. SOFTWARE DIVERSITY

The means to achieve software fault tolerance is the software (design) diversity. This means that functionally equivalent but independently developed software versions are applied to produce the output of the system. Different techniques, recovery blocks and N-variant techniques being the most common can apply software diversity.

Eckhardt & Lee [D-2] dealt with diversity in multi-version software. The main idea is that the demands placed upon systems by their environment can vary in ‘difficulty’, and that this variation induces dependence upon failure processes of different ‘diverse’ versions. We can say that observing version A to fail causes us to believe that the demand was probably a ‘difficult’ one, and thus increases the chance that B will fail. The greater the variation in ‘difficulty’, the greater the dependence in their future behavior. They will fail independently only in the case where there is no variation of difficulty at all [D-1].

Even a single version software environment can take advantage of fault-tolerant mechanisms. These include naturally fault detection and recovery in the system. In addition to these there are some basic fault-tolerant features for a single version system like program modularity, atomicity of actions and exception handling. In fact, diversity can be included in this approach as well. To be able to detect faults, which in practice often means verifying the execution results of a part of a program, a different approach for implementing the part of the program is called for. The problem that is solved by the piece of code, has to be solved in an other way for fault detection.

The use of common specification is a source of common cause errors. The fact that the human mind sometimes has the propensity to make certain types of errors may also cause a common fault in diverse programs. The basic idea of diversity is extended to incorporate a notion of 'forced diversity'. Development teams are forced to use, for example, different programming languages, different testing regimes, etc. The kind of faults that would enter version A would not be present in version B, and vice versa. In the terminology of 'difficulty' [D-2], those inputs that were different for A would tend not to be difficult for B, and vice versa.

One application where the use of diverse programs is particularly useful is as 'oracles' in testing. Back-to-back testing of a program against diverse programs with a large number of automatically generated data can give a high confidence in the fault freeness of the program.

Another approach to fault-tolerance is the Recovery Block approach. As the name implies, the basic goal is to detect a software fault in a module (or program), recover the machine state at the time the faulty module was entered, and execute another module that performs the same function as the faulty module. As in the N-version programming approach, a number of independently designed modules that perform the same function must be developed. It is usually assumed that the probability of common design faults, or coincident errors, among the different versions is negligibly small.

The versions are ranked based on some quality criteria. When a given task is requested, only the "best" module is executed. If an acceptance test detects an erroneous output, then the "next best" one is executed, etc., until an acceptable output is obtained. If all versions are deemed faulty, an error is posted.

4. A DISCUSSION OF SOFTWARE DIVERSITY

4.1. THE PROBLEM OF SOFTWARE DIVERSITY

To achieve a complete diversity it would be necessary to ensure that diversity is applied consistently within the whole development process, from team selection, to using different development environments, different tools and languages at every level of specification, design and coding, implementing each function with different algorithms, applying different V&V methods, etc. Even then there is a need for some commonality, such as for example that the two diverse versions should be used for the same task, at the same plant and by the same individuals. Diversity also carries additional cost through duplication of activities, added co-ordination effort, need for staff with specific skills.

A typical diverse system includes three or four redundant units, and depending on the selected approach, there are some additional units like a voter or an acceptance test unit and a selector. Also supporting functions are needed: inter-version communication, version synchronization and enforcement of timing constraints, local supervision of each version, the global executive and decision function for version error recovery, and a user interface for observation, debugging, injection of stimuli and data collection during N-version execution of application programs.

From the software point of view, the state space of the software increases along units and functions. The increase is naturally affected by the structure of the system and the software. It can be estimated that even if testing effort is increased, the test coverage of the whole system will in most cases be clearly worse than that of single version software. At the same time the development cost of N-version software are multiple compared to a single version system. Therefore it may be justified to put a question about the necessity of redundant units: would it be easier to develop a single version system, equip it with fault-tolerant mechanisms and test it thoroughly?

Diversity also sets high requirements on software maintenance. In principle the problem of maintenance is equal to software development. In both cases there are N versions to deal with, and the original requirements of failure probability are still valid. Thus the same procedures that were applied to N-version software development and demonstration need to be used in software maintenance.

Operational experiences show that reliability of some fault tolerant systems is remarkably high, but do we have theoretical means to measure the degree of dependence between the units? In a use of diverse software it is difficult to keep cost at a reasonable level. The generation of N versions of a given program instead of a single one shows an immediate increase in the cost of software prior to the verification and validation phase. The cost/effort of diversity is not directly proportional to the number of software versions, but proportional to the amount of diversity. If the steps — program specification, coding, program test — is executed in parallel for N versions, the cost are N times the cost for the same steps of a single version. The costs of the steps — requirement specification, system specification, test specification, system test — may still be the same, but may on the other hand introduce unexpected dependence between the versions. Overall replacing costly verification and validation tools and operations may also reduce cost by a generic N-version environment in which the versions validate each other.

4.2. SOFTWARE DIVERSITY METRICS

It is usually difficult to measure the overall system reliability when redundancy and / or diversity play a role in system design. The problem has been tried to quantify by defining a β-factor — the probability that if a failure happens in one channel, the other channels are also failed because of common cause of the fault.

In the beginning of this Section, four metrics were defined: process diversity, product diversity, fault diversity and failure diversity. We can propose a metric which support qualitative analysis of N-version systems. A hypothesis is put forward saying that the term ’software diversity’ correlates with four following aspects [D-3]:

(1) Structural diversity is measured among software versions. It is further divided into basic metrics such as: deliverable source lines; non-commented source code lines; Halstead’s measures; decision count; and McCabe’s cyclomatic complexity.

(2) Fault diversity reflects the differences between faults found among the software versions.

(3) Tough-spot diversity measures the differences in fault-proneness among the elements of the software versions.

(4) Failure diversity specifies the differences in the failure behaviors among the software versions.

Structural diversity had a clear correlation with the total number of faults. Halstead's measures and McCabe's cyclomatic complexity showed the strongest correlation with the total number of faults. Other performance of metrics was found poor in the experiment.

Fault diversity metrics turned out to be rather useless in the context of reliability assessment. The fault diversity had the value of 90/92, i.e. only two of the faults were common cause faults. In the experiment, both faults ere caused by the specification. However, the measure is certainty application dependent. Thus the result of the study in itself is not applicable to other N-version software and for the assessment — if the measure is found useful — it needs to be extracted for each N-version software separately.

Tough-spot diversity measure could be utilized in the selection of optimal component configuration for the application. Analysis of the failures proved that tough-spot could actually be localized in the software. Therefore it is reasonable to include such components in the configuration that have different tough-spots. However, this is seldom possible in real world development projects.

Failure diversity has same comment than the recent diversity measure. Different combinations of components produce different failure probabilities.

5. EMPIRICAL RESULTS ON FAULT-TOLERANT SYSTEMS

5.1. ASSUMPTION OF INDEPENDENCE BETWEEN VERSIONS

The empirical results have proved that the theoretical value of system reliability under the assumption of independence of N-version software does not hold (e.g. [D-4]. Actually the differences between the assumed failure probability and the measured reliability were quite remarkable.

Experiments have also shown that it is dangerous to assume the β-factor to be a constant, even within one experiment. In the experiment by [D-4] different β-factors were measured in the 3- and 4-version systems configured out of total number of 27 versions. Thus the different versions with various failure behavior also produced different β-factors. The result indicates that experiences from other, similar systems should not be used as a strong evidence for the actual system, or at least, if they are used, the assumptions should be careful and conservative.

Empirical studies have largely been exploring the failure behavior of individual versions, but other components of fault-tolerant systems have mainly been neglected. For instance, the reliability associated to the voter is known to have significant differences [D-5], but no quantitative studies have been conducted on the subject. Yet the N-version programming fault and failure analysis by [D-5] show that decider failure has a strong impact on the overall reliability of the system. One reason to the avoidance of this issue in the studies might be the fact that the reliability of the supporting software is much dependent on system architecture and design.

The possibility of common cause failure in the voting logic has been stated as a problem in diverse programs. However, if diverse programs are implemented on channels, which are redundant to protect against consequences of hardware errors, we do not see that this will introduce any new complexity. One problem with voting is that two diverse programs may give diverging results, even if they are both correct, but we do not see this as a large problem in practical application, since both results by assumption are correct.

An example of problems in the voter is reported by and named as a Consistent Comparison Problem. The problem arises whenever a quantity used in a comparison is the product of inexact arithmetic (floating point values for instance). It may occur even when all software versions are correct. It results from rounding errors, not software faults, and so an N-version software built from "correctly" operating versions may have a nonzero probability of not being able to reach consensus.

