KAERI/AR—591/2001

技術現況分析報告書

액금로 디지털 보호계통 소프트웨어 안전성 평가 기술

(Software Safety Analysis Techniques for Developing Safety Critical Software in the Digital Protection System of the LMR)

2001. 2 월

韓 國 原 子 力 研 究 所

제 출 문

한국원자력연구소장 귀하

이 보고서를 2001 년도 "유체 및 계측제어 계통 설계기술개발" 과제의 기술현황분석보고서로 제출합니다.

2001 년 2 월

주관연구기관명: 한국원자력연구소

주저자: 이 장수

공동저자: 천 세우

김 창회

심 윤섭

요 약 문

전세계적으로 디지털 보호계통의 소프트웨어 안전성을 보장할 수 있는 확실한 기술이 없는 실정이다. 다만 품질보증, 수학적 검증, 안전성 평가, 형상관리, 확인 및 검증, 신뢰도 분석, 심층방어 및 다양성 기술 등 관리적 측면과 기술적 측면에서의 철저하고 다양한 노력을 통해 안전에 대한 믿음의 정도를 수용 가능하게 높임으로써 설계 인증 및 운전 허가를 받게 된다.

본 기술현황 분석보고서에서는 액체금속로(LMR: Liquid Metal Reactor) 보호계통 소프트웨어 안전성 현안을 파악하고 소프트웨어 안전성 보장을 위한 관리적 측면의 기술현황을 분석하였다. 3장에서 소프트웨어 개발공정 각 단계별로 필요한 소프트웨어 안전성 분석행위를 살펴보았다. 4장에서는 계통 측면과 소프트웨어 측면에서의 안전성 분석 기술을 비교 분석하였고 5장에서는 소프트웨어 신뢰도 평가 기술을 파악하였다. 아직까지 소프트웨어 신뢰도의 정량적 평가 가능성 자체가 논란이 되고 있는 소프트웨어 신뢰도 분석 관련기술의 현황을 파악하였다. 6장에서는 디지털 보호계통 개발 시 가장 심각하게 대두되고 있는 공통모드 고장(CMF: Common Mode Failure)의 방어 기술을 분석하였다. 차후 액금로 I&C 계통 설계시 공통모드 고장을 방지하기 위한 방법들에 대한 기술현황 분석과 아울러 분석 지침을 제시하였다. 소프트웨어 고장 허용(Fault Tolerance) 방법, 가능한 CMF 원인 및 파급 효과, 소프트웨어로 인한 CMF 방어 대책 및 Guideline 변천과정, 미국 NRC 의 4가지 규제입장, 적용 가능한 규제 법안 및 가이드라인 등에 대해서 기술 현황을 조사하였다.

Abstract

This report has described the software safety analysis techniques and the engineering guidelines for developing safety critical software to identify the state of the art in this field and to give the software safety engineer a trail map between the code & standards layer and the design methodology and documents layer. We have surveyed the management aspects of software safety activities during the software lifecycle in order to improve the safety. After identifying the conventional safety analysis techniques for systems, we have surveyed in details the software safety analysis techniques, software FMEA(Failure Mode and Effects Analysis), software HAZOP(Hazard and Operability Analysis), and software FTA(Fault Tree Analysis). We have also surveyed the state of the art in the software reliability assessment techniques. The most important results from the reliability techniques are not the specific probability numbers generated, but the insights into the risk importance of software features. To defend against potential common-mode failures, high quality, defense-in-depth, and diversity are considered to be key elements in digital I&C system design. To minimize the possibility of CMFs and thus increase the plant reliability, we have provided D-in-D&D analysis guidelines.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

I.1 Purpose

The development, use, and regulation of computer systems in nuclear reactor Instrumentation and Control (I&C) systems to enhance reliability and safety are the complex issues. Software cannot be proven to be error-free, and therefore is considered susceptible to common-mode failures because identical copies of the software are present in redundant channels of safety-related systems. To defend against potential common-mode failures, high quality, defense-in-depth, and diversity are considered to be key elements in digital I&C system design. This report describes the software safety analysis techniques and the engineering procedures for developing safety-critical software. This report has a role of glue to fill a gap between the mandatory requirements (*what*) and the work practices (*how*) when conducting the processes and activities intended to improve the safety of safety-critical software.

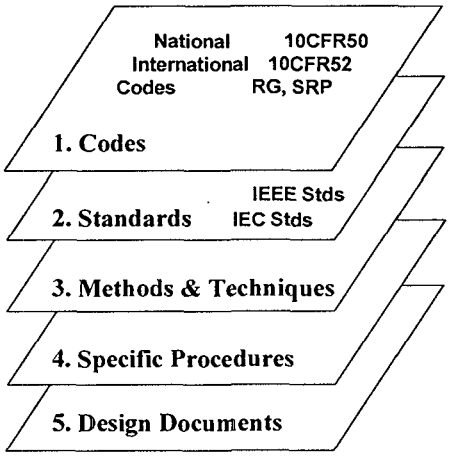


Fig. 1.1 Standard Framework

There is a framework of methods, as illustrated in Fig. 1.1, to justify the safety of a critical software, from codes and regulations, through industry-specific standards, generic standards and guides, to the developer's work practices. The codes and standards layer can be roughly grouped as an international and a member country's positions and standards, We have surveyed the 3rd layer techniques to guide the gap between the 2nd and 4th layer, which may also be a glue to resolve the discrepancies between the international and national standards, specially to meet the related IEC and IEEE framework of standards.

The purposes of this report are to identify the state of the art in this field and to give the software safety engineer a rough map between the code & standards layer and the design methodology and documents layer for the software important to safety in nuclear power plants.

The objective of this report is to survey methods which can be used to evaluate the reliability and safety of 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. The evaluation methods will generally be of qualitative nature, although quantitative methods will be described where they are appropriate. The evaluation should be based on all relevant evidences available from documentation at each stage of the system development. The evaluation methods will be based on a variety of techniques, including questionnaires, well-known risk analysis methods, document metrics, statistical methods etc.

Important issues to improve safety integrity are the structure of an I&C system. 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 characteristic of programmable processors. In this context common mode failures, which may be introduced by software faults, are of specific concern. A distributed software-based system usually has, at least to some extent, data sharing and the same software modules in redundant channels which make the overall system vulnerable to CMF. We have surveyed the techniques and guide to protect the CMF.

1.2 Scope

This report describes the software safety analysis techniques and the engineering procedures according to the software safety plan used for the development, procurement, maintenance, and retirement of safety-critical software; for example, software products whose failure could cause loss of life, serious harm, or have widespread negative social impact. This report requires that the plan be prepared within the context of the system safety program.

Because software safety can be analyzed from the relationship between a logical fault of software and a physical hazard of a system, the software safety process should be a subset of the system safety process. However, current approaches for analyzing the software safety, originated from system safety techniques, do not provide formal basis of conducting systematic safety analysis of software, in particular, for software requirements. Software fault tree analysis is the most commonly employed safety analysis technique, and its industrial practice depends heavily on technical expertise of human analysts about the physical hazard and is often ad hoc. The scope of this report includes only the safety aspects of the software. This report does not contain special provisions required for software used in distributed systems or in parallel processors.

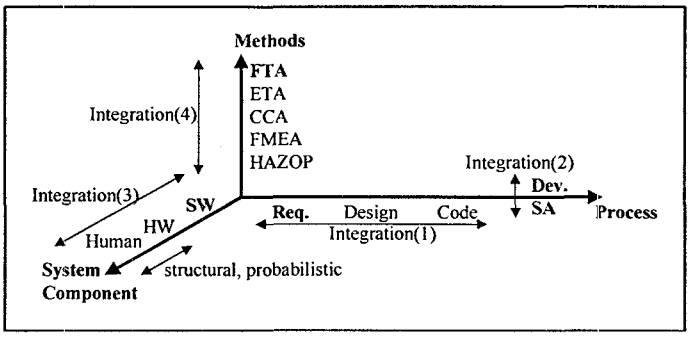


Fig. 1.2 Integration of Safety Analysis Methods

There are already a number of methods and tools that claim to assist the software safety analysis, several of which will be discussed in more detail later. However, it is our belief that these methods suffer from poor integration into the system development lifecycle at several levels. There must be an integrated safety analysis method that has several dimension as shown in Fig. 1.2 because of following reasons:

1. Because it is critical to analyze the safety of a whole lifecycle to develop software, there must be an integrated and iterative safety analysis method through a software development lifecycle. The safety of software cannot be analyzed by applying a method at the final product. There are no failure modes according to an aging of components like in electro-mechanical hardware. When analyzing the safety of software using a traditional fault tree analysis method, it is inappropriate to analyze the safety according to the structure of components and to apply numbers into the leaf modules of software in order to calculate a quantitative reliability.
2. It is preferable to integrate the safety analysis method and a formal development method in a semantic level because a development process and a safety analysis process are the different sides of a same coin. It will make easy to change the design according to the analysis results, and make the safety analysis method be more systematic, which will not depend on engineering judgment of analysts.
3. Software itself is not hazardous. A software safety analysis is to search the logical contribution of software faults to the physical hazard of system. Software safety analysis method must be able to integrate with hardware safety analysis and human safety analysis, and also an upper level analysis method of system safety
4. Even though there are many possible methods to be applied at the analysis of software safety, no one technique can guarantee the completeness of safety analysis. It is preferable to be integrated among those methods, such as Failure Modes and Effect Analysis (FMEA), Failure Modes, Effect, and Criticality Analysis (FMECA), Fault Tree Analysis (FTA), and Hazard and Operability (HAZOP), and configurable according to applications.

The final goal of our research on the software safety is to try to establish the integrated safety analysis approach in all the dimensions. We have developed a software safety analysis technique, the Causal Requirements Safety Analysis (CRSA) [Lee00]. The goal of CRSA is to present a software fault tree analysis method, which can be integrated with a formal method and a system safety analysis method. The target system to be analyzed by CRSA is an embedded real-time system like a digital protection system in LMR. Generally, a system consists of the hardware, software, and human (or operator). CRSA is focusing into only the software safety analysis at a conceptual requirements phase, by assuming that the hardware and human factors are perfect.

1.3 Application

The engineering procedures for the software safety are prepared under the direction of project or system safety program management to address the identified potential software safety risks. The level of detail in, and the resources required by an software safety plan will be determined by factors including the type and level of risks associated with the software product, the complexity of the application, and external forces such as contractual requirements.

Software is a portion of a system. Other portions of that system include computer hardware, other devices (possibly including mechanical, electrical, chemical, or nuclear devices), and people. Software alone is not a safety issue; it is only an issue in the context of this larger system. Hence, software safety must begin with the larger system. Software safety must be considered in the context of its associated hardware, environment, and operators. The engineering procedures for the software safety need to address interfaces with these elements.

1.4 Modeling Safety

1.4.1 Security Model

A general model for security systems is that certain assets exist, and certain threats exist. Typical assets are people, equipment and information. Examples of threats are assassins, saboteurs and spies. A security system has three primary functions. First, keep the assets and threats apart. Second, in recognition that this is not always possible, detect any contact between an asset and a threat in time to prevent damage. Third, in recognition that this is not always possible either, detect damage in time to permit some recovery. Consider the case of protecting military documents. The documents will be kept in a secure room surrounded by several layers of fences and guards; getting access to the room requires increasing scrutiny as one passes through the layers of defense. The room itself will contain a variety of detection devices, designed to detect pressure, heat and motion. Finally, counterespionage agents keep alert to any indication that the documents have been compromised.

1.4.2 Basic Safety Model

An analogous model can be created for safety systems. Again, there exist certain assets and certain threats. Typical assets are people, equipment, the living environment and the non-living environment. Examples of threats are fire, poison and explosion. A safety system has three primary functions. First, keep the assets and the threats apart - that is, keep the system in a nonhazardous state. Second, if the assets and threats start coming into contact, separate them. That is, if the system moves into a hazardous state, the safety system is expected to move it back to a nonhazardous state. Third, if contact cannot be prevented, attempt to reduce the damages. In other words, if an accident occurs, control the damages. In the following, the first two functions are grouped under the general heading of accident-prevention.

1.4.3 Classification of Safety Systems

The safety portions of application systems that involve safety considerations can be classified in different ways. Here's one way to cover the accident prevention functions.

1. First order systems are those that are immediately concerned with hazards. That is, first order systems actively work to keep assets and threats apart. There are at least two subclasses.
    a) *Shut-down systems* have, as a basic goal, to monitor the threats, and (if a hazardous event occurs) to cause the application system to move to a safe inactive state. That is, a shut-down system attempts to prevent an accident by ceasing operation (shutting down) in a safe manner, in order to provide time for diagnosis and remedial action. Examples are shut-down systems in power plants and chemical plants. Automated highway systems may also be examples.
    b) *Keep-going systems* have, as a basic goal, to monitor the threats, and (if a hazardous event occurs) to take remedial action to remove the threat. That is, a keep-going system will itself diagnosis the problem and take remedial action. This is done because shutting down is not a feasible option. Examples are aircraft control and space craft control. It appears that most keep-going systems involve vehicles that are not in contact with the ground.
2. Second order systems are those used to create or service first order systems. Simulation models of aircraft and power plants are examples. Information management systems (used, for example, in decision making in a hospital patient-care application) also are examples of this category.

1.4.4 Refining First Order Models

There are probably many ways to refine the accident-prevention first-order safety model. The method presented here divides the safety functions into three complementary principles (submodels), termed diversity, defense-in-depth and quality parts.

- *Defense-in-Depth (DiD or D-in-D)* means that there is more than one way to keep the assets and threats separated. In a chemical plant where a potentially explosive batch is being mixed, (1) use a strong vessel capable of resisting considerable over-pressure, (2) place the vessel in an isolation building with strong walls, (3) locate the building in an isolated portion of the plant site, (4) locate the plant away from population centers, and (5) have emergency evaluation procedures worked out in advance.
- *Diversity* means multiple ways to obtain the same information or to carry out the same actions. Examples are multiple temperature sensors in the vessel, and multiple valves on the gas feed line to the heater. Use of a temperature sensor and a pressure sensor, either of which is sufficient to deduce over-pressure, is also an example.
- *Quality* parts means building components of sufficient quality that failure is rare. The three principles are complementary. They should be used in designing a safety system in a plant so that:
  - The components of the plant are of sufficient quality that the safety system is rarely needed. A possible goal might be no more than a few times per year.
  - The safety system is of sufficient quality that failure of individual components of that system are rare. Say, a few times per decade.
  - Diversity is used so that if a component of the safety system does fail, other components are available as substitutes. A goal might be that failure of the entire safety system happens a few times a century.
  - D-in-D&D is available to handle the rare instances when the safety system does fail. This might reduce the probability of damage to assets to a few times per millennium.

The principles apply at multiple levels of design. For example, this set of models is itself an example of defense-in-depth. These principles are well known in the nuclear power field, and have been used to design nuclear power plants for many decades. Observation indicates that they are used in many other applications as well.

1.5 Safety Issues of Digital I&C Software for the LMR

We had already identified the issues on the software verification and validation (V&V) techniques for the digital instrumentation and control system of the advanced nuclear power plants. They include the technologies against the common-mode software failures, the safety and reliability

assessment methods, practical formal methods, and so on. Another issue in this problem is how will software reliability measures be used in PRA and what characteristics must the software reliability measure have.

Along with important benefits, such as computational capabilities and flexibility, digital I&C systems introduce potential new failure modes that can affect operations and margins for safety. In order to successfully apply the digital technologies into the I&C system of the LMR, we have identified the technological and strategic issues.