5.2. THE IMPACT OF SPECIFICATION TO RELATED FAULTS

Since faults in the design and coding stages tended to be detected earlier, quite a high proportion of faults in the final program versions stemmed from the specification. In PODS – A Project on Diverse Software [D-6], all the errors were caused by the specification. In these and several other experiments quite a high proportion of the faults were also similar, i.e. related faults.

Diverse specifications could possibly be a solution for these specification-originated faults. The approach was tested in PODS and it was found out that the use of relatively formal notions were effective in reducing specification-related faults caused by incompleteness and ambiguity. The use of diverse specifications may, however, have drawbacks: the performance in minimizing common design faults is uncertain, and there is a risk that the specifications will not be equivalent. In practice, only a single good specification would be used unless it could be shown that diverse specifications were mathematically equivalent.

Theoretical analysis by [D-7] gave a somewhat different interpretation to coincident failures. They suggested that certain inputs are more difficult than others, i.e. the failure probability with them is high and there is a tendency for many programs to fail. Thus the probability distribution over the different inputs which is created by operational environment of the system, is — according to them — in turn connected to a probability distribution on the probabilities of failure.

To put the previous interpretation in a more simple way, designers tend to make similar errors. For instance, in this context coincident errors are not necessarily originated from the specification, but also from designers. A typical example could be a 'specification' for using a proper algorithm. One of two different algorithms, X and Y need to be selected, which depends on an input signal from sensor A. If the input value from A is below some limit, algorithm X should be selected, and if the input value is above the limit, then Y should be chosen. But what happens when the input value is exactly equal to the limit value? Totally regardless of the specification, many of us would surely propose the value 'limit' from sensor A as a candidate for difficult inputs.

So it is not only the specification that needs to be taken into consideration. As well the degree of difficulty distribution' of the designers should be considered. Using a different development process could alter this distribution for instance. If the 'degree of difficulty distribution' could be manipulated in an optimal manner, it might even lead to a negative correlation between failures (coincident faults would then be even more rare than the theory of fault independence suggests).

6. IMPLICATIONS FOR SOFTWARE DIVERSITY

One of the cornerstones of fault-tolerant systems is in the diversity of individual versions. Therefore reliability assessment also concentrates much on diversity. Qualitative assessment naturally studies software development issues, such as the structure of the software process and the results of software testing, and emphasizes elements that have a direct impact on software diversity. From earlier experience some useful parameters can be deduced like the influence of methods, tools or languages on diversity. Also other type of metrics have been proposed quite recently. These metrics specify software diversity by defining new terms: structural diversity, fault diversity, tough-spot diversity and failure diversity. Unfortunately, on the bases of a case study, the terms do not give much support for the assessment. Only structural diversity seems useful, as the two metrics included in the structural diversity (software complexity and size of the vocabulary) show a strong positive correlation with the number of defects. However, it should be noticed that the correlation value of the experiment cannot be used as a common tool for predicting the number of faults in software. The correlation factor must be defined for each application separately.

The major concern of the N-version systems is indeed common faults. To minimize the probability of related faults there are two means: firstly to increase the quality of software by reducing the possibility of mistakes and inconsistencies, and secondly to eliminate the most potential causes of related faults. The first one can be tackled by applying process improvement practices, which are generally used in the production of high dependability software. The second one is more difficult, but may be partially solved by means of diverse formal specifications—as specification is considered one of the major causes for related faults—and diverse production process. Even in the case of a single specification formalism may be used to raise its quality and remove inaccuracies and ambiguities, which are potential sources of human mistakes. A different type of an approach for avoiding common faults is functional diversity, where the whole function is implemented in a different way by measuring a different magnitude in the target process, e.g. temperature in one unit and pressure in another. The approach is especially used with programmable automation systems, in which diversity may be hard to implement by other means. The weakness in the approach is that the versions can be functionally too different, causing problems for the voter in finding an agreement among the unit responses. Additional difficulties may also arise in proofing that the specifications are equivalent, which may turn out a demanding task.

REFERENCES TO APPENDIX D

[D-1] Littlewood, B., Popov, P., Strigini, L. A note on reliability estimation of functionally diverse systems. Reliability Engineering and System Safety, 66, 1999. pp. 93–95.

[D-2] Eckhardt, D.E. & Lee, L.D. A theoretical basis for the analysis of multiversion software subject to coincident errors. IEEE Transactions on Software Engineering. SE-11, (12), 1985, pp. 1511–1517.

[D-3] Lyu, M. & Avizienis, A. Assuring design diversity in N-uersion software: a design paradigm for N-uersion programming. In J.F. Mayer and R.D. Schlichting (eds.), Dependable Computing for Critical Applications 2. Springer-Verlag, New York. 1992.

[D-4] Knight, J.C. & Leveson, N.G. An experimental eualuation of the assumption of independence in multi-uersion programming. IEEE Transactions on Software Engineering. SE-12, 1 (Jan.). 1986. pp. 96–109.

[D-5] Laprie, J-C., et al. Architectural issues in software fault tolerance. In Luy, M.R. (ed.). Software Fault IDlerance. John Wiley and Sons. 1995.

[D-6] Bishop, P., Esp, D., Barnes, M., Humphreys, P., Dahll, G. & Lahti, J. PODS—a project of diverse software. IEEE Transactions on Software Engineering, vol.SE-12, September 1986.

[D-7] Littlewood, B. & Miller, D.R. Conceptual modelling of coincident failures in multi-uersion software. IEEE Trans on Software Engineering, vol. SE-15, no. 12, 1989. pp. 1596–1614.

APPENDIX E

Methods used in the confidence building process

Building confidence that the software based I&C system is fit for its purpose should be done during its whole lifecycle. The confidence building should also be based on the largest amount of available information. Some of this information can be collected just simply documenting for instance used design methods, data sets, test results, deviance reports, etc. Other parts have however to rely on specific evaluation efforts carried out as a part of the V&V process. It is very important that this collection of information is accurately defined in the V&V plan, because it may be impossible to regenerate information from earlier design activities, which have been lost. Finally the judgment of the safety of the software based I&C system should be based on this collected evidence.

The methods laid out below are examples of evaluation methods that can provide different pieces of evidence during the design and implementation of software based I&C systems. In applying these methods it is however important to note that the final decision on the methods to use depends on the intended use of the software module and the available information regarding for instance source code, test data, specifications, etc.

1. HAZARD AND RISK ANALYSIS METHODS

The objective of the hazard and risk analysis methods is to evaluate a system with respect to potential hazards it may cause. A risk analysis (PSA/PRA) also includes the probabilities and costs of hazards.

A Preliminary Hazard Analysis (PHA) provides an initial overall view of risks. This provides the initial framework for a master listing of hazards and associated risks that require tracking and resolution during the course of the system design and development. The PHA effort should ideally be started during the concept exploration phase or earliest life-cycle phase of the system development. However, in the present case this analysis was made in retrospective.

Preliminary hazard analysis of the entire target system is performed top-down to identify hazards and hazardous conditions. Its goal is to identify all credible hazards up front. For each identified hazard, the PHA identifies the hazard causes and candidate control methods. These hazards and hazards causes are mapped to system functions and their failure modes. This should then be followed up by a more detailed System Hazard Analysis (SHA).

Some particular methods in hazard and risk analysis are described below. These are the commonly used methods, but there are also various other methods.

1.1. HAZARD AND OPERABILITY ANALYSIS

Hazard and operability (HAZOP) analysis looks at possible disturbances in a system, rather than failures for each component. Disturbances involve a variation in some process variable such as “too high speed”, “wrong direction”. Aspects like causes, consequences, detectability, correctability, safety barriers etc. are determined.

In general terms a HAZOP analysis is performed as a kind of 'brain storming' activity: An analysis team is gathered, consisting of different experts, and headed by a HAZOP leader. The team leader prepares in advance a so-called 'HAZOP form'. The columns in this form identify a set of 'objects', and a set of 'guide words' about these objects. At the analysis meeting the HAZOP leader put forward each object and corresponding guideword, and asks the team to openly discuss the objects on the basis of the guidewords. A team secretary is, during this discussion, making a HAZOP log by filling out the pre-made HAZOP form.

In principle there are no difference in making a HAZOP analysis on a programmable system and any other system. However, a HAZOP analysis is based on a schema containing different guidewords for the analysis, and this need to be adapted to the actual functioning of the system and the potential disturbances. For a programmable system they will most probably be different than for a mechanical system.

1.2. FAULT TREE ANALYSIS

Fault Tree Analysis (FTA) is a method, which has been widely used, for safety analysis for many years. It considers each accident, which can occur, and searches for possible causes. The causes are recorded on a tree structured diagram, with AND symbols to indicate where several conditions must occur together with an initiating event, in order for an accident to occur, and OR symbols are used to indicate alternative causes. One main target of FTA is to identify potential common cause failures in diverse or redundant systems. A characteristic of a software fault, in distinction from random hardware faults, is that it occurs in all instances of the same software module used in redundant channels. Software failures therefore constitute a potential risk for common cause failures in such systems, and a task of FTA is to reveal where such failures can occur.

Leveson and Harvey [E-1] suggested another application of FTA on software. Here FTA is applied directly on the code listing, and is more closely related to code verification than actually to PSA. (For more details about this method, see [E-2], pp. 497 - 507)]. A rather different attempt has been to apply FTA, not on the product, but on the development process of safety related software [E-3]. The goal of this application is to find critical events concerning fault introduction, and to relate these events to the development process.

1.3. FAILURE MODE, EFFECTS AND CRITICALITY ANALYSIS

Failure Mode Effect and Criticality Analysis (FMECA) is an analysis, which concentrates on the potential failures of individual components. The basis for an FMECA is a functional description of the system to be analyzed in terms of its components. For each of the components in the system, the aim is to identify all possible or potential modes of failure. Then, for each failure mode one makes an evaluation with respect to a set of points:

- The *failure mode*, i.e. how the failure manifest itself.
- The *failure cause*. This includes both immediate causes and more basic causes, as e.g. design errors.
- The *failure mechanism*, i.e. the mechanism which leads from the cause to the failure.

- The *failure effect.* One can here distinguish between *local effect*, which is the effect on the component in question and its immediate surroundings (e.g. failure mode: pump stop, effect: no flow), and the *end effects*, which are the effects the failure may have at the highest system level, i.e. on the plant and its environment. In the latter case one could utilize fault tree analysis.
- *Failure criticality, i.*e. the consequences the failure may have on the safety at the plant or potential harm in the environment. One can also here distinguish between immediate consequences (e.g. radioactive release after a tube rupture), and more indirect consequences which are consequences of the end effects.
- *Failure detection*, i.e. the way the failure can be detected and the likelihood that it will be detected.
- *Failure probability.* This can be stated in qualitative terms (e.g. high, medium, low), or quantitative as probability of occurrence per time unit or per demand.

This type of analysis has traditionally been applied to hardware components, but in recent years it also have been applied to software systems. At the Halden Project we have investigated two different approaches to this.

One approach has been to apply FMECA to configurable software systems, i.e. systems that are built up by standard software components in the same way as a hardware system is built up by standard hardware components. Such systems are often used in the control of NPPs, also in safety related applications. The method has been to apply FMECA on the standard software modules as one would apply them to hardware modules, although there are considerable differences in the detailed analysis. Hardware failures occur randomly during operation, whereas the components considered here are realized in software. Only potential design (programming) errors which results in inherent faults in the programs are therefore taken into account in this connection. However, such faults can result in stochastic behavior of a program during execution, due to randomness in the input data.

The following contains some general considerations on the points mentioned above with respect to software modules.

- Failure Modes.

From the specification it should be possible to identify the correct output from a module in all situations, and further to identify all potentially incorrect outputs. Most failure modes can be classified into some main types:

- incorrect response (output)
- no response when it should occur
- correct response, but outside required time limits
- correct response, but undesired side effects

Depending on the functioning of the particular module to be analyzed, these general failure mode types could be divided into more specific failure modes, e.g.:

- The result of a computation can be completely wrong, or only slightly inaccurate.
- The failure can occur in all executions of the module, or only in some cases, or perhaps only in very special cases.
- The algorithm used in the module is not applicable in the particular case.

- Failure Causes.

When software failures are concerned, there are two main classes of causes, inherent faults in the software or incorrect use of existing software modules. The more basic causes of inherent faults can be defects in the specification, programming errors, incorrect corrections, incorrect modifications in new releases etc. Even if one has limited information about the software modules, one should utilize all available information to reason about likely failure causes.

- Failure Mechanism.

The failure mechanism concerning software failures is how a programming error can lead to a software fault and how a software fault can lead to an execution failure. For an inherent fault the failure typically occurs when an actual input hits the failure domain of the faulty program. The most common faults to look for are those, which cause wrong results. The most obvious and common failure mechanism is logical or structural faults in the program, which makes clearly incorrect results. Other failure mechanisms may be that an approximation algorithm may be too inaccurate for a certain range of input data, or that singularities in the algorithm may occur. A particularly important failure mechanism is related to timing problems.

- Failure Effects

It is in general only possible to identify local effects based on the module itself. The end effects must be found in context with the complete program. If for example a failure mode is ‘no response’, the immediate effect is that the module cannot pass control to the next module. This failure mode will then have an end effect, which might be easy to see from a graphical representation of the complete program.

- Failure Detection

This is a point where software modules have some clear advantages compared to their conventional counterparts. Software faults can be detected and removed during V&V and testing, and therefore make no harm. Inherent faults that remains can lead to failures that are detected during execution. As the detection of a failure is important to safety, it is possible to design failure detection facilities into the software system. These are mainly designed to trap hardware errors, but facilities also exist which detect failures caused by software faults. In the analysis for failure detection, one should look for possibilities and likelihood to detect them, both before implementation and during execution. Concerning the latter, the possibility for fault tolerance is also interesting.

- Failure Criticality.

Software does not harm anyone, so any immediate safety consequences of failures in the modules are not expected. The safety consequences should rather be found by an analysis, e.g. fault tree analysis, of the application program and its influence on the environment.

- Failure Probability.

The other way to apply FMECA was made in the System Hazard Analysis of a train communication system [E-4]. In this case the analysis was based, not on the components of the actual system, but rather on the elements of a functional specification of the system. This specification was made in the form of a set of MSC (Message Sequence Chart) diagrams. An MSC consists of a set of processes, and each process can perform actions and send and receive messages to/from other processes. The 'components' in this analysis were the actions and messages. The method was to ask the following questions for each action and message, based on the FMECA framework:

- What can go wrong (*failure mode*) ?
- How can this occur (*failure cause/mechanism*) ?
- Which consequences will this have on the further actions and messages (*failure effect*) ?
- Can any of these consequences potentially lead to any critical event previously identified in a fault tree analysis (*failure criticality*)?
- Are there any internal failure detection mechanisms in the system which can detect this failure, and which can prevent, or reduce the possibility, that this failure will lead to a critical consequence (*failure detection)*?
- If a safety critical failure could be caused by a software fault, which test procedure should be followed to detect this potential fault (*fault detection*) ?

2. SOFTWARE ANALYSIS METHODS

There are various analysis methods that can be used for software modules. It is perhaps necessary to point out that these methods always should be seen as complementary to the standard testing carried out to ensure proper functionality of a software module.

Static analysis is defined as evaluation of a computer program without executing it. The main objectives are to check that the final program conforms with the specification or design documents, and to reveal defects in the program concerning as well direct faults as violation of coding standards.

It is good practice to always perform some form of static analysis. The amount of effort put into the static analysis should depend upon the dependability requirements to the system. The most natural is manual inspection of all documentation, from specifications to code listings. A cost effective inspection could be achieved by using some kind of structured inspection, such as e.g. Fagan inspection method, structured walkthrough etc.

In addition to static analysis it is obviously always necessary to run the code, i.e. to perform a dynamic analysis. There have been various methods developed to increase the efficiency of the testing, for example through the establishment of certain domains in the input space, which are equivalent in the sense that one could conclude that the software will perform correctly also in the other domains if it has been shown to perform correctly in one of the domains. There have also been methods developed to do a symbolic execution of the code to detect discrepancies in variable definition and use. Finally dynamic testing using suitable simulated plant functions also is an important part of ensuring confidence in the system.

2.1. STATIC ANALYSIS

Rigorous static verification involving a high degree of formality to show conformance between source code and Software Requirements is usually required by Licensing Authorities. An approach is applied that maximizes confidence in the correctness of the code in a cost effective manner. The approach involves a tool-supported verification of the semantics of the code against a formal representation of the required operation of the code. The verification not only indicates that each source module implements its module design specification, but also gives a high degree of confidence that the operation of each software subsystem fully meets its requirements.

In the following subSections is as an example described the static analysis process applied for the NPP Temelin Safety Critical SW independent assessment.