- Digital I&C requires rigorous treatment of the system aspects of their design and implementation. For example, system level hazard analysis should extend into software components by systematic and formal approaches to ensure that software does not contribute to system hazards.
- For software quality assurance, the main issue is how to define the generally accepted methodologies in the spectrum of technologies related to the process and product assurance, in the trade-off between the rigorous approaches and cost-effective approaches. In other words, we have to be able to answer these questions, "how safe is safe enough?," "how much test is test enough?," and so on.
- Digital technology introduces a possibility that common-mode software failures may cause redundant safety systems to fail in such a way that there is a loss of safety function. There are still lots of uncertainties in the technologies against the common-mode software failure.
- We have identified the issues on the safety and reliability assessment methods. The bulk of the safety case for a 'traditional' engineering discipline, e.g., for mechanical or hydro-mechanical systems, comprises safety analyses, which provide the evidence that the expected hazard rate is tolerable. This evidence is based on known component failure rates, and an analysis of the design. However, it is comparatively rare to see a safety case which includes the use of safety analyses on the software.
- We have identified the issues on Commercial-Off-The-Shelf (COTS) software assessment methods, including the assessment of ASIC and PLC based software.
- We have identified that there must be an investigation of effective testing techniques, including the linkage between testing and reliability methods.
- We identified the issues on practical formal methods, and the integrated approach for the multi-disciplined engineering. It will be studied how formal methods can contribute to the elicitation, specification, and validation of software requirements to fill the gap between system and software. A research topic will be the formalization of traditional software specification, with particular emphasis on graphical formal methods.

2. MANAGEMENT OF SOFTWARE SAFETY ACTIVITIES

2.1 Introduction

The goals of system safety can be achieved only with the support of management. In general, management is responsible for setting safety policy and defining goals; defining responsibility, fixing accountability, and granting authority; establishing communication channels; and setting up a system safety organization.

Management has its greatest influence on safety by setting safety policy and goals, defining priorities between conflicting goals, establishing procedures for detecting and settling goal conflicts, and setting up incentive structures.

A policy is a written statement of the wisdom, intentions, philosophy, experience, and belief of an organization's senior managers that guides attainment of stated goals. A safety policy should define the relationship of safety to other organizational goals and provide the scope for discretion, initiative, and judgement in deciding what should be done in specific situations.

A safety policy contains such things as the goals of the safety program; a set of criteria for assessing the short- and long-term success of that program with respect to the goals; the values to be used in trade-off decisions; and a clear statement of responsibilities, authority, accountability, and scope of activities. Procedures must exist for reporting back to the policymakers any problems in carrying out the policy.

A safety policy may be broken into two parts: (1) a document that concisely states general policy and organization and (2) a more detailed document or set of documents including standards, manuals, and handbooks describing rules and procedures. Detailed standards have both advantages and disadvantages: They ensure a minimum level of practice, but they also can inhibit flexibility and optimisation for particular circumstances.

Not only must a safety policy be defined, it must be disseminated and followed. Management needs to ensure that safety receives appropriate attention in decision making. Progress in achieving goals should be monitored and improvements identified, prioritised, and implemented. The flexibility to respond to safety problems needs to be built into the organizational procedures. For example, schedules should be adaptable to allow for uncertainties and possibilities of delay due to legitimate safety concerns, and production or productivity goals must be reasonable.

Perhaps most important, there must be incentives and reward structures that encourage the proper handling of trade-offs between safety and other goals. Not only the formal rewards and rules but also the informal rules (social processes) of the organizational culture must support the overall safety policy.

When conflicting goals exist, proper trade-offs can be ensured using two management approaches: (i) establish strict and detailed guidelines, which sacrifices flexibility, or (ii) leave the decisions to employees, which allows more opportunity for major errors of judgement. A compromise strategy between the two extremes leaves much of the decision-making authority in the hands of employees, but also establishes incentives for following general organizational safety policies. For this compromise strategy to work, employees need to feel that they will be supported by management when they make reasonable decisions in favour of safety over alternative goals.

2.2 Software Safety Analysis Reporting Requirements

There are three major types of documentation in system safety: planning documents, information systems, and reports.

2.2.1 Software Safety Planning Documents

The system safety program plan (SSPP) is a management document that describes the system safety objectives and how they will be achieved. It provides a regulatory agency, contracting agency, or manager with a baseline with which to evaluate compliance and progress. Developing and gaining approval of a plan should be the first step in any safety-critical project.

Plans for subsystem safety should be part of the SSPP rather than in separate documents. Safety is a system quality, and separate documents for the subsystems appears to have only disadvantages. Thus planning for software safety should be included within the overall system safety plan; it should not be a separate and hence potentially inconsistent or unintegrated process. Software development too often is separated from the overall system engineering process, with unfortunate results. However, the subsystem development groups should have major input into the SSPP, or compliance may be difficult to achieve.

Many standards exist for program plans-each is a little different, but all contain similar information. Devising a general plan for everyone is not practical, however: Each plan needs to be tailored to its project and goals and to fit the corporate personality and management system.

As shown in Fig. 2.1, the following information should be included in the software safety plan document under SSPP.

2.2.2 Safety Information System

Although setting up a comprehensive information system can be time consuming and costly, such a system is crucial to the success of safety efforts, and resources invested in it will be well spent.

1. Purpose
2. Definitions, acronyms and abbreviations, and references
3. Software safety management
3.1 Organization and responsibilities
3.2 Resources
3.3 Staff qualification and training
3.4 Software life cycle
3.5 Documentation requirements
3.6 Software safety program records
3.7 Software configuration management activities
3.8 Software quality assurance activities
3.9 Software verification and validation activities
3.10 Tool support and approval
3.11 Previously developed or purchased software
3.12 Subcontract management
3.13 Process certification
4. Software safety analyses
4.1 Software safety analyses preparation
4.2 Software safety requirements analysis
4.3 Software safety design analysis
4.4 Software safety code analysis
4.5 Software safety test analysis
4.6 Software safety change analysis
5. Post development
5.1 Training
5.2 Deployment
5.2.1 Installation
5.2.2 Start-up and transition
5.2.3 Operations support
5.3 Monitoring
5.4 Maintenance
5.5 Retirement and notification
6. Plan approval

Fig. 2.1 Contents of Software Safety Program Plan

Control of any activity requires adequate and accurate information. Documenting and tracking hazards and their resolution are basic requirements for any effective safety program. All hazards need to be recorded, not just the most critical; otherwise, there is no record that a particular condition has already been evaluated. Because the hazard list may become quite large, hazard or problem priority lists summarising the most significant information on complex programs can be useful for reviews and management oversight. The complete hazard log and audit trail will show what was done and how and why decisions were made.

An organization's system safety information system will contain such information as the System Safety Program Plan and the status of its activities, results of hazard analyses, tracing and status information on all known hazards, incident and accident information including corrective action, trend analysis data, and so on. Interfaces with various project databases, such as the software configuration control database, should be well defined.

A crucial aspect of any management system is the feedback of information on which to base future decisions. Information can be used to describe, to diagnose, to compare, to evaluate, and to improve. For example, a safety information system can provide the information necessary (1) to detect trends and deviations that presage an accident, (2) to evaluate the effectiveness of safety

controls and standards, (3) to compare models and risk assessments with actual behaviour, and (4) to identify and control hazards and to improve designs and standards.

An effective information system must consider not only accidents, but also incidents or near-misses. An accident usually occurs only when the complete set of necessary conditions exist; a near miss or incident results when only some of these conditions exist. Whether all or some of the causal conditions exist at any time is often a matter of luck-usually the ratio of incidents to accidents is several orders of magnitude. Examination and understanding of near misses can warn of an impending accident and also provide important information about what conditions need to be controlled.

No matter how the information is collected, understanding its limitations and inaccuracies is important. Simply collecting information is not enough - the information must be accurate and timely, and it must be disseminated to the appropriate people in a useful form. Three factors are involved: information collection, analysis, and dissemination

2.2.2.1 Collection

Various types of information can be collected: performance and operational data, accident and incident investigations, results of studies and evaluations, technical information such as standards, manuals, and professional literature, and so on. Operational data, obviously, is the most difficult to accumulate; only a limited sample of operational data may be available for a particular hazard. Collection of such data for a single company or organization may be very expensive and require a large quality control group working over an extensive period of time. Industry-wide efforts may be more practical, but it is difficult to obtain comparable data from multiple sources in an unbiased and systematic manner. Data may be seriously distorted by the way it is collected. The two main problems in collecting data are systematic filtering or suppression and unreliability. Data usually must be obtained from written descriptions of events in a report about an accident or near miss. Such reports tend to identify only the proximal events (events closest in time) and not events that are early in the sequence or causal factors that are at a higher level in the causal framework such as management problems or organizational deficiencies.

Hazards, by their nature, involve relatively rare and unpredictable events, and these events are therefore difficult to describe. Data collection is more reliable for accidents that are similar to those that have occurred in the past than for accidents in new types of systems where past experience on hazards and causal factors are less well understood. Software errors and computer problems are often omitted or misdescribed because of lack of knowledge, lack of accepted and consistent categorisations for such errors, or failure to consider them seriously as a causal factor.

2.2.2.2 Analysis

Data, once collected, needs to be analysed and summarised. Systematising and consolidating a large mass of data into a form useful for learning is difficult. Raw quantitative data can be misleading and should always be tested for statistical significance. Remember, however, that statistical analysis alone is not enough-it too can be misleading and can leave out important information for hazard control.

2.2.2.3 Dissemination

Dissemination of information in a useful form may be the most difficult aspect of maintaining an information system. If information is not presented to decision makers in a meaningful way, learning from the data is inhibited. Traditionally, information has been disseminated in checklists, standards, and codes of practice.

Accidents are often repeated, and accident/incident files are one of the most important information sources for hazard analysis and control. However, the information needs to be presented in a form that people can learn from, apply to their daily jobs, and use throughout the life cycle of projects, not just in the conceptual design stage. Accidents are frequently the result of risk decisions that were changed by default when operations changed or are due to insufficient updates to the hazard analysis when engineering modifications were made.

2.2.2.4 Safety Reports

The hazard report contains a description of the potential problem and what is being done about it. These reports are compiled into the hazard catalogue or log to form the basis for the hazard auditing and tracking system. As each hazard is identified, it should be documented on a uniquely numbered hazard report form and tracked through closure.

The hazard report form includes, at least, a description and classification of the hazard, a history of action taken, and some verification that the action has been taken. It might also include the system or subsystem involved, the operational phase, the cause(s) and possible effects, and corrective or preventive measures.

In the normal design documentation, a section should be included that describes the basis on which safety-related design decisions were made and any special design features that were incorporated for safety reasons. This information is essential for safety reviews and also for maintenance so that safety features are not inadvertently eliminated because the reasons they were included are not known. It can also be useful in writing operating and maintenance procedures, safety manuals, and training manuals.

The various hazard analyses used in a program generate hazard analysis reports on the procedures used and the results. These reports usually are combined into one safety assessment report that integrates all system, hardware, and software analyses.

A final safety assessment report may be produced which is used (1) to determine compliance with the program safety requirements and (2) to provide system users and operators with a comprehensive description of the system hazards and the hazardous subsystems and operations associated with the system and (3) to provide reviews with a complete evidence of safety determination. This report might contain some or all of the following:

- General or detailed system and subsystem descriptions and operating characteristics
- Documentation on each hazard-including potential causes, implemented controls, results of the verification activities, close-out status, and any waivers or deviations from requirements
- Risk assessments
- Summaries of all hazard analysis and verification activities-including a description of the methods used, the sources of any basic data used, and simplifications and assumptions made in the analysis and their potential influence on the results
- Safety-related design or operating limits
- Hazardous materials
- Contingency/emergency procedures
- Incident/accident record-a record of all safety-related failures or incidents during development, previous use, or other programs using similar hardware or software, along with all corrective action taken to prevent recurrence.

Figure 2.2 is one of the software hazard reporting forms of LMR.

LMR software hazard reporting form

System _____ Hazard Level _____

Subsystem _____ Date _____

Operation/Phase _____ Closure _____

Hazard

Possible Effects

Hazard Controls	Status	Reference
Verification methods	**Status**	**Reference**

Remarks

Closure Concurrence

Software Safety Engineer _____ Date _____

System Safety Engineer _____ Date _____

Safety Director _____ Date _____

Fig. 2.2 LMR Software Hazard Reporting Form

3. SOFTWARE SAFETY ACTIVITIES

3.1 Tasks and Methods

Software hazard analysis is performed within the context of the overall system design. It addresses aspects of the system design that contribute to the system's ability to perform assigned tasks derived from the plant's safety mission as well as aspects derived from the plant's primary mission that could be detrimental to the plant's safety mission. Consequently, those performing the software hazard analysis must understand the role of the software in the performance of the system safety functions and system control and monitoring functions. The effect of the software acting within the system with respect to its potential impact on the accomplishment of the plant's safety mission must be understood. This understanding is obtained from the system safety analysis; in particular, the system's hazard analysis.

This guideline does not discuss methods or techniques for performing the recommended hazard analyses. Many different hazard analysis techniques have been proposed and are used, but all have serious limitations and only a few are useful for software. But whether these techniques or more ad hoc techniques are used, the software behaviours that can contribute to system hazards must be identified. The possible methods or techniques are Checklists, Hazard Indices, Fault Tree Analysis(FTA), Management Oversight and Risk Tree Analysis(MORT), Event Tree Analysis(ETA), Cause-Consequence Analysis(CCA), Hazard and Operability Analysis(HAZOP), Interface Analysis, Failure Mode and Effects Analysis(FMEA), Failure Modes, Effect, and Criticality Analysis(FMECA), Fault Hazard Analysis(FHA), and State Machine Hazard Analysis. There are also many Human Error Analysis Techniques and other possibilities. FTA is the most popular technique for software hazard analysis not in the requirements phase but in code level. Most of these techniques has mutual aid property. This guideline does not prescribe specific methods and techniques.

The software hazard analysis described could require a significant effort when applied to the digital computer-based I&C system for modern reactor control and protection systems or another process I&C system whose failure could result in significant adverse public, environmental, or financial consequences. It must be recognised that in reality, software hazards analysis is only one of several activities necessary for the development of software to be used in safety-critical applications. Principal activities in this regard include configuration management, verification and validation, and quality assurance activities.

One can consider where software hazards analysis offers a unique capability to improve the integrity of safety-critical software. A major impact of the results from the software hazards analysis is on changes to the software requirements specification for the purpose of eliminating identified hazards that are affected by the software or that are not adequately managed by the software. Another

major impact of these results is on the software architecture, in particular the addition of software architectural features that improve the management of hazards through the concept of defence-in-depth.

The impact of software hazards analysis on the software design specification, with the exception of the use of potentially complex operations associated with data flow and control flow, is overshadowed by the need to address concerns related to correctness through the traceability and V&V aspects. The emphasis on correctness is even more true for the software code.

In conclusion, limiting the bulk of the software hazards investigation to the software requirements specification and the software architectural design and the judicious selection of the events to be assessed should lead to a hazards analysis result that (1) minimizes the probability of occurrence of those hazards with the more significant consequences and (2) minimizes the increase in design requirements that could have the potential for an increase in the complexity of the design.

3.2 Prerequisites to Software Hazard Analysis

Considerable work is required before a software hazard analysis process can begin. The following list of activities will generally require some modifications to fit specific projects. Since iterations of analyses are necessary as the software development proceeds, no strict chronology is implied. For example, a Preliminary Hazard Analysis is needed before a Software Requirements Hazard Analysis can take place. However, the results of that analysis or some other requirements analysis might result in a system design change, which in turn might require modifications to the Preliminary Hazard Analysis.

Each of the prerequisite activities should result in one or more documents. These will be required in order to perform the various software hazard analyses.

1) Prepare a Preliminary Hazard List (PHL) for the application system. This will contain a list of all identified hazards, and will generally be based on the reactor Safety Analysis Report and the list of Postulated Initiating Events (PIE).
2) Prepare a Preliminary Hazard Analysis (PHA) for the application system and subsystems which have impact on, or are affected by, the software. This evaluates each of the hazards contained in the PHL, and should describe the expected impact of the software on each hazard.