2.1.1. Analysis overview

The static analysis process can be drawn in the following way as indicated in Figure E-1:

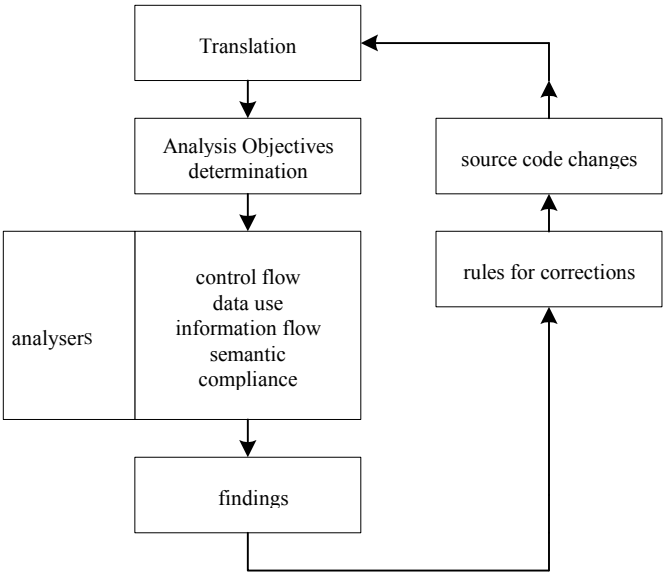


Figure E-1. Static analysis process.

To make a static analysis it is necessary to apply supporting tools. There are various commercially available analysis tools on the marked. As there is often large volumes of software to be analyzed, it is necessary to divide the task between many analysts to make a complete analysis in a reasonable period of time.

The main stages in the analysis process are the following:

- Preparations
- Goal setting
- Syntax analysis
- Semantic analysis
- Optional Review
- Compliance Analysis
- Review
- Evaluation of findings.

At the end of significant activities, the analysis results are reviewed to check the standard of work. Details of each analysis should be recorded using the documents specifically developed for the assessment.

2.1.2. Goal setting

The goal setting is the process by which correctness properties are identified and the corresponding analysis objectives are captured. It is necessary to perform a subsystem — wide review of the software to identify correctness properties and corresponding analysis objectives prior to the start of the analysis.

These fall into two categories:

(1) Identification of functional integrity

(2) Identify hidden assumptions, i.e. to identify the analysis objectives for each procedure in the module or subsystem.

2.1.3. Syntax analysis

Syntax analysis is a collective term used to encompass Control Flow, Data Use and Information Flow analysis

Control Flow analyzer examines the structure of a procedure to identify all entry and exit points. It also identifies all loops with their entry and exit points. Control flow analysis also reveals more serious errors within the source code such as unreachable code and dynamic halts.

Data Use analyzer checks how data (variables and formal parameters) is used within the procedure and from this one can check (for example that all outputs are written on each path through the procedure).

Information Flow analyzer identifies all information upon which each output depends and provide an initial check that the procedure outputs are depend upon the correct inputs. The dependencies are specified in a list that the analyzer compares these against the calculated dependencies. In order for the Information Flow analysis to be meaningful, these lists must be written to reflect the intended or specified information flow properties. The aim of Data Use and Information Flow analysis is to choose access modes that reflect the true data use properties, not to get empty error sets.

When completing the Syntax Analysis report care must be taken that all the sections of the report are completed correctly. The Syntax Analysis report should provide a complete description of the Syntax Analysis activity in terms of discovered anomalies and modifications that are made and any.

2.1.4. Semantic analysis

The principal aim is to show conformance of the source code to its design documentation. The analysis identifies each path through the procedure by a condition on the input value. For each path, the analyst reveals the actions taken when the path condition is met. The analyst should then compare this against the design documentation to decide if the procedure will behave correctly.

The analysis has the following objectives:

- checking that each source module implements its module design specification
- correctness of real arithmetic will be informally considered
- checking that variables are initialized before they are used
- consider the possibility of overflow during expression evaluation without making implementation dependent assumptions about order of evaluation
- possible ambiguities are flagged and explained
- only legitimate values are written to variables
- confirm that base pointers are correctly set for all accesses to based variables
- aliasing of parameters does not occur
- addressability of objects is investigated where necessary
- all preconditions for changed to the IL model should be formalized in IL wherever possible
- demonstration that return values for typed procedures are well defined for all inputs.

2.1.5. Compliance analysis

The purpose of Compliance Analysis is twofold. Firstly, to support semantic analysis in ensuring that each unit has been implemented in accordance with its specification and design, and secondly, and secondly to ensure that the implementation conforms to the rules of the used language and satisfies various integrity requirements.

The specified behavior of a procedure is expressed by a list which specifies how its outputs depend on its inputs, which are strong enough to ensure that the procedure will correctly perform its intended function without triggering an error handler or otherwise halting the CPU. Additionally post conditions are defined, which are used to formally express properties of the outputs.

When performing Compliance Analysis, it is often necessary to introduce extra statements into the procedure body to help an algebraic simplifier. Such extra code fragments are called proof tactics. In performing Compliance Analysis, the analyst must formally express the required behavior and properties of the code by developing these components.

If proof obligations are associated with modifications, these must be justified.

2.1.6 Reviewing

All the analysis work should be reviewed to ensure that it meets the required standards. The aim of reviewing was be as follows:

a) to ensure that specified quality standard was met

b) to ensure that all anomalies found were adequately reported

c) to assist in achieving consistency of analysis amongst the assessment team

d) to identity inconsistencies in the specifications and coding of different parts of the SW

e) to agree a satisfactory interface description of each procedure.

All work performed to correct deficiencies identified during a review should be documented to the same standard as required for the original work by updating the analysis report as necessary.

Productivity during the analysis should be continuously monitored. Because the analysis process generates large number of files and careful configuration control is therefore of great importance throughout. Also the quality assurance program should be maintain during the analysis process.

2.2. OTHER METHODS

2.2.1. Symbolic execution of software modules

For certain smaller software modules where the integrity requirement is very high, there are some other static analysis methods that can be used. One method is symbolic execution.

As the name indicates, it consists of executing a program with symbolic names rather than numbers. The result of a symbolic execution is a logical combination of algebraic expressions of variables. An example: the execution of the program statements

a:=2*b; if a > 4 then a:=3 else c:=1;

gives the result

b>2 and a=3 or b<2 and a=2*b and c=1

The result from a symbolic execution should be verified against a, preferably formal, specification. This method requires that the program is written in a programming language with certain restrictions, e.g. a safe subset of a high level language. One problem with the method is that the result becomes very bulky for a program over a certain size, and therefore difficult to interpret. It is therefore best suited for small modules, e.g. modules for logical expressions.

Another method is reverse engineering. This means one starts with the final version of a program and analyze it backwards to a higher level of description which can be compared to the specification. Program defects and violation of coding standards can also be detected in this way. A typical application for this method is to start with a hexadecimal dump of the final program on the host computer. This may be relevant for analysis of COTS modules were there are no code listing. Another use is to check for compile/linker/loader errors. This method is, however, rather time-consuming, and should mainly be used on critical modules.

2.2.2. Dynamic Testing

Dynamic testing means to execute a program with selected test data, and to compare the result with a presumably correct answer. However, as the number of input combinations often gets astronomical, even for normal programs, an exhaustive testing is in general impossible. Therefore testing can only find faults, not prove correctness. The objective is therefore rather to enhance the confidence in the program correctness.

There are various strategies for how to perform testing and select test data. Specification based testing means to select test data so that all aspects of the specification are tested. The advantage of this method is that one does not know any details of the final program. The set of test data, together with the expected results, can therefore be generated by the specifier of the program, so that misunderstandings between specifier and developer can be revealed. This is the most common testing practice, and should be the minimum requirements for an acceptance test.

Program coverage testing applies test data, which exercise certain subsets of elements in the program structure. Typical examples are branch and statements coverage, which checks that all branches and statements in the program are executed at least once. Path testing means to count the number of different program paths that have been traversed. A complete path testing is, however, not feasible in practice. The test data in this type of testing is usually chosen randomly from a probability distribution. Different types of distributions have different advantages. A very simple and distribution is the uniform one, and experiments have

shown that this is a very effective one to reveal program faults. One can also use distributions that enhances certain aspects of the program, e.g. to enhance the safety critical aspects to ensure that they are particularly well tested. A third strategy is simulation based testing, i.e. to simulate the input data one expect from controlled process, with random variations. This method is not so effective to reveal program faults, but is more relevant if one will use this to estimate software reliability during real operation. To perform coverage testing it is necessary to instrument the program with counters on each condition statement.