   It is recommended that the PHA assign a preliminary severity level to each hazard. The method outlined in IEC 1226 is acceptable. This method assigns a level code of A, B or C to each hazard, where "A" is assigned to the most critical system.
3) Carry out the required hazard investigations and evaluations at the application system and application subsystem level. This should include an evaluation of the impact of software on hazards. There are at least four potential impacts of software on each hazard. These are:

a) The software may challenge the reactor safety systems. Failure of the software to operate correctly has the potential for creating a hazardous condition that must be removed or mitigated by some other system. An example is a software-based reactor control system whose failure may initiate a reactor transient that causes reactor operation to diverge toward an unsafe operating region.

b) The software may be responsible for preventing a hazard from progressing to an incident. Failure of the software to operate correctly has the potential for converting the hazard to an accident. An example is software control of the reactor trip system, where potential failure during an emergency would permit a reactor transient to progress to a significant event.

c) The software may be used to move the system from a hazardous state to a nonhazardous state, where the hazardous state is caused by some portion of the application system other than the software. Software controlling the emergency core cooling systems is an example of this, where decay heat is removed to move a reactor from hot to cold shutdown when other cooling systems are unavailable.

d) The software may be used to mitigate the consequences of an accident. An example is software controlling the containment isolation system, which prevents a radiation release inside the containment structure from escaping and affecting the general public.

4) Assign a consequence level and probability of occurrence to each identified hazard. The former could be based on IEC 1226 while the latter could be based on Mil-Std-882C.

5) For each hazard identified in the PHL, PHA or other hazard analyses, identify its risk level as a function of consequence level and probability of occurrence. These could be qualitative levels such as "high," "medium," and "low."

6) Prepare an application system requirements specification.

7) Create and document a system design, which shows the allocation of safety functions to software components and other system components and shows how the software component and the remaining application system components will co-ordinate to address the hazards discovered in previous analyses.

8) Prepare the remaining documents to the extent required in order to specify, design, implement, verify and analyse the software component of the safety system. This includes analysis of additional hazards introduced by choice of specific digital hardware, computer language, compiler, software architecture, software design techniques, and design rules. This analysis will be revisited as digital system design and software design.

3.3 Requirements Hazard Analysis

Software requirements hazard analysis investigates the impact of the software requirements specification on system hazards. Requirements can generally be divided into sets, each of which addresses some aspect of the software. These sets are termed qualities which are to be considered during software hazard analysis: accuracy, capacity, functionality, reliability, robustness, safety, and security. Some variations may be required to match special situations.

The general intent of software requirements hazard analysis is to examine each quality, and each requirement within the quality, to assess the likely impact on hazards. There are also numerous traditional qualities generally considered necessary to an adequate software requirements specification Completeness, consistency, correctness, traceability, unambiguity and verifiability are, of course, necessary, but should be handled as part of requirements analysis and verification, not as part of hazards analysis. However, software requirements hazard analysis will be hampered if the software requirements specification does not possess these qualities.

3.3.1 Inputs to Software Requirements Hazard Analysis

The following information should be available to perform the requirements hazard analysis.

- Preliminary Hazard List
- Preliminary Hazard Analysis
- Safety Analysis Report
- Safety System Design Description
- Software Requirements Specification

3.3.2 Analysis Procedures

The following steps may be used to carry out the requirements hazard analysis. The steps are meant to help organize the process. Variations in the process, as well as overlap in time among the steps, is to be expected.

1) Identify the hazards for which software is in any way responsible. This identification includes an estimate of the risk associated with each hazard.
2) Identify the software criticality level associated with each hazard and control category.
3) Match each safety-critical requirement in the software requirements specification (SRS) against the system hazards and hazard categories in order to assign a criticality level to each requirement.
4) Analyse each requirement using some guide.
5) Document the results of the analysis.

The information collected during this hazard analysis can be of considerable use later during software development. The combination of criticality level assigned to the various software

requirements provides information that might affect the assignment of resources during further development, verification and testing. It can also suggest the need for redesign of the application system to reduce software-affected hazards.

It is possible that the Software Requirements Hazard Analysis leads to the conclusion that some changes should be made to the system design. For example, it might be discovered that some system requirements assigned to software can be better met through hardware.

It is likely that the hazard analysis will conclude that some requirements do not pose hazards that is, there are no circumstances where failure to satisfy the requirements can cause a hazard. Such requirements probably do not need to be considered in subsequent analyses.

There are many ways to carry out the analysis of step 4. The technique most prominently documented in the literature is Fault Tree Analysis (FTA). Event Tree Analysis (ETA) should also be considered as top events in the tree and expanding the tree to consider consequences. The choice of technique depends on what information is known to the analyst and what information is sought.

3.3.3 Outputs of Software Requirements Hazard Analysis

The products of the requirements hazard analysis consist of the following items:

- A list of software hazards.
- A criticality level for each hazard that can be affected by the software.
- A criticality level for each software requirement.
- An analysis of the impact on hazards of the software when it operates correctly or incorrectly with respect to meeting each requirement.

3.4 Architectural Design Hazard Analysis

Software design hazard analysis is divided here into two sections: one which examines the computer system architecture, and one which examines the detailed software design. The former is discussed first.

A computer system architecture consists of three segments: the hardware architecture, the software architecture and the mapping between them. The hardware architecture describes the various hardware elements: processors, memories, disk drives, display devices and communication lines. The software architecture describes the various software processes, data stores, screen layouts and logical communication paths. The mapping describes how the software will operate on the hardware; this includes identifying which processes will operate on which processors, where the various data stores will be located, where the various screens will be displayed, and how logical communications will take place over physical paths.

Some architectures may introduce complex functions or may have failure modes that other architectures do not have. These represent additional hazards introduced by design choices and which are not identified by previous hazards analyses.

The architectural design documents should contain a two-way trace between requirements and design elements. Each requirement is traced to the design elements that implement that requirement, and each design element is traced back to the requirements which it implements. If this trace does not exist, it should be created before the architecture hazard analysis begins (V&V will also need this trace for part of its activities).

The analysis here builds on the software requirements hazard analysis by extending the latter to the software architecture. A similar analysis is recommended for the hardware architecture and the overall computer system architecture (i.e., hardware, software and mapping).

3.4.1 Inputs to Software Architecture Hazard Analysis

The following information should be available to perform the architecture hazard analysis. Preliminary Hazard List

- Preliminary Hazard Analysis
- Safety Analysis Report
- Software Requirements Specification
- Software Requirements Hazard Analysis
- Requirements to Architecture Trace Matrix
- Software Architecture Description

3.4.2 Analysis Procedures

The following steps may be used to carry out the software architecture hazard analysis.

1) For each software architectural element, determine all the requirements affected by the element. This results from the trace matrix.
2) Assign a risk level to each software architectural element, based on the risk associated with all the requirements affected by the element. The suggested algorithm is as follows:
    - Pick one requirement. Assign the architectural element severity level to be the same as that of the requirement.
    - For each additional requirement, accumulate an architectural element severity estimate by estimating the severity of consequences should all of the identified requirements fail to be met simultaneously.
    - Continue until all requirements affected by the architectural element have been considered. The final architectural element risk level is the design failure probability of the architectural element times the accumulated severity associated with failure.

3) Analyse each safety-critical architectural element.

4) Document the results of the analysis.

The information collected during this analysis can supplement that of the software requirements hazard analysis. In particular, if several architectural elements are classified as very-high-risk, consideration should be given to redesigning the architecture, either to lower the risk associated with the software architecture or to provide compensatory mechanisms to lower overall application system risk. As with the requirements hazard analysis, assignment of resources to further development, verification, and testing can be based on this hazard analysis.

Architecture hazard analysis is likely to demonstrate that some architectural elements are nonhazardous; that is, the analysis shows that no possible failure of the element can affect a system hazard. Such elements require only minimal attention during design and implementation hazard analysis.

If FTA or ETA were used during the requirements hazard analysis, they may be extended to include the software and hardware architectures. The value of the trees comes mostly in the information contained in the structure of the trees. It is not likely to be possible to make a convincing assignment of failure probabilities to architectural elements, so using the tree to attempt to calculate the probability of root events should be used as a reality check and resource allocation tool only.

3.4.3 Outputs of Software Architecture Hazard Analysis

The products of the architecture hazard analysis consist of the following items:

- A list of software architectural design elements with assigned risk level.
- Analysis of the impact on hazards of the software when the specified architecture is used.
- A list of design constraints and coding constraints needed to mitigate hazards associated with the chosen architecture.
- Recommendations for design changes which will reduce the hazard criticality level of software elements.
- Recommendations for increased analysis and testing to be carried out during detailed design V&V, code V&V and final system validation analysis and testing.

3.5 Detailed Design Hazard Analysis

The detailed design documents should contain a two-way trace among the software requirements, the software architecture and the detailed design. Each requirement is traced through the architecture to the detailed design elements that implement the requirement. Each detailed design element is traced back through the architecture to the requirements which it implements. If this trace does not exist, it should be created before this hazard analysis begins.

The primary task here is to see if the detailed design changes any of the results of the requirements or architecture hazard analyses. If the latter have been performed carefully and completely, there should be little more to do. Verification becomes of increasing importance at this point in the life cycle, using the results of the hazard analyses to direct the verification activities.

3.5.1 Inputs to Software Detailed Design Hazard Analysis

The following information should be available to perform the architecture hazard analysis.

- Preliminary Hazard List
- Preliminary Hazard Analysis
- Safety Analysis Report
- Software Requirements Specification
- Software Architecture Description
- Software Detailed Design Description
- Software Requirements and Architecture Hazard Analyses
- Trace Matrix, Requirements to Architecture to Detailed Design

3.5.2 Analysis Procedures

The following steps may be used to carry out the software detailed design hazard analysis.

1) For each software architecture element, prepare a list of detailed design elements which together constitute the architectural element. It may happen that some design elements are used in more than one architectural element. For example, low level communication software may be used by almost every element of the architecture. Device drivers are additional examples.
2) For each design element, determine if the hazards associated with the architecture elements have changed. This may occur if design elements, design rules, design tools, or design techniques introduce common-mode failure mechanisms to two or more architectural elements. If so, previous hazard analyses may need to be redone.
3) Document the results.

If resources do not exist to analyse all design elements, choose those elements that (1) constitute architectural elements of very high or high risk and (2) those elements that occur in many architectural elements. The latter are most likely service elements, such as communications modules, device drivers or file managers.

It should be expected that, in most cases, the analysis will quickly determine that there has been no change to systems hazards due to the detailed design. That is, if a careful job has been done in identifying, controlling and mitigating hazards during the requirements and architecture phrases,

there should be little left to do at the detailed design phase. If this is true, emphasis can start shifting from the global concern of systems hazards to the more local concern of implementation correctness.

The information collected during this analysis can help provide assurance that no new hazards have been introduced by the detailed design. It can also help with the assignment of resources for coding and testing.

3.5.3 Outputs of Software Detailed Design Hazard Analysis

The product of the software detailed design hazard analysis consists of the documented analysis.

3.6 Code Hazard Analysis

The software documents should contain a two-way trace between the detailed design element and the code elements which implement the design elements. If this trace does not exist, it should be created before code hazard analysis begins.

Correctness is much more a concern at this point than hazard analysis, provided that the previous three analyses have been performed well. The main emphasis is on making sure that nothing in the code changes the previous analyses or creates a new hazard. Results of the previous analyses can be used to direct verification and testing resources to the most critical code elements.

3.6.1 Inputs to Software Code Hazard Analysis

The following information should be available to perform the architecture hazard analysis.

- Preliminary Hazard List
- Preliminary Hazard Analysis
- Safety Analysis Report
- Software Requirements Specification
- Software Architecture Description
- Software Detailed Design Description
- Code
- Software Requirements, Architecture and Design Hazard Analyses
- Trace Matrix, Requirements for Architecture to Design to Code Elements

3.6.2 Analysis Procedures

The following steps may be used to carry out the code hazard analysis.

1) For each code element, use the guides to determine if the results of the design hazard analysis need to be modified or if new hazards have been introduced. If so, some or all of the previous analyses may need to be redone. Resources are not likely to exist to analyse all

code elements. Concentrate on those that encode the most risky design elements and those that support basic computing system functions.

2) Examine tools, computer language, and coding techniques for their potential to introduce common-mode failure mechanisms to all modules. Identify coding rules or tool-usage rules that avoid risky tool features or coding techniques. If a pre-existing operating system will be used, identify the risky features or functions that should be avoided.
3) Document the results.

3.6.3 Outputs of Software Code Hazard Analysis

The product of the code hazard analysis consists of the documented analysis.

4. SYSTEM AND SOFTWARE SAFETY ANALYSIS

4.1 System Safety Analysis Techniques

The objective of a safety analysis 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.

4.1.1 FMEA and FMECA

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, 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.

4.1.2 HAZOP

Hazard and Operability Analysis (HAZOP) was developed by Imperial Chemical Industries in England in the early 1960s. HAZOP is based on a system safety theory model of accidents that assumes accidents are caused by deviations from the design or operating intentions. HAZOP is a qualitative technique whose purpose is to identify all possible deviations from the design's expected operation and all hazards associated with these deviations.

HAZOP does not attempt to provide quantitative results, but instead systematizes a qualitative approach. The strength of the method lies in its simplicity and ease of application and in the early identification of design problems. Although HAZOP is closely connected with the chemical industry, the basic idea could be adapted to other industries. HAZOP has the advantage over checklists of being applicable to new designs and design features.

The drawback of the technique are the time and effort required and the limitations imposed by the search pattern. HAZOP relies very heavily on the judgment of the engineers performing the assessment.

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, delectability, 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 identifies 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 guidewords, and ask 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 these 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.

4.1.2.1 Hazard and Safety Analysis

A HAZOP Study is a hazard identification technique within the field of hazard analysis which, in turn, falls within the broader scope of safety analysis. Hazard analysis comprises those activities within safety analysis which pertain to identifying hazards, determining their causes, and planning their elimination or mitigation. Cost-effective hazard analysis requires an approach which considers the whole design at differing levels of detail and has clear definitions of the scope and method of analysis to be used at each level. A HAZOP Study should be used in conjunction with other safety analysis activities to derive adequate confidence in the safety of a system and to avoid continuing development of designs with potential hazards.

A number of HAZOP Studies carried out over the system life cycle can contribute significantly to achieving the level of confidence required in a hazard identification process. Hazard identification is a process which may be applied at any stage from the feasibility study through to disposal. A HAZOP Study can be carried out at any level of design representation. Given a suitable representation, a HAZOP Study can also be applied to a requirements expression.

4.1.2.2 HAZOP Study Process

A HAZOP Study is a hazard identification process, carried out by a team of optimum size between 5 and 7 members. It is based on examining one or more representations of the system's design. A HAZOP Study must be initiated by someone of appropriate authority who is responsible for defining the scope and objectives of the Study. A HAZOP Study shall examine the possible deviations from design intent of the attributes of the components of the system or the attributes of the interconnections between components.

Typically, a HAZOP Study consists of a number of HAZOP Study Meetings. A HAZOP Study must be planned in advance of its first Study Meeting. Planning includes selection of the Study team, definition of the content and logistics of the Study, and notification of the plans to the Study team.

At each Study Meeting, a design representation is examined systematically to identify what variations from the design intent could occur in the relevant attributes, and then to determine their possible causes and consequences. The procedure for identifying deviations from the design intent shall be facilitated by the application of a number of 'guide words.' Each relevant guide word is applied to each attribute, so that the thorough search for deviations is carried out in a structured manner.

On applying a guide word to an attribute, the team enquire into any possible causes and consequences of the deviation for the system as a whole. Also examined are any mechanisms which aid the detection or indication of any hazards. The results are recorded. When all the relevant guide words have been applied to the given attribute, the other attributes of the interconnection or component under consideration are examined. If it is an interconnection under consideration, all other entities on it are then similarly examined.

All components and their interconnections on the design representation are systematically investigated in the same way. When there are uncertainties which cannot be resolved, questions to be studied after the Meeting may be defined. Similarly, when a hazard is identified, there will be a specific recommendation that safety measures should be taken to eliminate or mitigate it, and follow-up work regarding this may be defined.

The result of the Study Meeting is documentation which identifies the system hazards and their causes and consequences, lists questions to be answered, and recommends follow-up actions. All documentation is agreed by the team and signed off by the Study leader. The success of a Study Meeting depends on having appropriate team members and on the control of the process by the Study leader. A HAZOP Study should end in recommendations, not questions, and it is not complete until the follow-up review of questions raised has been carried out. When the answers to the questions raised are available, it may be necessary to schedule further Meetings, with the same team, to complete the Study.

4.1.4 FTA

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.

Another application of FTA on software were suggested by Leveson and Harvey [Lev83]. Here FTA is applied directly on the code listing, and is more closely related to code verification than actually to PSA. A rather different attempt has been to apply FTA, not on the product, but on the development process of safety related software [Øvs91]. The goal of this application is to find critical events concerning fault introduction, and to relate these events to the development process.

4.2 Software Safety Analysis Techniques

A significant problem of developing software for safety critical systems is how to guarantee that the functional behavior of a developed software system will satisfy the corresponding functional requirements and will not violate the safety requirements for the associated overall system. In order to solve this problem, it is important to analyze thoroughly the safety properties of the overall system, to achieve accurate software functional requirements and to verify properly the implementation of the software.

Software failures alone cannot cause harm, only the interactions between "faulty" software and the rest of the system can do so. Furthermore, correctness and safety are not necessary the same thing. It is possible for a system to be in a state that is incorrect with respect to a functional specification but that still meets defined safety criteria. It is possible to separate the states of a system into four categories: [Lev83b]

a) correct and safe
b) correct and unsafe
c) incorrect and safe
d) incorrect and unsafe

The interface between system and software, requirements analysis, is a key activity in the process of software development for achieving the safety goal of systems. The quality of requirements analysis determines the quality of requirements specifications, which directly affects the quality of the developed system.

It is hard to bound precisely the environment which should be considered in requirements analysis, but it should cover at least those systems which interact directly with the target system. In the case of safety critical systems the environment model should cover sources of threats to the system and other systems or equipment in which hazards could arise due to failure in the target system.

Technically requirements analysis methods need to deal with causality, e.g. 'when this event occurs in the environment the system must perform the following actions', and other properties such as behavior of the system under hardware failure conditions. One of the key differences between 'normal' and safety critical systems is the need to be able to deal with causality in the presence of failure, and this is the reason that techniques such as failure modes effects analysis and fault tree analysis are used at this stage in safety critical systems developments.

Requirements safety is a property of an overall system, which depends on both hardware and software in embedded safety critical systems. A safety requirements can be expressed in terms of knowledge about the possible causes of system safety failure.

We believe that identification of appropriate safety requirements is a prerequisite for any useful safety critical application of formal methods. Therefore, safety analysis methods must be incorporated in the lifecycle of formal methods applications. It is important to realize that formal methods are not alternatives to safety analysis; the latter gets as close to analyzing physical reality as possible, while the former deals in models and abstractions.

Several techniques for safety analysis have been used by industry for decades, and some have attracted great attention in the research community. They include Fault Tree Analysis (FTA), Failure Modes, Effects and Criticality Analysis (FMECA), Failure Propagation and Transformation Notation (FPTN), Hazard and Operability (HAZOP). In Leveson's book, "Safeware" [Lev95], there is an excellent summary on techniques for system safety and computers. In this section we focus on the FTA related techniques only.

4.2.1 Software FMEA

4.2.1.1 Fenelon Approach

Fenelon [Fen93] and colleagues proposed an integrated safety analysis method which consists mainly of Hierarchical FTA (HFTA) and Failure Propagation and Transformation Notation (FPTN). They insist that while Leveson's template-based FTA is a depth-first and bottom-up approach, HFTA is a top-down and breadth-first method although compatible with Leveson's methods. FPTN is a new notation to integrate software FTA and FMECA, and is somewhat analogous to traditional data flow-based design notations, although instead of showing normal data flow between elements in a system, it describes the propagation and transformation of failures. Because existing methods of software FTA are not structured in terms of the failure behavior of the software but in terms of its logical structure, HFTA, with its emphasis on a failure-based approach to structuring the tree, is more suited to FMECA which analyzes the propagation of a single failure mode through a causal analysis of the failure behavior.

4.2.1.2 Halden Approach

At the Halden Project they have investigated two different approaches to software FMECA. One approach has been to apply FMECA to configurable software systems, i.e., systems which 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 [DaP96, Dah97]. 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 remain 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 FMECA 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*)?

4.2.2 Software HAZOP

We include a summary of the recent and current research into software HAZOP. This survey has enabled us to evolve some recommendations and draw together common threads of work.

4.2.2.1 Data Flow Based Approaches

Chudleigh [Chu93] has recently published an account of the application of HAZOP to Data Flow Models. This confirmed the suggestion made by Earthy [Ear92] that the use of DFD models during a HAZOP study was appropriate and could produce useful results. The DFDs provided the team with a systems view rather than a software view of the application under review. Chudleigh suggests that not only was the data flow model "readily understandable by all interested parties," but also that the model provided the "most natural" representation to use for a HAZOP study. Classical CIA guidewords were not directly applicable to DFD models so new set of guidewords shown based around data flow and algorithmic functionality were devised. The resultant method required the appropriate guidewords to be applied to each of the input flows and transformations, down through the hierarchy of the model.

4.2.2.2 BURNS AND PITBLADO

Burns and Pitblado [Bur93] presented a modified 3 stage HAZOP approach.

1. Conventional HAZOP
2. Programmable Electronic Systems HAZOP
3. Human Factors HAZOP

The PES HAZOP focuses on the control aspects of the system. They claim that several traditional views, such as cause and effect charts and ladder logic can be used as an appropriate system representation for a HAZOP study. Their PES HAZOP approach bears strong similarities to classical FMEA. The paper claimed that application on various case-studies has identified "numerous safety and operability problems, and provided possible solutions for most of them," although it is unclear how the PES HAZOP is related to an overall systems-level view of the system.

4.2.2.3 PUMFREY and McDERMID

A modified form of HAZOP is being used at the University of York [McD94] at the software design stage (using the MASCOT notation) to characterize likely failure modes of software

components. This HAZOP-like method is called SHARD (Software Hazard Analysis and Resolution in Design)

The innovative aspect of this work is the derivation of deviations by applying fault classes to flow types. The fault classification used is based upon those of Bondavalli and Simoncini [Bonda] and Shrivastava and Ezhilchelvan [Ezhil]. Currently the classification is based around the following fault taxonomy:

- SERVICE PROVISION: omission, commission
- SERVICE TIMING: early, late
- SERVICE VALUE: coarse incorrect; subtle incorrect

For Mascot they have identified seven basic flow types: Stim (binary signal), Binary (Boolean value), Timed Pulse, Value (scalar), Message, Complex and Compound.

4.2.2.4 CHAZOPS

The CHAZOP (Computer Hazard and Operability) study is a technique for undertaking an assessment of a computer system by investigating the areas where potential plant Hazard and Operability Problems could arise. There is substantial variation in detail of implementations throughout different organizations. However, the HSE report [Andow] appears to capture the essence of the overall approach, which is essentially a two-stage process combining a preliminary analysis early in the design stage with a post-implementation analysis.

4.2.2.5 SHAZOPS

SHAZOPS [ICI88] is basically a systematically applied checklist that can be used to review sequence flow charts, control schematics and high level program source (in this case RTL2). The approach can be divided into 2 stages:

- Stage 1 considers the whole process under review; specifically, discussion is prompted by considering design intent, historical data, compliance to standards and documentation.
- Stage 2 is concerned more with the implementation of the system, its failure behavior and its interaction with the outside world.

4.2.2.6 MoD PES HAZOP GUIDANCE

Defense Standard (DEF STAN) 00-56 of U.K. calls for Hazard and Operability Studies to be carried out on systems and sub-systems. Two feasibility studies have been undertaken with the intent of defining a HAZOP method for Programmable Electronic Systems. The first identifies six key areas [MoD94]:

- Team Structure

- Life-Cycle Issues
- Design Representation
- Parameters
- The choice and interpretation of Guidewords
- Reporting and Recording.

The overall philosophy of these initial recommendations appears to be pragmatic and built upon experience within the field. Much of the discussion reinforces existing HAZOP practices, and the work is aimed towards the production of guidelines that will "extend the previous guides by catering for systems which include PES, but is applicable to all systems." The second group have addressed a similar range of issues to the first. A key concept in their approach is the development of a formal reference model based upon object oriented techniques. The model has two elements:

- Object based definition of system components
- Formal definition of hazardous conditions and unsafe behavior of the system.
- The reference model a similar philosophy to the modeling stage of our work.

Within this paper we concentrate upon the issues that are innovative or controversial and are relevant to our interests, particularly the representation. They indicate that the notation used for system representation should conform to a list of specific attributes:

- well defined
- have an ability to adopt abstraction
- make visible all important issues
- be able to deal with the idiosyncrasies of the domain under study
- expressive yet comprehensive
- verifiable against the system it is modeling.

It is clear that their is no such single notation and that representations will be domain and expertise specific. However, it is interesting to note that although this work is independent of ours the conclusions reached are basically similar.

DEF STAN 00-56 states that a preferred method of conducting Preliminary Hazard Analysis is a HAZOP Study. For systems in the lowest risk class this might be the only HAZOP Study necessary. For systems in a higher risk class, safety analysis continues throughout the life cycle and it is recommended that the HAZOP Study process should be carried out at various stages of the system's life cycle in order to refine and extend the identification of hazards. Appropriate times include: when a high-level design is available, when a detailed design is produced, and when the documented system has been built.

The HAZOP Study results should be reviewed in accordance with DEF STAN 00-56 when a modification to the operational system is proposed or when its environment changes. If there are no existing HAZOP Study results, then a HAZOP Study should be carried out. If the change is likely to introduce new hazards, a new Study should be considered. If no results exist from a previous Study, consideration should be given not only to carrying out a Study on the operational system, but also to whether a Study should be carried out on a high-level system design.

When a high-level system representation exists, the first HAZOP Study should assess those hazards which can be identified at this level. In the context of DEF STAN 00-56, this implies assessing those hazards identified in an earlier preliminary hazard list'. This allows future Studies to check the design of the system against the previously identified hazards and to identify any new hazards which may have been introduced. The results may identify the most critical parts of the system and, hence, help to define the scope of further, more detailed Studies.

It is preferable not to carry out a HAZOP Study at a detailed level unless the higher levels have first been addressed. For some systems under development, there may be significant design detail without earlier safety studies being available. In this case, it is recommended that a HAZOP Study should examine the design in a top-down manner.

4.2.3 Software FTA

Fault Tree Analysis (FTA) is an analytical technique used in the safety analysis of electromechanical systems. An undesired system state is specified, and the system is then analyzed in the context of its environment and operation to find credible sequences of events that can lead to the undesired state. A fault tree thus depicts the logical inter-relationships of basic events that lead to the hazardous event.

FTA has been used for the assessment of system reliability and safety for decades and has been developed into an well-understood, standardized method with wide applications throughout the discipline of safety and reliability engineering. A comprehensive introduction to fault tree analysis is the extensive and authoritative Fault Tree Handbook [Ves81].

Traditional fault tree analysis is a probabilistic method in which potential causes of some failure ("top event") are organized in a tree structure reflecting causality—causality is a crucial notion underlying all safety analysis techniques. High-level events can be caused by various combinations of lower-level events, with the principal logical connectives used in the tree being AND and OR gates, which have meanings analogous to those traditionally used in electronic circuit design. Priority-AND gates, exclusive-OR gates and INHIBIT are also available for use.

Leveson and her colleagues [Lev83] were the first to apply fault trees to the safety analysis of software at the statement level. Software fault trees are derived from the software (programs) based on the semantics of statements (e.g. sequential, conditional and iteration statements). Since the

statements may be either concrete or abstract, Software Fault Tree Analysis (SFTA) can be applied on both software design and code levels. The goal of software fault tree analysis is to show that the logic contained in the software design will not produce system failures, and to determine environmental conditions that could lead to the software causing a safety failure. Unfortunately, the informal nature of the technique is its major weakness because he success of the technique is highly dependently on the ability of the analysts. SFTA is essentially a structured walk-through technique with special emphasis on safety issues rather than correctness ones.

Template-based FTA by Leveson is given for each major construct in a program, and the fault tree for the program (module) produced by composition of these templates. The templates are applied recursively, to give a fault tree for the whole module. As they are applied, the fault tree templates are instantiated, e.g. in the above template the expressions for the conditions would be substituted, and the event for the THEN part would be replaced by the tree for the sequence of statements in the branch. SFTA can go back from a software hazard, through the program, and stop with leaf events which are either "normal events" representing valid program states, or external failure events. If the hardware failure event probabilities are known, then the top event probability can be determined. Note that this does not rely on a statistical analysis of software reliability.

Experience with using the template-based approach to analysis indicates that it can, in the right circumstances, be an effective analysis technique. In an avionics system we studied, only 12 lines out of 2,200 could contribute to a given hazard, and the fault trees effectively showed that the program could not give rise to the hazard. Leveson has quoted positive results on the Darlington reactor protection system [AEC93, Lev95].

In the early papers, Leveson and her colleagues used the OR gate to represent sequential composition of statements in a program. This construct is incorrect, although it does give the right answers in some circumstances. In fact, the semantics of fault trees is closely linked to that of Dijkstra's weakest precondition (*wp*) calculus. One of the practical advantages of SFTA seems to be that it has the rigor of the *wp* calculus, but it is presented in a form that is familiar to safety engineers. The difficulty with sequential composition is thus a major drawback, although the links to *wp* calculus suggest there may be a fruitful area of research in linking formal verification and fault tree analysis. A recent book [Fri95] treats sequential composition in a rather different way, which may effectively address this semantic problem using the concept "program segment prefix," but there is still a challenge. They interpret the semantic of FTA as a Hoare's logic rather than Dijkstra's *wp* calculus.

Clark and McDermid propose a more traditional view of the application of fault trees to software [Cla93]. It is suggested that weakest preconditions are used for program specification and validation, and software fault tree analysis is employed for a system-wide analysis of hazards. The scope of software fault trees can be increased to include, for example, compiler errors, control errors, and

memory errors, as well as logical errors. Thus a more realistic view of the software's role in system hazards can be given.

Hansen and her colleagues have recently developed fault trees into a notation for describing software safety requirements for design specifications [Han94]. Specifications are given in a real-time, interval logic, based on a conventional dynamic systems model with a state changing over time. Fault trees are interpreted as temporal logic formulae giving a cause effect relationship between states. It is shown how such formulae can be used for deriving safety requirements for design components. Similar work on formalization of fault trees is also described in [Gor94].

Because software safety can be analyzed from the relationship between a logical fault of software and a physical hazard of a system, the software safety process should be a subset of the system safety process. However, current approaches for analyzing the software safety, originated from system safety techniques, do not provide formal basis of conducting systematic safety analysis of software, in particular, for HRTS software requirements. Software fault tree analysis is the most commonly employed safety analysis technique. Existing software FTA techniques can be grouped as their application phases of the software life-cycle. That is, FTA of software requirements [Han94], FTA of the software design specification [Cha91], and FTA of the software code [Lev83a, Lev87, Cha88, Cla93, Fri95]. However, its industrial practice depends heavily on technical expertise of human analysts, the understandability of the system physics, and is often ad hoc.