Mutation testing means to mutate statements in the program code to prove the absence of certain fault classes. This is not actually a test of the program, but a test of the testing strategy used, to see how effective it is to reveal program faults. This is also useful to check fault tolerance in fault tolerant computing.

To perform testing which requires a large amount of test data with corresponding answers, an automated dynamics testing system is necessary. Such a dynamics testing system consists of a test data generator, which gives input to the program system to be tested. It also contain an "oracle", i.e. a program which gets the same input data as the tested program and as well as the output from this program, and check whether the result is correct. This could for example be performed by a diversely made program. An illustrating example of the use of a dynamics testing system is described in the next section.

2.2.3. An example of testing using a dynamic testing system

A Dynamic Testing System which has been used to test the primary reactor protection system (PRPS) of the NPP Temelin (Czech Republic) under simulated reactor fault conditions constitutes a good practical example on how a dynamic testing system can be used. The objective of the test is to verify that the correct shutdown actuation's and safety system actuations are generated. For each reactor fault condition a large number of unique tests are performed, testing all elements of the PRPS including interactions between hardware and software, configuration data, final object code and time constraints.

The dynamics testing method involves subjecting the tested equipment to a large number of real time simulated fault conditions derived from reactor fault scenarios. For each fault scenario (known as base scenarios), the results of a plant transient analysis provide two bounding trajectories for each parameter. These trajectories represent two different accident severities. For a primary circuit leak for example, the results for a small leak and a large leak are provided. The individual test cases will be derived by randomizing the PRPS input parameters between the two values. This will allow the PRPS to be tested against a wide range of credible faults for each fault scenario. Each test case will place demands for reactor trip or ESF actuation on a number of different PRPS measurement parameters, and therefore each can be considered as a number of different challenges to the system.

The Dynamic Testing System is illustrated in the Figure E-2 below. It consists of an Offline System and an Online System.

The Online System consists of the two modules:

- The Input Driver and Data-link Simulator
- Controller and Data Logger (CDL)

The functions of the Offline System are.

- Generation of test input data
- Modeling of the PRPS functionality
- Comparison of „actual“ and „expected“ results
- Automatic reporting of results

These functions will be implemented in the following software programs:

- **Test Case Generator**. This program will read in the raw transient data (Base Scenario) and introduce randomized perturbations in order to generate large numbers of unique input transients. These transients will be written to separate files in suitable formats to be read by the Online System and the Offline System.
- **Logical Model**. This program will model the PRPS functionality and accept operator inputs specifying the test number of the first test and the total number of input files to be processed. The input data files will then be read in turn and processed to produce data corresponding to „expected output files“.
- **Comparator**. This program will accept operator inputs specifying the test number of the first test and the total number of tests to be processed. For each test, the „actual output files“ generated by the Online system will be compared with the corresponding „expected output files“.
- **Report generator**. This program is used to summarize the results of Dynamics Testing.

The Online System is configured as three units:

- Tested PRPS system
- Input Driver and Data-link Simulator which transfers the test data to a form suited for the PRPS system
- Controller and Data Logger which controls the correspondence between the input and output data from the PRPS system.

Test discrepancies detected by the dynamic test system falls into one of four categories.

(1) PRPS failures resulting from an error in the development process.

(2) Failures in the dynamic test system resulting from an error in the development process.

(3) Equipment failures in the PRPS or the dynamic test system

(4) Justifiable discrepancies due to the inherent limitations of the dynamics testing technique.

A final execution report is generated automatically by the Report Generator program. Added to this is a textual description of test investigations carried out and any problems

discovered. This is important because the length of the fault scenarios affects the total time required to perform dynamics testing, and also affects the size of the result analysis task.

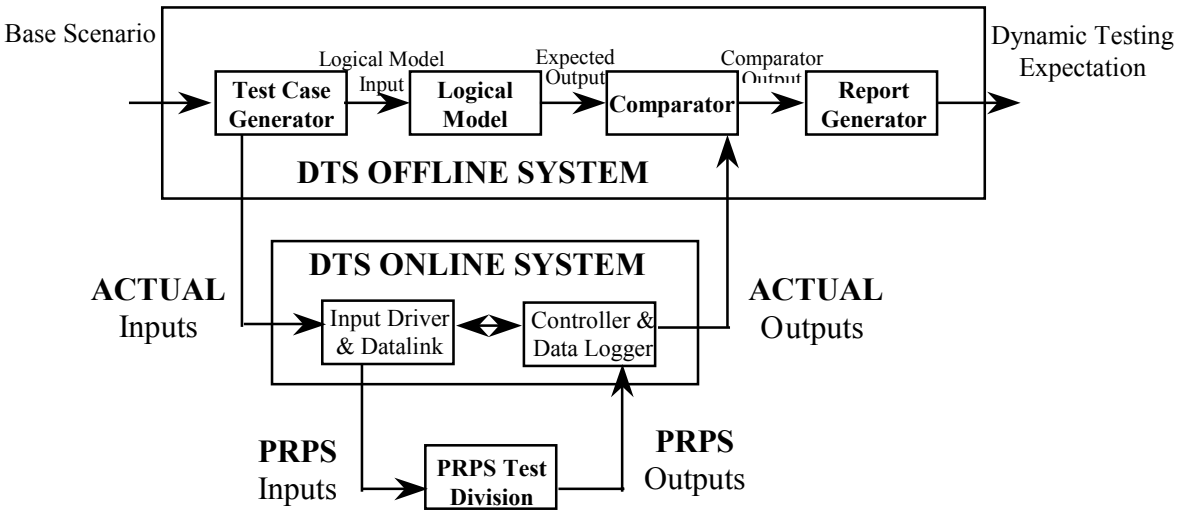


Figure E-2. Dynamic testing system.

3. SOFTWARE MEASURES

3.1. COMPLEXITY METRICS

The objective of complexity metrics is to measure the complexity of software modules and programs based on their structure. If a good correlation between the complexity metric and the number of faults could be established, then the metric could be used to determine the amount of attention that the various software modules should receive during development and testing. There are many different complexity measures, but none of them are generally accepted to be superior to others. To investigate this a set of common metrics were compared to the error counts of the modules of three test programs.

Some of the metrics, which can be applied, are given below:

- Simple metrics, like
  - number of lines
  - number of separations
  - number of variables
  - number of unique operators
  - number of unique operands
  - etc.

- McCabe's cyclomatic complexity [E-5]. This metric is based on the graph structure of the program, and corresponds to the number of closed regions in the graph, provided there are no crossing edges or sub-programs.
- Halstead's program length (L) and volume (V) estimate [E-6]. $N= n_1*\log_2(n_1)+ n_2*\log_2(n_2)$
- where n_1 is the number of unique operators and n_2 is the number of unique operands. $V=N* \log_2(n_1+ n_2)$
- Average number of entries and exits per software module – minimizing the number of entry/exit points is a key feature of structured design and programming techniques.

The general impression of this type of models is that they may provide a quick and easy way to compute a figure, which tells something about the complexity of the program and the propensity of making programming errors in them. However, the validity of these models are not proven, neither theoretically nor experimentally, and they are therefore not suited for reliability assessment of a safety related program for safety critical applications.

3.2. COVERAGE MEASURES.

Coverage measures are measures of to what degree all parts of a program are executed in a test, or in a real execution. The idea is that the probability that there are indigenous faults remaining in the program decreases the higher these measures are.

Program coverage measure is a measure of the fraction of a program code, which is executed. The best way to obtain such measures instrument the program with counters at each program branching point. Different types of coverage measures are

- *Statement coverage*, where the execution of statements in a program are counted.
- *Branch coverage,* where the execution of statements in a program are counted.
- *Condition coverage*, where the 'true' and 'false' values of all logical expressions in a program are counted.
- *Path coverage,* where each separate execution path through a program. This is the most rigorous coverage measure One can distinguish between entire paths, i.e. all statements executed from start to termination of a program, or module paths, which comprises all statements executed from start to termination of a program module, e.g. a subroutine. Since the number of paths is very large, complete path coverage is not to be expected, even in small programs.
- *Data coverage,* where the usage of data elements in the program are counted.

These measures are based on knowledge of the program code. This is, however, not always available, but here are other measures which can be made:

- *Input domain coverage.* The input space is divided into a set of domains, and the number of executed domains is counted.

- *Program property coverage.* This is a measure of to which degree the required properties given in the requirement specification and design documents are tested.