4.2.4 Comparison of software safety analysis methods

A safety analysis method should have the following features in order to be a good solution for analyzing the safety;

- *Formality:* In order to have the precision of the analysis, a method should have an appropriate formality. However, a safety analysis method should have a different formality for each phase from system to software development.
- *Cognitive approach:* A safety analysis method should be cognitively balanced in each phase from system to software, and also in requirements, design, and coding phases of the software. It should be easy to use, but precise. For example, there are cognitive approaches such as goal-based approach, causality-based safety analysis.
- *Model-based systematic approach:* A safety analysis method for HRTS software should provide a model-based approach because the safety of software is tightly related with the plant and controller model. It is also preferable to provide a systematic solution such as template-based FTA.
- *Behavioral safety analysis:* Most of the software safety analysis methods based on fault tree analysis are recognized as static method. However, the safety analysis method for

HRTS software should be a dynamic analysis method in order to be able to find the cause of the physical hazard of the system from the behavioral aspects of the software.

- *Integrated approach:* A software safety analysis method should be able to be integrated in all dimensions of Fig. 1.2.

Table 4.1 is the result of the comparison on the existing software fault tree analysis methods according to the above features.

Table 4.1 Comparison of software fault tree analysis methods

Features / Approaches	Formality	Cognitively Balanced	Model-Based Systematic	Behavioral Analysis	Integrated
[Leveson83]	L	H	L	L	L
[Cha88]	M	M	H	L	L
[Clarke93]	H	M	L	L	M
[Hansen94]	H	L	H	H	L
[Gorski95]	H	—	—	—	—
[Fenelon93]	M	H	H	L	H
[Subramanian95]	M	M	H	M	M
[Liu96]	M	M	H	H	M
CRSA	H	H	H	H	H

H: High, M: Medium, L: Low, -: Not applicable

5. SOFTWARE RELIABILITY ASSESSMENT

The goal of the section is to identify measures that can be used to estimate the failure rate of digital instrumentation and control(I&C) systems as part of a nuclear power plant Probabilistic Safety Assessment (PSA). PSA's are used to improve decision making about the safety of nuclear power plants. The most important results from PSAs are not the specific probability numbers generated, but the insights into the risk importance of system features. These insights are used to help focus review on areas of the greatest safety significance and to avoid imposing burdensome requirements with an associated safety benefit. In digital instrumentation and control systems the important risk insights often relate whether or not sufficient redundancy and diversity have been provided in the overall system architecture. These insights are needed early in the design process if they are to have a meaningful impact upon the architecture.

In essence, what is needed at this stage is a reasonably realistic reliability allocation that considers both the combined reliability of the software and hardware reliability as a system. This reliability estimate needs to be conservative enough to avoid setting unreasonable goals for the reliability of the final system, but realistic enough to avoid inappropriately focusing attention on the I&C systems if the largest contributors to risk truly lie elsewhere in the plant. Typically, relatively rough estimates of reliability suffice at the early stages of design. Risk estimates are used in conjunction with sensitivity analyses that determine which of the rough estimates need most to be improved. As design and development progresses, improved reliability estimates may be developed to confirm that the reliability allocations are being met. Ultimately, testing of the completed digital system may be conducted to demonstrate compliance with the reliability allocation commitments and screen out systems that cannot meet minimum reliability needs. Given this context, it is very unlikely that any single measure will be sufficient to provide even a rough number for PSA use. Both process and product measures exist, and both types can be useful. It appears that this study is leading to the combination of a variety of measures of various types and various degrees of objectivity into a single derived measure that can be useful assessing risk importance of digital I&C systems when compared to other plant features.

5.1 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 [I9821]. 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 [Hal77].
  $N= n_1*\log_2(n_1)+ n_2*\log_2(n_2)$
  where n_1 is the number of unique operators and n_2 is the number of unique operands.
  $V=N* \log_2(n_1 + n_2)$
- Prather's complexity [Pra92], which takes into account the number of lines, the number and complexity of conditions and the nesting depth of a program.
- 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.

5.2 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, a 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.

5.3 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.

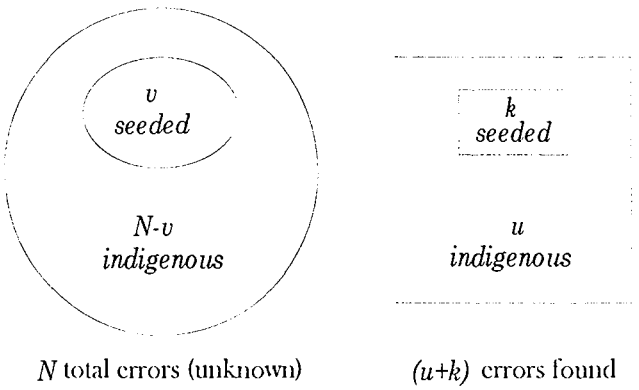


Fig. 5.1 Fault Seeding Metrics

A fairly simple way to derive the above result is to consider Fig. 5.1. 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 are indigenous and k are 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

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

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 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.

5.4 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. They can be classified as:

5.5 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.

5.6 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.

5.7 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.

5.8 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 recompute 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 judgment as a 'belief' in fault freeness. A methodology to use 'influence nets' (also called Bayesian networks or 'belief nets') and engineering judgment to combine evidences from different information sources for a quantitative assessment of this belief has been proposed [Nei96, Del97]. 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 judgment 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 judgment of acceptability.

5.9 Evaluation based on combination of evidences

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. 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 judgement.
- *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 judgement 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 Baye's methodology.

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.

6. GUIDELINES TO PROTECT COMMON MODE FAILURES

6.1 Background

The potential for common-cause or common-mode failures (CCF or CMF) has become an important issue as the software content of digital I&C systems has increased. CCFs are multiple component failures having the same cause. CMFs denote the failure of multiple components in the same way, such as stuck open or fail as-is [NRC97].

The potential for CMFs was not present in earlier analog I&C systems used in operating nuclear power plants because it could usually be assumed that CMF, if it did occur, was due to slow processes such as corrosion or premature wear-out. This assumption is no longer true for systems containing digital software, which are used in advanced reactors. Specifically, digital I&C systems share more data transmission functions and shares more process equipment than their analog counterparts. Redundant trains of digital I&C systems may share databases (software) and process equipment (hardware). With the advent of software operated devices—where multiple redundant units would all be executing the same program with essentially the same inputs and outputs and more-or-less synchronous—the possibility of simultaneous failure in redundant units becomes all too real. This was attributed to the issue of common software being operated in all units of a redundant system where a bug in one would be a bug in all—when a software error exists, all will execute a bug more-or-less simultaneously, and if this error causes the system to operate improperly or crash, each of the redundant machines will fail simultaneously [Wym97]. A study on failures of digital I&C systems in U.S. nuclear power plants from 1990 through 1993 shows that software errors was one of the primary causes of failures in these systems.

To overcome the software CMF, the possibility of including diversely produced programs in the software redundancy has been investigated scientifically at various institutions around the world. The idea of using diverse programs made by independent programming teams to enhance the reliability and safety of computer systems was introduced in the early 1970s. The concept was suggested by different people under different names, fault-tolerant programming, parallel programming, distinct software, redundant programming, etc. One conclusion from these early experiments is that the use of diverse software clearly increases software reliability. There are, however, various aspects of this method that have required further investigations. One is possibility of common mode errors in the diverse programs. A particular source of common mode errors is the common specification. 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 purpose of this section is to provide guidelines for analyzing computer-based nuclear reactor protection systems that discovers and identifies design *vulnerabilities* to the CMF. A special form of

CMF analysis called "defense-in-depth and diversity" (D-in-D&D) analysis has been developed to identify possible CMF vulnerabilities in digital systems. Defense-in-depth is the concept of multiple lines of defense against a perceived threat so that if one line of defense is penetrated, another line is invoked to limit the damage caused by the penetration. This can be carried through several levels. Diversity is the notion that if redundant systems are different in some substantial way from each other, a failure in one will not necessarily imply a failure in the other.

Diversity and defense-in-depth analyses should be performed when a credible potential exists for the CMF. This is presently the case for computer-based safety systems and would be the case for new-technology safety systems whose reliability properties are imperfectly known. The analysis technique can be used to demonstrate adequate diversity and defense-in-depth, or used as a constructive design technique to add diverse protection schemes or equipment to counteract common-mode failure vulnerabilities. The CMF analysis activity involves three major activities:

- Construction of a system model that describes what portions of the design have a potential for CMF.
- Analysis of the I&C system response to design basis events in combination with assumed CMFs.
- Analysis of the plant and offsite consequences of the combination of design basis events and CMFs.

6.1.1 Software fault tolerance

CMF may occur in the I&C architecture's systems & equipment implementing different lines of defense against the same postulated initiating event (see Appendix A for definition). Software by itself does not have a CMF mode. CMF is related to system failures arising from faults in the functional requirements, system design, or in the software [I60880].

Defense in depth is required to be applied to all safety activities, whether organizational, behavioral or design related, to ensure that there are overlapping defenses so that if a failure should occur in a subsystem, it would be compensated for or corrected in the integral system.

Software fault tolerance has several names and several forms: e.g., fault-tolerant programming, redundant programming and distinct software, concept of recovery blocks, dissimilar programming and dual programming, N-version programming, multi-version software, and software diversity. The main methods for software diversity are recovery block technique (RBT) and N-version programming technique (NVP). The common feature of all approaches is the independent generation of functionally identical program modules; the modules may be executed in parallel or sequentially, either in all cases, or depending on acceptance testing and error detection. N-version programming is that, working from the same software specification, several independent groups of programmers

would write the code to meet the specification, perhaps even using different programming languages. However, this idea has not met with much acceptance for several reasons [Wym97]:

- The cost of maintaining several different sets of software has been daunting.
- Coding errors are not the big problem; errors in the requirements cause much more trouble than coding errors.

Software faults are systematic, not random faults and therefore, the single failure criterion [R1153] cannot be applied to the software design of a system in the same manner as it has been applied for hardware. A means of enhancing the reliability of some systems and reducing the potential for certain CMFs is the use of diversity. *Software diversity* is a software development technique in which two or more functionally identical variants of a program are developed from the same specification by different programmers or programming teams with the intent of providing error detection, increased reliability, additional documentation or reduced probability that programming or compiler errors will influence the end results [Vog94]. As listed in Table 6.1, IEC 60880 [I60880] summarizes possible diverse features of software.

One possible view on the application of software diversity is the inspection of the software development life cycle and the use of software diversity aspects in the different phases such as specification, design, implementation, test and maintenance. In each of these phases, software diversity can be applied, and in some cases independent of the application in the other phases. To achieve software diversity, different approaches are possible, for example the use of independent teams, independent solution space, independent working environment/tools, and independent run time support (such as operating system and compiler).

The rationale for defense against software faults is that any software fault will remain in the system or channel concerned until detected and corrected, and can cause failure if a specific signal trajectory challenges it [I60880]. If two or more systems or channels implementing different lines of defense for the same postulated initiating event contain the fault, and are exposed to specific signal trajectories within a sensitive time period, both (or all) systems or channels can fail which is called a CMF.

Table 6.1 Possible diverse features of software

a) Diverse software features of importance include:
- Functional diversity;
- Different design specifications for the same functional requirements;
- Implementing the functions differently (N-version software) for the same specification

b) Diversity at the system level can include:
- Use of independent systems for different actuation criteria;
- Use of different basic technology, such as computers versus hardwired design;
- Use of different types of computers, hardware modules and major design concepts;
- Use of different classes of computer technique such as programmable logic controllers (PLCs), micorprocessors or minicomputers.

c) The design approach features and problem solutions which enhance diversity include differences of:
- Processing algorithms;
- Data for configuration, calibration and functionality;
- Signal input hardware;
- Hardware interfaces and communications;
- Input sampling processes;
- Time sequences of operations;
- Timing processes;
- Use of historical information, latches and rates of change

d) Differences in design and implementation methods include:
- Languages;
- Compilation systems;
- Software tools;
- Programming techniques;
- System and application software;
- Software structures;
- Different use of the same software modules;
- Data and data structures.

e) Diversity during tests (back to back testing).

f) Diverse aspects of management approach include:
- Two designs following deliberate dissimilar development methods (forced);
- Separation of the design teams;
- Restriction of communication between the teams;
- Formal communication of resolution of ambiguities in requirements or specifications;
- Use of different logic definition processes;
- Use of different staff.

6.1.2 Potential CMF causes and effects

The generic types of failures that lead to CMF include errors in requirements, errors in design, and errors in manufacturing that simultaneously affect multiple channels in a function. Draft Amendment 1 to IEC 60880 [I60880] classifies potential CMF causes and effects as follows:

6.1.2.1 CMF potential

- A potential for a software induced CMF of different systems or between different channels in one system exists if common software or software modules are used. Other common features with CMF potential include, common architecture, algorithms, development methods, tools, implementation methods, staffing and management.
- Requirements that are not properly understood or not correctly transformed can result in faults in the software specification resulting in risks of CMF due to exercising the resulting software fault. Deficiencies in software can be due to incorrect, incomplete, inaccurate or

misunderstood software requirements and software specifications. Design errors leading to software faults can be introduced into diverse programs, due to common human factors such as training, organization, thinking processes and design approaches.

- Another cause of CMF could result from connection of systems to ones with lower quality software.

6.1.2.2 Signal trajectories

The signal trajectories can cause a CMF when they are read:

- by each redundant channel of a system using common software;
- by two systems whose functions are diverse but which use common software.

A software fault may result in a software failure when a specific signal trajectory appears. If this signal trajectory is identical for two or more channels or systems, this may result in a CMF which will jeopardize one or more defense layers when sufficient quality, independence and diversity are not provided.

6.1.2.3 Abnormal conditions and events

Abnormal hardware failures, plant conditions and events can cause unforeseen signal trajectories, unexpected software states, transients or overload conditions that were not covered by the initial requirements or by the software design. Potential events which may cause CMF include:

- common timing signal failure, causing loss of timed actions;
- power supply transients causing software stop or auto-restart;
- plant trips causing communication channels to overload;
- saturation of operator capacity, causing an incorrect action;
- operator demands saturating system capacity during plant trips and transients;
- all automatic controller functions engaged and operating; and
- abnormal conditions during outages and commissioning.

6.1.3 Defense against CMF due to software

The potential for CMF due to software should be considered during the design. If postulated conditions of CMF can be foreseen, design changes and defense features, including software diversity, may be needed for protection against CMF due to software. Draft Amendment 1 to IEC 60880 summarizes defenses against CMF due to software as follows.

1. The basic defense against CMF due to software is the production of software to meet requirements correctly. The extent of coverage of self monitoring features such as for data plausibility, parameter range checking and loop timing, etc. is a further important factor in limiting the potential for CMF due to software.

2. The use of well developed software engineering methods with software tool support for software development and verification can help to reduce the number of human design decisions and so potentially reduce the number of faults in the developed software.
3. If human errors are made before software design starts, they may lead to faults of requirements and potential system failures against which software engineering alone cannot provide a defense. If human errors are made during the software engineering process, they may lead to software faults and potential system failures.
4. Implementation of software diversity should use independent systems with functional diversity. The use of system diversity, diverse software features and diverse design approaches should be considered. Different software specifications (e.g., by use of different specification methods) for different implementations of the same functional requirement shall be used for modules where diversity is being claimed provided they do not jeopardize the functional requirements.

The incorporation of *redundancy* into the system is one way of coping with errors during the operation of the system. If an error affects all redundant units at once, a common cause failure occurs. This is especially possible if the redundant units are identical, and if these units either contain an identical error or are prone to identical malfunction in the event of certain input or environment conditions (e.g., high temperature, high voltage for hardware, values out of range for software).