3.3. FAULT SEEDING METRICS

Fault seeding models compute the estimated number of remaining faults in a program, and the confidence in fault freeness, based on seeding and retrieval of artificial faults in the program. This method presupposes that one has the possibility to alter the program, and also the necessary knowledge about the program to seed faults into it. For this model to be valid these faults must also be as concealed as one can expect real faults to be, i.e. it implicitly assumes that the seeded faults have the same probability of being detected as the indigenous ones. Also, a seeded fault may "mask" an indigenous fault in the sense that it makes the indigenous fault undetectable.

The most popular and basic fault-seeding model is based on an unpublished paper by Mills in 1970.

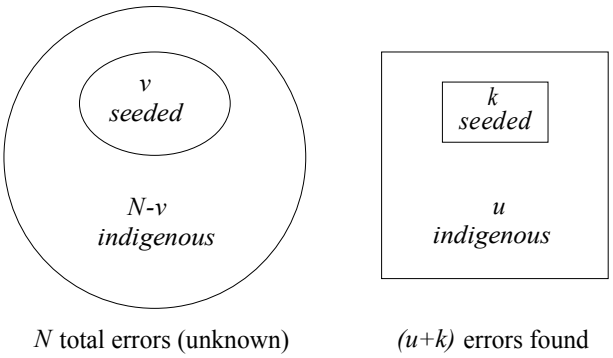


Figure E-3. Basic fault seeding model.

A fairly simple way to derive the above result is to consider Figure E-3. Here we have, in the circle on the left, all the faults in the program N, which includes v seeded faults. In the rectangle on the right, we have the (u+k) faults that were found during testing, where u is indigenous and k is seeded faults. If N and (u+k) are large, the proportions between the seeded and indigenous faults should be approximately the same in both cases, i.e.

$$\frac{v}{N-v} \cong \frac{k}{u}$$

Solving this gives

$$N = \left\lfloor \frac{v(u+k)}{k} \right\rfloor \qquad or \qquad N - v = \left\lfloor \frac{v \cdot u}{k} \right\rfloor$$

If one keeps on testing until all the seeded faults have been found (k=v), the number of remaining faults is estimated to zero. While this is a valid estimate, it is not statistically significant. Instead, one can estimate the probability that the remaining number of faults is less than or equal to some number w:

We make the *a prior* assumption that

$$P(N = n) = \begin{cases} 0 & n < u + v \\ C & n \geq u + v \end{cases}$$

where N is the true number of faults (indigenous and seeded), and C is some constant. For purposes of calculation, the number of faults n is set to be infinite when we derive equation 1.5.9 below. When we estimate the probability that there are less than or equal to w faults left in the program, we assume n to be finite, and we get the approximation

$$P(n \leq (u + v + w) | (u + v) found) \approx 1 - \frac{\binom{u+v-1}{v-1}}{\binom{u+v+w}{v-1}}$$

The method is applicable to safety critical programs where no real faults are found during testing. The result is a confidence in fault freeness, and not a reliability figure. It can therefore not be used directly in e.g. a PSA evaluation together with hardware reliability estimates. The method requires a high number of seeded faults to obtain a sufficiently high confidence level for safety critical applications.

In the special case where no indigenous faults are found, and all the seeded faults are found, we get the probability that there are no more faults in the program to be

$$P((n \leq v) \cap (w \leq 0) | (v \ found) \cap (u = 0)) = \frac{v - 1}{v}$$

This method presupposes that one has the possibility to alter the program, and also the necessary knowledge about the program to seed faults into it. For this model to be valid these faults must also be as concealed as one can expect real faults to be, i.e. it implicitly assumes that the seeded faults have the same probability of being detected as the indigenous ones. Also, a seeded fault may "mask" an indigenous fault in the sense that it makes the indigenous fault undetectable.

An advantage of this method is that it can be applied with different types of verification, testing, reviews, static analysis etc.

Experiments have shown that the method gives quite reasonable estimates when more than 50% of the seeded faults have been found.

4. STATISTIC RELIABILITY MODELING

Conventional reliability theory, which is based on reliability engineering of physical objects, in general is not suited for computer software. A computer program is not subject to wear-out. When a program is fault-free, it will remain fault-free forever, provided the environment in which the program operates does not change.

A measure of software reliability can be based on the statistical study of failures, which occur because of some defect in the program. The failure may be evident, but it may be difficult to see the error responsible, or what to do to make the fault disappear. Reliability models are supposed to provide quantitative information about the confidence one can have in the correct execution of a program. A simple estimate of the reliability could be based on a program test by dividing the number of failed tests with the number of executed tests. However, if a fault is revealed, the program would probably be corrected, and thus the reliability changed. And if no faults are found, such an estimate would not differentiate between no failures in 10 tests and no failures in a million tests. There are, however, developed a variety of more sophisticated models, which are described briefly below.

4.1. RELIABILITY GROWTH MODELS

Based on the sequence of times between observed and repaired failures, these models estimates the reliability and current failure rate, and predict the time to next failure and required time to remove all faults. These models are primarily used during the debugging phase of program development. However, they are also used for large systems, such as operating systems, which may fail and be corrected during periods of operational usage.

The general assumption of these methods is that a single system is followed chronologically with recording of times to failure, and that the faults are corrected. This is a situation, which for instance is easy to achieve in a debugging phase. However, the reliability figure obtained in this way estimates the probability of no failures if one continues to debug along the same way. It is questionable, however, whether this gives a realistic estimate of the reliability in real applications, with a different input distribution.

A more realistic estimate is obtained if the data are collected during real operation. This could be the case in the evaluation of a fairly large system, where a significant number of failures occur during operation. However, in order for the number of failures to be significant, it must be fairly high. It is questionable whether specific systems will reveal so many faults after they have been taken into use.

A type of systems which are known to reveal many faults, even after they are released, are large proprietary systems, as e.g. operating systems, information systems etc. The problem with such systems is that they are difficult to follow chronologically. Different faults are found by different users, which report them back to the producers. The same faults may also be found by different users. The producers will then usually collect the error reports, correct all the reported faults in the next program release, whereas the users will continue to use the faulty program, and possibly find new faults, until they get a new program release. None of the models, however, take into account this realistic scenario.

The methods do not distinguish between different types of faults, e.g. between real faults and more cosmetic faults. It is of course a possibility to only take into account a particular class of faults. In this case, however, one may easily obtain a number of faults, which is not significant. This is in particular the case when one only takes into account critical faults. An alternative way to measure the reliability of a program with respect to a certain class of faults is to perform the reliability estimation with all types of faults and multiply this with the ratio between the number of faults in the particular class and the total number of faults.

4.2. HIGH RELIABILITY SOFTWARE MODELS.

The reliability growth models are based on software failure data, and require a certain amount of data to give significant results. In case of safety related software this is usually not the case. For the final system one will usually not detect any faults, so the expected failure probability is zero, which is not very interesting. In such a case it is more interesting to consider the confidence level for the failure probability. Assume that one performs *n* tests of a program, with input data randomly sampled from a certain input profile, and no failure occurred. Then one can state with confidence

$$C=1-(1-q)^{n+1}$$

that the failure probability for the program $p<q$, if the input data correspond to the same input profile as during the test.

4.3. INPUT DOMAIN BASED MODELS.

In these models the input space is partitioned into a set of 'bins', and estimate the overall reliability from the reliabilities of the input partitions. These models assume that the input domain is partitioned into equivalence classes. An equivalence class is defined so that one can reasonably (but not with certainty) assume that a test of a representative value of the class is equivalent to a test with any other value of the same equivalence class, i.e. if one test with input from an equivalence class is correct, then the program is probably correct for all values in this class. The probability of failure is generally assumed to be the same for all members of the same equivalence class. The main advantage of these models is that one can obtain a high confidence in the reliability estimates with a limited number of tests.

An advantage of the input domain approach is the possibility of exhaustive testing within a particular bin. If the cardinality of a bin is small enough to allow exhaustive testing, and testing reveals no failures, then we can assign the probability of failure within that bin to zero. Unlike random testing, input domain testing can take immediate advantage of this information. Another advantage becomes evident if the usage distribution represented by the bin selection probabilities is changed. If we assume that the new distribution uses exactly the same partition of the domain, and that only the selection probabilities are changed, the new estimate of the overall reliability can easily be calculated.

4.4. Markov models

Markov models estimate the overall reliability from the reliabilities of the individual program modules and their transition probabilities. Essential for these models is the definition of the states and interpretation of what is meant with transitions between them. Different approaches have been used: A definition based on the control flow structure of the program, i.e. an actual state is defined by the program module with the locus of control. Another definition based on where one is in the debugging phase of the program. The third alternative is to define the states based on a Markov model of the usage of the system, or of the process the system shall control.