6.1.4 Examples of common mode sensitivity

According to Draft IEC 61513 [I61513], the following typical CMF situations may exist:

- Case 1: a redundant or distributed system, with all channels using the same software—two or more parts can suffer software CMF (see Fig. 6.1a),
- Case 2: a redundant system with different software in each channel, to the same requirements—incorrect common requirements can result in software CMF (see Fig. 6.1b),
- Case 3: two systems operating the same plant differently (such as automatic initiation or control room logic control)—both can have common system software modules, resulting in software CMF (see Fig. 6.1c),
- Case 4: a distributed system, with different functions in each channel (such as different control loops) —each channel can use common system software, resulting in software CMF (see Fig. 6.1d),
- Case 5: two different systems operating different safety actuation systems for the same basic function (such as reactor shutdown)—both can have common design features (such as voting algorithms) which can suffer software CMF (see Fig. 6.1e).

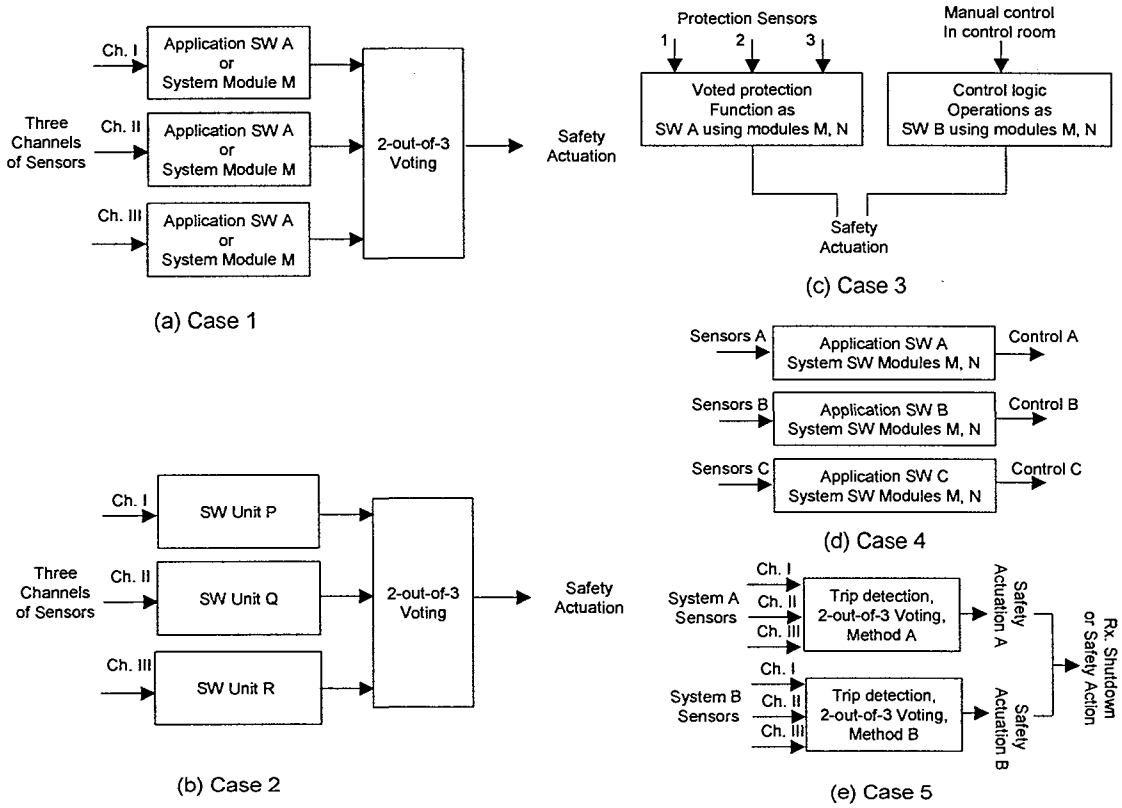


Fig. 6.1 Examples of common mode sensitivity

6.1.5 History of D-in-D&D in nuclear power plants

NUREG-0493 [N0493], "Defense-in-Depth and Diversity Assessment of the RESAR-414 Integrated Protection System," published March 1979, was an assessment of a single reactor protection system that addressed common-mode failure concerns and introduced a method of analysis. In NUREG-0493 defense against common-mode failures was based upon an approach using a specified degree of system separation between echelons of defense. Although the application was specific, the 1979 work established sufficiently general principles that it was adapted to analyze the GE ABWR in 1991, the Westinghouse AP-600 in 1993, and the GE SBWR in 1993 by an independent Nuclear Regulatory Commission (NRC) contractor. ABB Combustion Engineering used the principles themselves in 1992 to analyze their System 80+ protection system.

Subsequently, in SECY 91-292 [SEC91], "Digital Computer Systems for Advanced Light-Water Reactors," the Staff included discussion of its concerns about common-mode failures in digital systems used in nuclear power plants. Some of the major points in that paper are summarized as follows:

1. Common mode failures could defect the redundancy achieved by the hardware architectural structure, and could result in the loss of more than one echelon of defense-in-depth

provided by the monitoring, control, reactor protection, and engineered safety functions performed by the digital I&C systems.

2. The two principal factors for defense against common-mode and common-cause failures are quality and diversity. Maintaining high quality will increase the reliability of both individual components and complete systems. Diversity in assigned functions (for both equipment and human activities) equipment, hardware, and software, can reduce the probability that a common-mode failure will propagate.
3. The staff intends to require some level of diversity, such as a reliable analog backup.

As a result of the reviews of ALWR design certification applications that used digital protection systems, the staff documented an initial statement of a four-point diversity and defense-in-depth requirement with respect to common-mode failures in digital systems and defense-in-depth. This position was documented as Item II.Q in SECY 93-087 [SEC93], "Policy, Technical, and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor (ALWR) Designs," and was subsequently modified in the associated Staff Requirements Memorandum (SRM) [MSE93] dated July 21, 1993. In the SRM, the full Commission approved the modified four-point requirement (see section 6.1.6). Based on experience in the detailed reviews, the NRC staff has established acceptance guidelines for D-in-D&D assessments as a branch technical position, BTP HICB-19 [BTP19].

Experience applying NUREG-0493 to four other vendor's protection systems has led to a clearer picture of how to do the analysis. As computer systems and their developers got more sophisticated, NUREG-0493 needed to be updated to account for this evolution. NUREG-0493 has been rewritten and extended as NUREG/CR-6303 [N6303], "Method for Performing Diversity and Defense-in-Depth Analysis of Reactor Protection Systems," published December 1994, to capture that experience, to explain the techniques for performing the analysis, to remove those details specific to the RESAR-414, and to reflect the technical position of the staff and the Commission. NUREG/CR-6303 is advisory for the assessment of the ALWR designs and the method described is not mandatory. Analyses performed using the methods of NUREG/CR-6303 are not intended to require the inclusion or exclusion of specific failures in a reactor protection system design basis, but are intended to determine points of vulnerability in a design to common-mode failures, should they occur.

6.1.6 The four point regulatory position on D-in-D&D

The US NRC established the following four point position on D-in-D&D for the advanced reactors as a result of the reviews of ALWR design certification applications that used digital protection systems [BTP19, SEC93, MSE93].

1. The applicant/licensee should assess the defense-in-depth and diversity of the proposed instrumentation and control system to demonstrate that vulnerabilities to common-mode

failures have been adequately addressed (Software design errors to be credible common-mode failures that must be specifically included in the evaluation [NRC97]).

2. In performing the assessment, the vendor or applicant/licensee shall analyze each postulated common-mode failure for each event that is evaluated in the accident analysis section of the safety analysis report (SAR) using best-estimate methods. The vendor or applicant/licensee shall demonstrate adequate diversity within the design for each of these events.
3. If a postulated common-mode failure could disable a safety function, then a diverse means, with a documented basis that the diverse means is unlikely to be subject to the same common-mode failure, should be required to perform either the same function or a different function. The diverse or different function may be performed by a non-safety system if the system is of sufficient quality to perform the necessary function under the associated event conditions (Diverse digital or nondigital systems are considered to be acceptable means. Manual actions from the control room are acceptable if time and information are available to the operators. The amount and types of diversity may vary among designs and will be evaluated individually [NRC97].).
4. A set of displays and controls located in the main control room should be provided for manual system-level actuation of critical safety functions and monitoring of parameters that support the safety functions. The displays and controls should be independent and diverse from the safety computer systems identified in items 1 and 3 above.

The position for existing plants is the same except that item 4 is not required. The above position is based on the US NRC concern that software design errors are a credible source of common-mode failures. Software cannot be proven to be error-free, and therefore is considered susceptible to common-mode failures because identical copies of the software are present in redundant channels of safety-related systems. To defend against potential common-mode failures, high quality, defense-in-depth, and diversity are considered to be key elements in digital system design. High-quality software and hardware reduces failure probability. However, despite high quality of design, software errors may still defeat safety functions in redundant, safety-related channels.

Therefore, as set forth in points 1, 2, and 3 above, the staff requires that the vendor/evaluator perform a D-in-D&D assessment of the proposed digital I&C system to demonstrate that vulnerabilities to common-mode failures have been adequately addressed. In this assessment, the vendor/evaluator should analyze design basis events (as identified in the safety analysis report). If a postulated common-mode failure could disable a safety function that is required to respond to the design basis event being analyzed, then a diverse means of effective response (with documented basis) is necessary. The diverse means may be a non-safety system, automatic, or manual if the

system is of sufficient quality to perform the necessary function under the associated event conditions and within the required time.

6.1.7 Applicable regulatory basis and guidelines

6.1.7.1 Regulatory basis

The applicable regulatory basis on D-in-D&D is as follows:

- 10 CFR 50.55a(h), "Protection Systems," requires in part that protection systems satisfy the criteria of ANSI/IEEE Std 279 [I279], "Criteria for Protection Systems for Nuclear Power Generating Stations." IEEE Std 279 includes the following requirements [N6303]:

  4.17, *Manual Initiation.*

  The protection system shall include means for manual initiation.

  4.2, *Single Failure Criterion.* Any single failure within the protection system shall not prevent proper protective action at the system level when required.

  4.6, *Channel Independence*. Channels that provide signals for the same protective functions shall be independent and physically separated.

  4.7.4, *Multiple Failures Resulting From a Credible Single Event.* Where a credible single event can cause a control system action that results in a condition requiring protective action and can concurrently prevent the protective action from those protection system channels designated to provide principal protection against the condition, one of the following must be met.

  4.7.4.1, *Alternate channe*ls, not subject to failure resulting from the same single event, shall be provided to limit the consequences of this event to a value specified by the design bases. In the selection of alternate channels, consideration should be given to (1) channels that sense a set of variables different from the principal channels, (2) channels that use equipment different from that of the principal channels to sense the same variable, and (3) channels that sense a set of variables different from those of the principal protection channels using equipment different from that of the principal protection channels. Both the principal and alternate protection channels shall meet all the requirements of this document.

  4.7.4.2, *Equipment*, not subject to failure caused by the same credible single event, shall be provided to detect the event and limit the consequences to a value specified by the design bases. Such equipment shall meet all the requirements of this document.

- 10 CFR 50.62, "Requirements for Reduction of Risk from Anticipated Transients without Scram," requires in part various diverse methods of responding to ATWS.

- 10 CFR 50 Appendix A, General Design Criteria (GDC) states in part in the introduction that [N6303]:

  The development of these General Design Criteria is not yet complete... (S)ome of the specific design requirements for structures, systems, and components important to safety have not as yet been suitably defined. Their omission does not relieve any applicant from considering these matters in the design of a specific facility and satisfying the necessary safety requirements. These matters include:

  a) ... (2) Consideration of redundancy and diversity requirements for fluid systems important to safety... (T)he minimum acceptable redundancy and diversity of subsystems and components within a subsystem, and the required interconnection and independence of the subsystems have not yet been developed or defined.

  b) ... (4) Consideration of the possibility of systematic, nonrandom, concurrent failures of redundant elements in the design of protection systems and reactivity control systems. ... There will be some water-cooled nuclear power plants for which the General Design Criteria are not sufficient and for which additional criteria must be identified and satisfied in the interest of public safety. In particular, it is expected that additional or different criteria will be needed... for water-cooled nuclear power units of advanced design.

- 10 CFR 50 Appendix A, GDC 21. Protection systems reliability and testability requires in part that, "...no single failure results in the loss of the protection system..."

- 10 CFR 50 Appendix A, GDC 22. Protection system independence requires in part that, "design techniques, such as functional diversity or diversity in component design and principles of operation, shall be used to the extent practical to prevent loss of the protection function."

- 10 CFR 50 Appendix A, GDC 23. Protection system failure modes requires that, "the protection system shall be designed to fail in a safe state or into a state demonstrated to be acceptable on some other defined basis if conditions such as disconnection of the system, loss of energy (e.g., electric power, instrument air) or postulated adverse environments (e.g., extreme heat or cold, fire, pressure, steam, water, and radiation) are experienced."

- 10 CFR 50 Appendix A, GDC 24. Separation of protection and control systems requires in part that, "interconnection of the protection and control systems shall be limited so as to assure that safety is not significantly impaired."

- 10 CFR 50 Appendix A, GDC 29. Protection against anticipated operational occurrences requires that, "the protection and reactivity control systems shall be designed to assure an

extremely high probability of accomplishing their safety functions in the event of anticipated operational occurrences."

6.1.7.2 Relevant guidelines

The applicable relevant guidelines on D-in-D&D is as follows:

- KINS/AR-541, "Development of the Technology for D-I-D and Diversity Regulations of Computer-based Reactor Protection Systems," is the first Korean regulatory guideline on D-in-D&D, which was published from the Korea Institute of Nuclear Safety in April 1998 [KINS98].

- Reg. Guide 1.53, "Application of the Single-Failure Criterion to Nuclear Power Plant Protection Systems," clarifies the application of the single-failure criterion (GDC 21) and endorses ANSI/IEEE Std 379 [I379], "Standard Application of the Single-Failure Criterion to Nuclear Power Generating Station Safety Systems," providing supplements and an interpretation.

- Reg. Guide 1.153, "Criteria for Power, Instrumentation, and Control Portions of Safety Systems," endorses IEEE Std 603 [I603], "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations," as an alternative to ANSI/IEEE Std 279.

- NUREG-0493, "A Defense-in-Depth and Diversity Assessment of the RESAR-414 Integrated Protection System," is the first formal defense-in-depth and diversity assessment of a reactor protection system, the RESAR-414 [N0493].

- NUREG/CR-6303, "Method for Performing Diversity and Defense-in-Depth Analyses of Reactor Protection Systems," documents several D-in-D&D analyses performed after 1990, and presents a method for performing such analyses [N6303]. The methods and results of D-in-D&D assessments used in ALWR design certification submissions are documented in NUREG/CR-6303. This document describes an acceptable method for performing such assessments.

- The Staff Requirements Memorandum on SECY 93-087 describes the NRC position on defense-in-depth and diversity [MSE93].

6.2 Analysis Guidelines

Modern computer-based systems have become sufficiently complex that details can soon overwhelm the analyst. Dividing the system into blocks is intended to reduce design detail to the abstraction level consistent with the goals of the analysis. Consequently, the failures postulated herein subsume many kinds of similar, individual failures and must be considered group failures

whose inner workings need not be defined precisely. This is typical of the effects of software failures in which many individual failures are capable of producing the same or similar outputs. Attempting to postulate all possible individual software errors is impossible on any relevant human time scale and is unnecessary.

As shown in Fig. 6.2, NUREG/CR-6303 describes fourteen analysis guidelines on the D-in-D&D. In this section, the brief outline of each guideline will be described.

6.2.1 Guideline 1 – Choosing blocks

To conduct a D-in-D&D analysis, components of the system architecture should be defined. A block is the smallest portion of the system under analysis for which it can be credibly assumed that internal failures, including the effects of software errors, will not propagate to other equipment (see section 2.5 of NUREG-0493 [N0493]). The objective of choosing blocks is to reduce the need for detailed examination of internal failure mechanisms while examining system behavior under reasonable assumptions of failure containment. As shown in Fig. 6.3, the different types of blocks are defined in section 3 of NUREG-0493. The basic idea is to aggregate the components and modules of the system into a manageably small number of functional units, or blocks, to systematize the postulation of CMF and the analyses of the consequences of these postulated CMF.

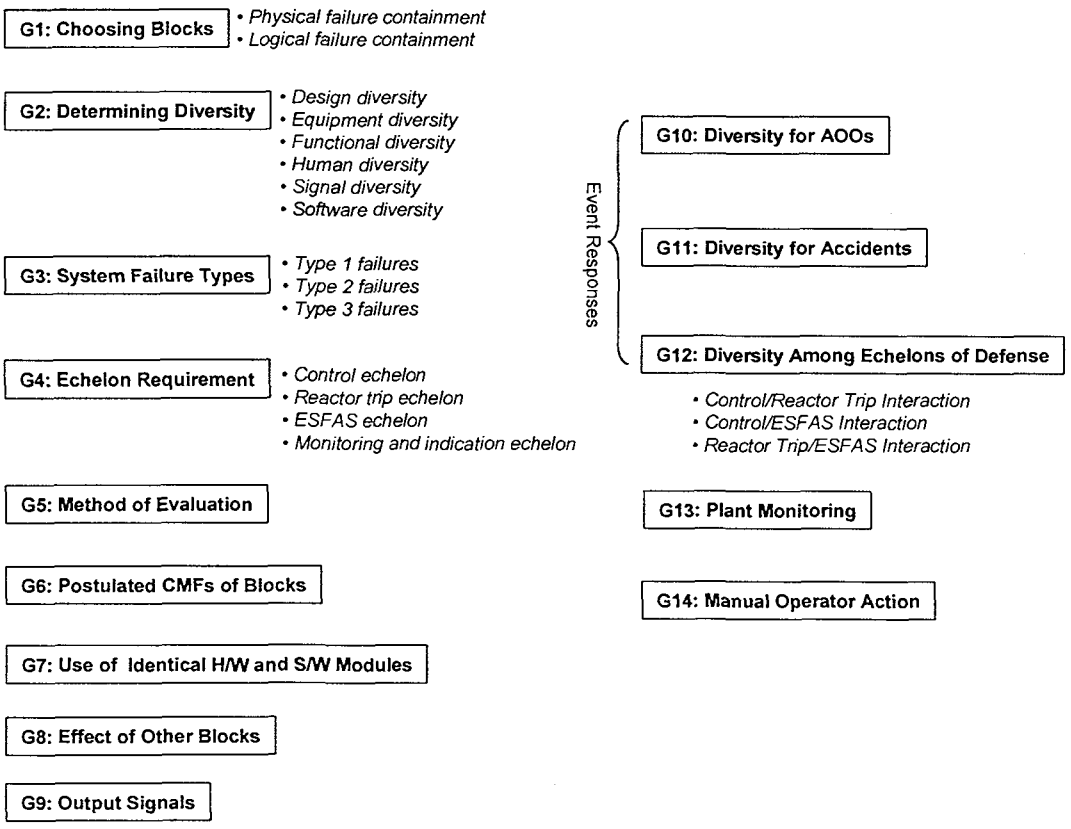


Fig. 6.2 Overall structure of guidelines

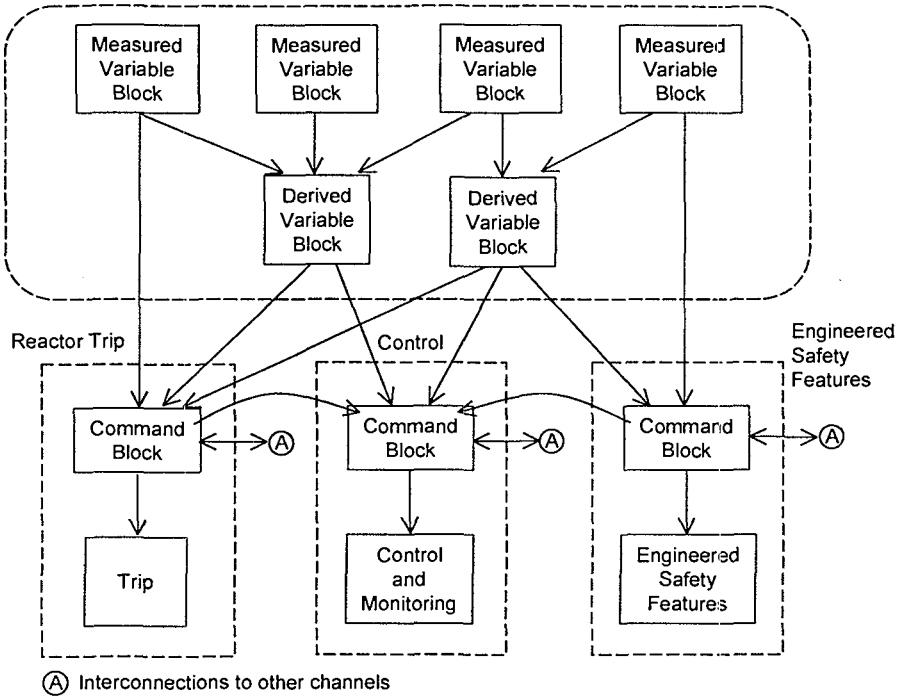


Fig. 6.3 Basic system architecture for evaluation of defense-in-depth principle

Since an objective of this analysis method is to view the subject design at a level of abstraction that reduces the level of detail, the main criterion for selecting blocks is that the actual mechanism of failure inside a block should not be significant to other blocks. Therefore, a block is a physical subset of equipment and software for which it can be credibly assumed that internal failures, including the effects of software errors, will not propagate to other equipment or software.

Failure propagation modes can be divided into two classes: physical (e.g., electrical) and logical (e.g., by corrupted data or corrupted interactions caused by software design faults). In general, physical containment of faults is well understood and consists of (but is not limited to) physical separation, electrical isolation, electrical shielding, and separation of power supplies. Propagation of logical faults (caused, for example, by software design errors), however, is not so well understood. In general, logical faults can propagate by the transmission of data for which the recipient is unprepared, or by failure to transmit data for which a recipient is waiting.

The decision about where to draw block boundaries may hinge upon design commitments made by the applicant about certain equipment interconnections and logical dependencies.

Criteria for determining physical failure containment are:

- Physical separation
- Electrical isolation
- Power supply separation
- Electrical shielding

Criteria for determining containment of logical failures are:

- Given two software modules A and B, if it is physically impossible for a software fault in A to cause module B to fail, then there is sufficient fault isolation between A and B.
- There is no interaction through shared memories.
- There is only unidirectional communication (no handshaking) with other systems.
- The software continues to work regardless of local area network faults (i.e., the software is impervious to errors transmitted by, or occurring in, networks to which the processor running the software is connected).
- All input data from other systems are qualified before use.

6.2.2 Guideline 2 – Determining diversity

The digital I&C systems should provide three echelons of defense-in-depth: control, trip and ESFAS. These three functional echelons of defense should be sufficiently separated and diverse so that postulated CMF events will not lead to unacceptable consequences.

Diversity cannot be considered in the absence of independence; diverse protection system elements that are not independent are assumed to fail simultaneously through interdependencies. Thus, diversity is not a substitute for, nor should it be proposed instead of the independence required by regulation and by standard. Rather, diversity should be seen as a necessary accessory to independence for increasing system robustness in the face of unidentified common-mode failures.

To determine the degree of diversity between two blocks, subsystems, or items of equipment, each block, subsystem, or item should be assessed with respect to the diversity attributes. A set of recommended criteria is listed below for each attribute. A documented basis for claimed diversity attributes should be assembled, with arguments or supporting data.

After assessing individual diversity attributes between two blocks, subsystems, or items of equipment, the combined assessment should be used to present an argument that the one is either diverse or not diverse from the other. Following the suggested criteria for judging diversity attributes, an example is given for computer-based systems of combining such results to reach a diversity conclusion.

In NUREG/CR-6303, diversity is assumed to be separable into the following six attributes:

i) *Design diversity*—Factors increasing diversity between two designs meeting the same requirements—excluding the effects of human diversity—are listed here in decreasing order of effect:
   - Different technologies (e.g., analog versus digital)
   - Different approaches within a technology (e.g., transformer-coupled AC instrumentation versus DC-coupled instrumentation)
   - Different architecture (i.e., arrangement and connection of components)

ii) *Equipment diversity*—Factors increasing equipment diversity between two groups or items of equipment are listed here in decreasing order of effect:

- Different manufacturers of fundamentally different designs
- Same manufacturer of fundamentally different designs
- Different manufacturers making the same design
- Different versions of the same design

iii) *Functional diversity*—Factors increasing functional diversity between two independent subsystems are listed here in decreasing order of effect:

- Different underlying mechanism (e.g., gravity convection versus pumped flow, rod insertion versus boron poisoning).
- Different purpose, function (e.g., normal rod control versus reactor trip rod insertion), control logic, or actuation means.
- Different response time scale (e.g., a secondary system may react if accident conditions persist for a time).

iv) *Human diversity*—Factors increasing the human diversity of a design in decreasing order of effect are:

- Different design organization (i.e., company).
- Different engineering management team within the same company.
- Different designers, engineers, or programmers.
- Different testers, installers, or certification personnel.

Management has the most significant effect on diversity because management controls the resources applied and the corporate culture under which designers, engineers, or programmers work.

v) *Signal diversity*—Factors increasing signal diversity between two signal sources are listed here in decreasing order of effect:

- Different reactor or process parameters sensed by different physical effects (e.g., pressure or neutron flux).
- Different reactor or process parameters sensed by the same physical effect (e.g., pressure versus water level or flow sensed by differential pressure sensors).
- The same reactor or process parameter sensed by a different redundant set of similar sensors (e.g., a set of four redundant water level sensors backed up by an additional set of four redundant water level sensors driving a diverse design of protective equipment).

vi) *Software diversity*—Factors increasing diversity between software designs meeting the same requirements, excluding the effects of human diversity, are listed here in decreasing order of effect:

- Different algorithms, logic, and program architecture
- Different timing, order of execution
- Different operating system
- Different computer language

Another way of expressing these points is that software must differ significantly in parameters, dynamics, and logic to be considered diverse, but only if the "operating system" is sufficiently simple that it can be considered a small set of demand-driven subroutines. Two different safety-critical subsystems that use the same operating system may be subject to CMF through the operating system even if no CMF exists in the safety software. Computer language has little effect on algorithms, logic, architecture, timing, or operating system services.

vii) *Combining Diversity Attributes*—Once an assessment of diversity attributes is made, the results can be combined to make an overall decision or to declare, for instance, that sufficient signal diversity exists. Which diversity attributes assume the greatest importance depends upon the situation.

- The clearest distinction between two candidate subsystems would be design diversity; a non-digital subsystem would easily be considered a diverse alternative to a digital subsystem.
- Between two digital systems (limited design diversity), different computer equipment (equipment diversity) made by different manufacturers (human diversity) would be considered diverse provided there was some functional and signal diversity or some software diversity.
- Some caution is indicated even where there is apparent computer equipment diversity, since program portability is now fairly common and the same software may run on two different computer types.
- In the likely instance of the same developer (limited human diversity) and similar equipment (limited equipment diversity), then software diversity coupled with either functional diversity or signal diversity would probably be necessary to declare that two subsystems were diverse.

In any case, the basis for claiming that a particular combination of diversity attributes constitutes sufficient diversity should be documented.

6.2.3 Guideline 3 – System failure types

Guidelines 5 and 6 (Method of Evaluation/Postulated Common-Mode Failure of Blocks) of NUREG/CR-6303 describe the method for postulating common-mode failures of blocks of the protective system. The system-level effects of these postulated CMFs are described here as three instrumentation system failure types, in order to clarify what the analyst should look for. Note that these failures are not the same as those considered in SAR Chapter 15 analyses.

i) *Type 1 Failures*—Failures of type 1 happen when a plant transient is induced by the instrumentation system for which reactor trip or ESF function is needed, but may not occur, because of an interaction between echelons of defense. Type 1 failures typically begin with a challenge presented by the control system to the reactor trip system or to the ESFAS due to failure of a common sensor or signal source. Defense against such failures depends upon means of accomplishing safety functions that are diverse to the shared signals or equipment (i.e. not impaired by the postulated common-mode failure). Defense-in-depth analysis of type 1 failures is required by general analysis Guideline 12 (Diversity Among Echelons of Defense).

ii) *Type 2 Failures*—Failures of type 2 do not directly cause plant transients but are undetected until environmental effects or physical equipment failure cause a plant transient or design basis accident to which protective equipment may not respond. Failure to respond is due to postulated common-mode failure of redundant protection system divisions or portions thereof. Type 2 failures can have serious consequences only if the event needing safety action occurs while the protection system is in the failed state and before the failure is repaired. Defense against type 2 failures depends upon some combination of diverse *control system, reactor trip system*, ATWS mitigation equipment, *ESFAS,* and functions that are sufficient to mitigate the postulated incident. Defense-in-depth analysis of type 2 failures is required by general analysis Guidelines 10 and 11 (Diversity for Anticipated Operational Occurrences/Diversity for Accidents).

iii) *Type 3 Failures*—Type 3 failures occur because, for some reason the primary sensors expected to respond to a design-basis event instead produce anomalous readings. Since type 3 failures are unpredictable by definition, a strategy dictated by experience is to ensure sufficient signal diversity that alternate means of detecting significant events exist.

- At a minimum, there should be sufficient signal diversity to ensure that for each anticipated operational occurrence in the design basis in conjunction with postulated CMFs, the plant shall be brought to a stable hot standby condition.
- For each accident in the design basis in conjunction with postulated CMFs, the plant response calculated using best-estimate (using realistic assumptions) analyses should

not result in exceeding the 10 CFR 100 dose limits, violation of the integrity of the primary coolant pressure boundary, or violation of the integrity of the containment.

Defense-in-depth analysis that supports signal diversity required for type 3 failures is required by general analysis Guidelines 10 and 11 (Diversity for Anticipated Operational Occurrences/Diversity for Accidents).

6.2.4 Guideline 4 – Echelon requirement

The instrumentation system should provide four echelons of defense-in-depth: *control, reactor trip, engineered safety features* (ESF) actuation, and *monitoring and indicator system.*

i) The *control echelon* is that non-safety equipment which routinely prevents reactor excursions toward unsafe regimes of operation and is used for normal operation of the reactor.

ii) The *reactor trip echelon* is that safety equipment designed to reduce reactivity rapidly in response to an uncontrolled excursion.

iii) The *ESFAS echelon* is that safety equipment that removes heat or otherwise assists in maintaining the integrity of the three physical barriers to radioactive release (cladding, vessel, and containment).

iv) The *monitoring and indication echelon* is that set of sensors, safety parameter displays, and independent manual controls required for intelligent human response to events.

In general, the normal operational hierarchy for transients and accidents is that the second echelon (*reactor trip*) functions when the first (*control*) fails, and the third (*ESFAS*) and fourth (*monitoring and indication*) echelons support the first two. The *monitoring and instrumentation echelon* allows operators to compensate for control system excursions, or, in some cases, for failure of one of the two automatic safety echelons.