The use of finite state automata is a powerful way to specify a program, and this specification can be the basis for the states in the Markov chain. The knowledge about the usage or the process should make it possible to make reasonable guesses about the transition probabilities. This also defines a testing strategy to obtain a testing chain with the reliability parameters, and together they can be used to predict future behavior.

An advantage of a Markov model is the distinction between the stochastic behavior due to the actual usage and the one due to potential failures. It should thus be possible to recalculate the reliability for different usage profiles without changing the basic reliability figures. In particular it should be possible to compute the possibility to reach hazardous states, which is important for safety related software.

A computer program implemented in a safety critical system contains presumably no known faults. There is, however, a possibility that it contain unknown faults, and an alternative reliability measure is the confidence in fault freeness of the program, or more generally in the upper limit of the 'bug-size'. Such a measure can be made on statistical basis from data obtained during testing of the complete program, as well as of the different modules. One set of data can be gained through a controlled testing combined with a coverage measure. The latter requires a fairly detailed knowledge about the program structure, which is not always available when proprietary software modules are used. For these, however, there may be an additional large set of 'test' data obtained from information about the usage of the system.

Another type of measure is more qualitative expressed as a subjective judgement as a 'belief' in fault freeness. A methodology to use 'influence nets' (also called Bayesian networks or 'belief nets') and engineering judgement to combine evidences from different information sources for a quantitative assessment of this belief has been proposed [E-7]. Such networks can be used to show the link between basic information and the confidence one can have in a system. Each node in the network, except for the roots, expresses the 'belief' one can have in a statement, given information on its immediate predecessors in the net. Quantitative expressions for these conditional 'beliefs' must to a large degree be based on expert judgement or reasonable guesses. This might itself introduce further uncertainty and potentiality for error. In addition, there are serious unresolved difficulties in combining such disparate evidence in order to make a single evaluation of the overall dependability and thus to make a judgement of acceptability.

5. VALIDITY OF PRIOR EXPERIENCE

'Proven design', i.e. that a system has been used by a wide range of users over a long period, with no, or few, reported faults, is often used as an argument for high reliability by the producers of COTS based software systems. In principle this sounds reasonable, but in order to use this for a reliability assessment, more detailed knowledge is required. The total operation time from all users is one parameter, which could be used. This, together with reports on faults, which have occurred during operation, could be used in a reliability growth model.

The idea behind the claim of proven design is that long user experience should reveal all inherent faults, if they exist. So if no faults have been reported over a long period, this should

be a strong indication on error freeness. It is questionable, however, whether this is sufficient. A COTS based software system has often a quite general purpose, and consists of many standard modules, each of which has many modes of operation. To claim general high reliability of the system based on user experience, it is necessary to show the experience with all modes of operation of all software modules. This requires information about all installations of the system. Another aspect is the auxiliary software, which is used to generate the application program with COTS modules. This is used only once per installation. To get a broad user experience with such software it is necessary to have many installations. Information about the number of installations is therefore important for the reliability assessment.

Another information, which is of interest for the assessment is the number of versions of the COTS based software system, which are released. A new version implies changes in the system, and changes may have influence on the reliability of the system. It is therefore relevant to know which changes have been made, or at least where the changes are made. In an actual application one should know whether any changes were made in the software modules which are used in the application.

6. SPECIAL CONSIDERATION FOR ASSESSMENT OF HARDWARE DESCRIPTION LANGUAGES.

If applying Application Specific Integrated Circuit (ASIC) into safety system, their design usually described by Hardware Description Languages (HDLs). HDLs are a type of languages specially made for describing hardware systems consisting of concurrently operating components. It thus differs from most programming languages used in micro-controller which operates in time sequence. HDL programs will finally be mapped to hardware structures such as flip-flop, register, memory, combinatorial logic, etc, which will be executed parallel and independently (except for the information dependence between different parts), by hardware SILicon gates, after the so-called synthesis process. On the other side, the ordinary software instructions are executed serially in the CPUs. So HDLs have different syntax, are based on different principle, and have strict timing relationship and requirement than such languages. The assessment of HDL design will therefore require special considerations.

The main difficulty comes from the fact that the final logic represented by silicon gates array becomes from the original HDL design through several levels of synthesis or compilation processes, this synthesis process is done by synthesis tools automatically or based on the optimization level and optimization constrain defined by user. For HDL, the design may be transferred from behavioral level represent, then to Register Transfer Level (RTL) represent, to an expanded device independent logic network, to another modified device independent logic network optimized according to the operation time or silicon area, and finally to the logic network represented by the connection between the logic cells provided in the ASIC devices. This process may change the timing relationship, meanwhile the actual timing relationship will not be exactly same even for same hardware structure because of the variance of the device technology, but timing relationship is very critical and essential for ASIC device, so the development tools suppliers highly recommended to check the synthesis result for each step by simulation methods. And the things will become worse because the tools from supplier are not qualified according to nuclear standard. So, in order to make sure

the design coincides with the specification and what we expected, it is required to verify that the design in each level is correct. This can be done by checking the simulation results in each level, and finally checking the response of real system by real signals.

But to prove the behavior of ASIC with real signal is not so easy, because ASIC runs very fast, even with clock up to several hundreds of MHz, and all hardware components run simultaneously and parallel, and there are less pads to pick up the signal to check internal state of the high density ASIC device. So it is necessary to prove the design by software simulation (in each level) and finally prove the design in real signal in real time, and the testing should better be exhaustive.

Of course, it is very difficult to test a system exhaustively if with analog input, unless the system is very simple or only with binary signals. For system with analog input and complex logic, at least the test must cover all paths and all branches, and the treatment of the analog signal must be in a mathematically strict way. And it is recommended to use formal type of definition for the logic processing, and partition the solution space into small one, in order to make the test easier, possible for exhaustive.

Another issue about the assessment of ASIC-and-HDL-base design is to assess the self-test capability of the design, which must be coped with all failure pattern of the system. The failure pattern of ASIC can be divided into software failures (errors), hardware failures, and failure of input from outside. And hardware failure pattern includes the failure during initialization, single event upset (SEU), failure of part of the device, or failure of whole device. So the self-test must cover all the signal propagation paths, and must be supervised by other devices.

Only when the specialty of ASIC and HDL is well understood, the assessment of the HDL can be effective and cost effective .

7. COMBINATION OF EVIDENCE FROM DIFFERENT SOURCES

To evaluate whether the goal of a lifecycle phase has been fulfilled one should utilize all evidences, which may contribute to this goal. These evidences may be of rather disparate nature, both qualitative and quantitative. A problem is how to express observations about these evidences. Ideally these observations should be objective, but this is often difficult to obtain. Objectivity can be obtained if the evidence is in itself objective, either quantitative as e.g. the number of tests performed, or as objective facts, as e.g. that a certain coding standard is used. However, even if the evidence is in itself not objective, there are sometimes made objective procedures to observe these evidences. An example is program complexity, which is a qualitative concept, where there are constructed different metrics to measure this complexity.

Another problem is how to combine these evidences to obtain a common measure of the achievement of the goal. The Bayesian Belief Net (BBN) methodology is suggested as a way of combining the evidences. The objective of using BBNs in software safety assessment is to show the link between basic information and the confidence one can have in a system (see e.g. [E-8]). A BBN is a connected and directed graph, consisting of a set of nodes and a set of directed links between them. A variable, which can be in a set of states, is associated to each

node. Probability density functions over the states express the probability (or belief, confidence etc.) that the variable is in a particular of these states. This probability depends on the status of the variables represented by the start nodes at the incoming edges to the variable (the parent nodes). The edges represent conditional probabilities.

The nodes and associated variables can be classified into three groups:

Target node(s) - the node(s) about which the objective of the network is to make an assessment. This assessment is expressed with a quantifiable target variable. Typical examples of such nodes are “No faults in a program” or “No failure on demand for a trip”.

Observable nodes - nodes which can be directly observed. Some examples are “No failures during *N* test”, “No reported failures during previous usage of modules”, “All quality requirements are fulfilled” etc. The associated observable variables should be measurable or quantifiable, although this measurement may not be exact and objective, but based on judgment.

Intermediate nodes - nodes for which one have limited information, or only “beliefs”. The associated variables are the hidden variables. Typical hidden variables are development quality, producer’s reputation etc.

Application of the BBN method consists of three tasks:

- construction of BBN topology
- elicitation of probabilities to nodes and edges
- making computations.

The construction of the BBN is made gradually, by combining the target node(s) with the observable and the intermediate nodes. The aim is to combine all available relevant information into the net. One way to do it is to start from a target node and draw edges to nodes influencing this. Then from these nodes to draw edges to new nodes. In this way one will gradually build up a large BBN.

The next step is the elicitation of probability distribution functions (PDFs) to the nodes and edges. To begin with one gives prior PDFs to hidden variables (some, at least the top nodes, but not necessarily all), and conditional PDFs for the influences represented by the edges. These PDFs may be either continuous functions or they may be discretized. The latter means that the range of the variables is dividend into a finite number of states. This has some advantages, both because it is conceptually easier in an expert judgment to assign discrete values, and because it usually makes the computation much simpler. The conditional probabilities for edges between discrete variables are given as dependency matrices between the states of the variables associated with the start node and the end node of the edge respectively. For the observable nodes one would expect exact values, and therefore no PDF. However, for some observables the quantification is somewhat fuzzy, and so a PDF might also be the appropriate representation of these.

The computation method is to insert observations in the observable nodes, and then use the rules for probability calculation backward and forward along the edges, from the observable nodes, through the intermediate nodes to the target node (which again can be an intermediate node in a BBN at a higher level). Forward calculation is straight forward, while backward computation is more complicated. It can, however, be solved using Bayes methodology.

An illustration of a BBN for the *Concept* phase is shown in Figure E-4. The goal, *Preparedness,* expresses the confidence at this stage that the company has properly considered the safety aspects of introducing a computer based safety system, and also that it is experienced enough to be able to handle the safety aspects throughout the system development. The bottom nodes represent the observable evidences. In this example these evidences are of a qualitative nature. In order to give a quantitative value to the observation a set of corresponding questions where one can give answers on a scale from "yes" to "no". The PDFs associated with the edges between nodes represents quantitative expressions of the importance the value of the "child" node has on the "parent" node.

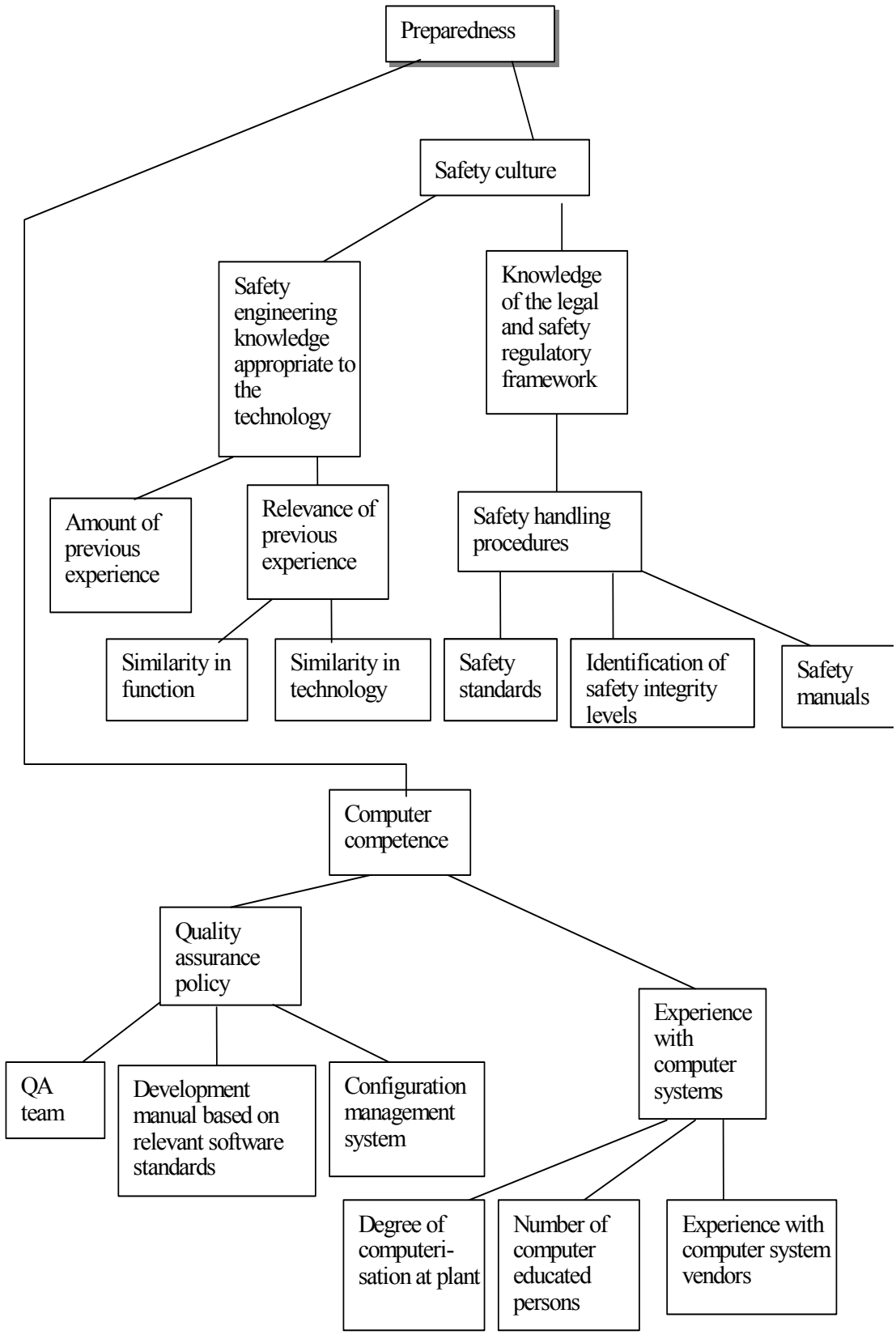


Figure E-4. BBN for the concept phase.

REFERENCES TO APPENDIX E

[E-1] N.G. Leveson and P.R. Harvey: “Analyzing Software Safety,” IEEE Trans. Software Eng., SE-9 (5), 569–579, 1983.

[E -2] N.G. Leveson: "Safeware", Addison Wesley Publishing Company 1995.

[E -3] E. O. Ovstedal: “Using Fault Tree Analysis in Developing Reliable Software” Proceedings from SAFECOMP ‘91, 77–82, 1991.

[E -4] G. Dahll 'Safety Evaluation of a Train Leader Telephone System” SAFECOMP’99, Toulouse Set 27th to 29th 1999.

[E -5] T.J McCabe. “A Complexity Measure”. IEEE Trans. Software Eng., SE-2 (4), 308–320, 1976.

[E -6] M.H. Halstead: “Elements of Software Science”, Elsevier North Holland Publ. Co. New York 1977.

[E -7] Neil, M. and Fenton, N.: “Predicting software quality using Bayesian Belief Networks”. In: Proceedings of 21st. Annual Software Engineering Workshop, NASA Goddard Space Flight Centre (1996).

[E -8] Fenton N., Littlewood B., Neil M., Strigini L., Sutcliffe A. and Wright D., 1998. “Assessing Dependability of Safety Critical Systems Using Diverse Evidence.” IEEE Proc. on Software Engineering, 145 (1), 35–39.

CONTRIBUTORS TO DRAFTING AND REVIEW

BOZHENKOV, O.	Research Institute of Nuclear Power Plant Operation, Scientific and Technical Center for Informational Technology, Russian Federation
CALVERT, J.A.	US Nuclear Regulatory Commission, United States of America
DAHLL, G.	OECD Halden Reactor Project, Norway
FU, Li	Institute of Nuclear Energy Technology, Tsinghua University, China
HARJU, H.	VTT Automation Industrial Automation, Finland
JOHNSON, B.W.	Department of Electrical Engineering, University of Virginia, United States of America
KIM, Jang-Yeol	Korea Atomic Energy Research Institute, Republic of Korea
KANG, Ki-Sig	International Atomic Energy Agency
ANDREI, K.	International Atomic Energy Agency
NASER, J.	EPRI, Science and Technology Development, United States of America
PALAMIDESSI, H.A.	Nucleoelectrica Argentina S.A., Argentina
RUBEK, J.	I&C Energo, Czech Republic
WACH, D.H.	Institut für Sicherheitstechnologie GmbH, Germany
YASTREBENETSKY, M.	State Technical Center on Nuclear and Radiation Safety, Ukraine

Research Co-ordination Meetings

Vienna, Austria: 8–12 November 1999

Halden, Norway: 4–8 December 2000

Vienna, Austria: 29 October–2 November 2001