6.2.5 Guideline 5 – Method of evaluation

i) The protection system is usually subdivided into redundant divisions, with each division consisting of interconnected blocks. Each block should be considered a "black box," so that any failure required to be postulated within the block fails all output signals.

ii) Block output signals must be assumed to fail in a manner that is credible but that produces the most detrimental consequences when analyzed in accordance with Guideline 9 (Output Signals).

iii) In blocks containing software, it is credible that outputs shall assume values irrespective of inputs because the only logic connecting inputs to outputs is software, and the effects of software failures on outputs are unpredictable.

6.2.6 Guideline 6 – Postulated common-mode failure of blocks

Analysis of defense-in-depth should be performed by postulating concurrent failures of the same block or identical blocks (as defined in Guideline 7) in all redundant divisions. Since several channels may pass through the same block or identical blocks, such common-mode failures have the potential to cause multiple channel failures in a single division, with the same failure replicated across all (four) protection system divisions. The output signals of the blocks thus postulated to fail should do so in accordance with Guideline 5 (Method of Evaluation). In other words, signals entering failed blocks assume the most adverse credible values on output, essentially losing their protective function at that point. Subject to Guidelines 7, 8, and 9 (Use of Identical Hardware and Software Modules/ Effect of Other Blocks/ Output Signals), concurrent failure of each set of identical blocks in all divisions should be postulated in turn (until the list of diverse blocks has been exhausted), and the result of the failure should be documented as a finding of the analysis.

6.2.7 Guideline 7 – Use of identical hardware and software modules

To limit the postulated CMF to a single block in all redundant channels, the likelihood of CMF among different blocks in the same channel should be shown to be acceptably low [N0493]. Blocks are to be considered identical for the purposes of the postulated common-mode failures required in Guideline 6 (Postulated Common-Mode Failure of Blocks) when the likelihood of a CMF affecting them simultaneously is not acceptably low. This means that the probabilities of block failure are not independent and the probability of system failure cannot be calculated by simply multiplying block failure probabilities. Guideline 2 (Determining Diversity) should be used to provide the basis for judging diversity of blocks. For more information, refer to section 3.3.4 of NUREG-0493 [N0493].

6.2.8 Guideline 8 – Effect of other blocks

The analysis should include propagation of the postulated CMF in the single block in each channel via its output signals to all the other blocks influenced by these signals, directly or indirectly [N0493]. During any postulated common-mode failure, signals from failed blocks are propagated to downstream blocks, which react to the possibly erroneous signals. Subject to Guidelines 7 and 9, the other blocks are assumed to function correctly in exact response to all true or false input signals they receive.

6.2.9 Guideline 9 – Output signals

Output signals are assumed to function one-way; that is, failures cannot propagate backwards into an output of a previous block. In cases where a block has more than one output signal, no output signal should be significantly influenced by any credible change or failure of equipment to which any other output signal is connected. This guideline includes any signal transmission paths involving multiported memory, local area networks, serial communication links, or multiplexers. If compliance

with this guideline cannot be demonstrated, block definitions are incorrect and involved blocks should be redefined so that blocks mutually affected through output interconnections are coalesced into one block.

6.2.10 Guideline 10 – Diversity for anticipated operational occurrences

For each anticipated operational occurrence in the design basis which occurs in conjunction with each postulated CMF, the calculated plant response should not exceed a small fraction (10%) of the 10 CFR 100 dose limit or violate the integrity of the primary coolant pressure boundary. This guideline covers instrumentation system CMFs of types 2 and 3 (Guideline 3) for anticipated operational occurrences. A part of the analysis described herein should either (1) demonstrate that sufficient diversity exists to achieve these goals, or (2) identify the vulnerabilities discovered and the corrective actions taken, or (3) identify the vulnerabilities discovered and provide a documented basis that justifies actions not taken (see also the first item of the acceptance criteria in BTP-19, NUREG-0800 [BTP19]).

6.2.11 Guideline 11 – Diversity for accidents

For each limiting fault in the design basis which occurs in conjunction with each postulated CMF, the combined action of all echelons of defense should ensure that equipment provided by the design and required to mitigate the effects of the accident is promptly initiated, supported by necessary auxiliary equipment, and operated for the necessary period of time. This guideline covers instrumentation system CMFs of types 2 and 3 (Guideline 3) for accidents.

The plant response calculated using best-estimate (using realistic assumptions) analyses should not exceed the 10 CFR 100 dose limits, violate the integrity of the primary coolant pressure boundary, or violate the integrity of the containment. A part of the analysis described herein should either (1) demonstrate that sufficient diversity exists to achieve these goals, or (2) identify the vulnerabilities discovered and the corrective actions taken, or (3) identify the vulnerabilities discovered and provide a documented basis that justifies actions not taken (see also the second item of the acceptance criteria in BTP-19, NUREG-0800 [BTP19]).

6.2.12 Guideline 12 – Diversity among echelons of defense

The control system, which includes most instrumentation and control equipment not part of the protection system, is not required to be Class 1E. The plant design basis includes postulated failures, some involving the control system, for which the reactor trip and the ESF actuation systems must provide ample protection. Yet the control system, even though not Class 1E equipment, plays three important roles in defense-in-depth.

1. First, most disturbances are controlled without the need for action by the protection system.

2. Second, failures in the control system may challenge the protection system.
3. Third, during an incident in which one of the protection system echelons (*reactor trip* or *ESFAS*) is incapacitated by a CMF, the control system may mitigate the disturbance.

The control system and the protection system should not be disabled by the same single failure. This concern is stated by GDC 24 of 10 CFR 50 Appendix A (separation of protection and control systems), the IEEE 279 requirement for no interaction between protection and control due to single random failures, and also by the IEEE 379 inclusion of identifiable cascaded failures within the definition of single failures.

Diversity between echelons is therefore necessary and is a concern of this analysis. The instrumentation and control system should be examined for potential interactions between the four echelons of defense, the *control system*, the *reactor trip system*, the *ESFAS*, and the *monitoring and indicator system*, with the intention of determining that the *functions* of at least two out of the four echelons of defense are unimpaired by interconnections.

According to NUREG/CR-6303, potential interactions (see Fig. 6.4) between the nominal echelons are considered as follow (see also the third item of the acceptance criteria in BTP HICB-19, NUREG-0800 [BTP19]).

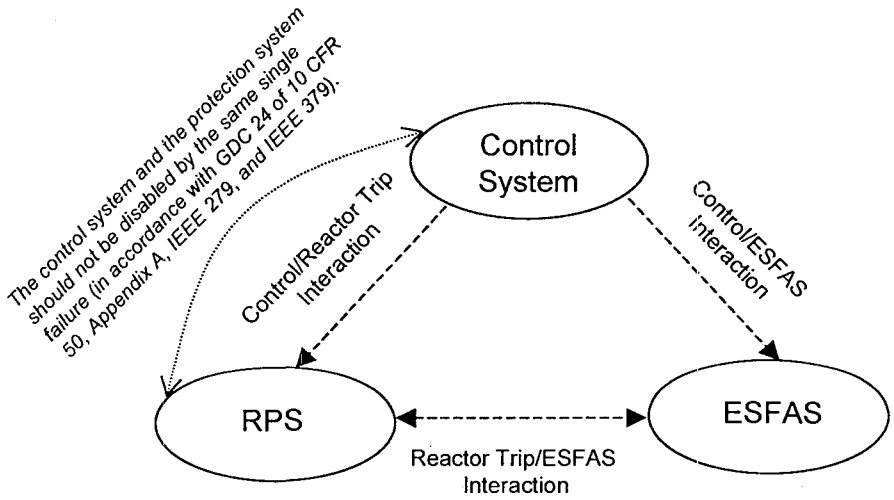


Fig. 6.4 Echelon diagram showing possible interactions

i) *Control/Reactor Trip Interaction*—When a CMF of a common element or signal source shared between the *control system* and the *reactor trip system* is postulated according to Guidelines 5 through 9, and (1) this CMF results in a plant response that requires reactor trip and (2) the CMF also impairs the trip function, then diverse means, which are not subject to or failed by the postulated CMF, should be provided to ensure that the plant response calculated using best-estimate (using realistic assumptions) analyses should not exceed a small fraction (10%) of the 10 CFR 100 dose limit or violate the integrity of the

primary coolant pressure boundary. The diverse means may include manual action if the conditions of Guideline 14 (Manual Operator Action) are met.

ii) *Control/ESFAS Interaction*—When a CMF of a common element or signal source shared between the *control system* and the *ESFAS* is postulated according to Guidelines 5 through 9, and (1) this CMF results in a plant response that requires ESF and (2) the CMF also impairs the ESF function, then diverse means, which are not subject to or failed by the postulated CMF, should be provided to effect the ESF function and to ensure that the plant response calculated using best-estimate (using realistic assumptions) analyses should not exceed a small fraction (10%) of the 10 CFR 100 dose limit or violate the integrity of the primary coolant pressure boundary. The diverse means may include manual action if the conditions of Guideline 14 (Manual Operator Action) are met.

iii) *Reactor Trip/ESFAS Interaction*—Interconnections between reactor trip and ESFAS (for interlocks providing for (1) reactor trip if certain ESFs are initiated, (2) ESF initiation when a reactor trip occurs, or (3) operating bypass functions) are permitted provided it can be demonstrated that functions required by the ATWS rule (10 CFR 50.62) are not impaired under the constraints of Guidelines 8 and 9 (Effects of Other Blocks/Output Signals).

6.2.13 Guideline 13 – Plant monitoring

Signals may be transmitted from the *reactor trip* and the *ESFAS* to the control system or other display systems for plant monitoring purposes provided that all guidelines are met (with special attention to Guidelines 8 and 9 (Effect of Other Blocks/ Output Signals) and the independence required by regulations and standards is maintained (GDC 24—10 CFR 50 Appendix A, IEEE 279 [I279], IEEE 603 [I603], IEEE 379 [I379], and IEEE 384 [I384]).

Connections and software used for plant monitoring and for surveillance of the *reactor trip* and *ESF* actuation systems should not significantly reduce the reliability of or increase the complexity of these systems. No failure of monitoring or display systems should influence the functioning of the reactor trip system or the ESFAS (see also the fourth item of the acceptance criteria in BTP HICB-19, NUREG-0800 [BTP19]). A part of the analysis described herein should address the possibility that failure of the plant monitoring system may induce operators to attempt to operate the plant outside safety limits or in violation of the limiting conditions of operation. The analysis should demonstrate that such operator-induced transients will be compensated by protection system function, or a basis should be documented for claiming that the identified operator-induced transients are either not credible or result in no damage.

6.2.14 Guideline 14 – Manual operator action

The fourth point of the NRC position (see section 6.1.6, item 4) requires that independent and diverse displays and manual controls be available so that operators can initiate a system-level actuation of critical safety functions. To verify this, the analysis should identify the critical safety functions, identify variables necessary for operator decisions using Regulatory Guide 1.97 [R197] for guidance, and demonstrate that the required sensor channels, displays, and manual controls are diverse and independent from the other three echelons (*control*, *reactor trip*, and *ESFAS*). In addition, manual operator action is permissible as a diverse means of response to postulated CMFs if the following criteria are met:

- The postulated CMF and its effects do not impair any related aspect of the manual action, including information displayed that is necessary for operator action.
- Sufficient information is available to the operator.
- Sufficient time is available for operator analysis, decision, and action.
- Sufficient information and time is available for the operator to detect, analyze, and correct reasonably probable errors of operator function.

6.3 Analysis Procedure

This section describes the typical sequence of activities by the vendor for developing a D-in-D&D analysis. Most of activities will be described based on NUREG/CR-6303 [N6303]. Figure 6.5 shows a brief D-in-D&D analysis procedure. Other analysis sequences are acceptable. In practice, the analyses may be iterative in nature with the analysis being refined as information is gained and as issues are identified.

6.3.1 Develop I&C system block diagram

Typically a D-in-D&D analysis begins with the development of an overall I&C system block diagram that shows the protection systems and the other systems that may be credited as diverse mitigation systems. The block diagram should depict the four echelons of defense in depth as described in Guideline 4 (see section 6.2.4) of NUREG/CR-6303. The block diagrams should include the operator displays and controls credited to meet the criteria in Guidelines 13 and 14 (see sections 6.2.13 and 6.2.14) of NUREG/CR-6303.

Blocks within a system should be selected consistent with Guideline 1 of NUREG/CR-6303 such that it may be credibly assumed that failures internal to each block will not propagate (either physically or via common design errors) to other blocks. Blocks may be selected as the smallest portion of the system that meet this criterion, but the selection of larger blocks is acceptable. For example, some analyses have made the simplifying assumption that the entire protection system fails

and have demonstrated that plant consequences are acceptably mitigated by diverse systems. More often, however, a more detailed block diagram of the I&C system is needed to support the analysis.

The block diagrams should show all communication connections between blocks. The communication connections may be depicted very simply, but the analyst must understand what kinds of information can propagate along each connection. Figure 6.8 in section 6.4.7.2 shows an example of typical block diagrams.

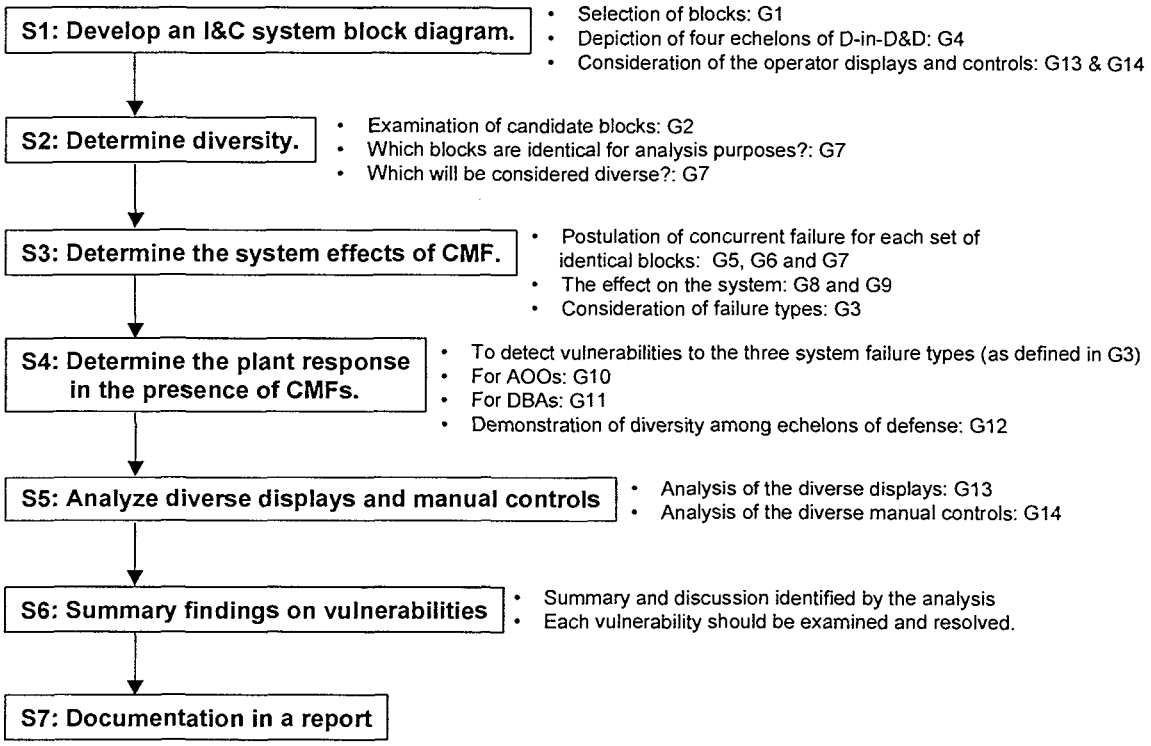


Fig. 6.5 A brief D-in-D&D analysis procedure

6.3.2 Determine diversity

Each system, subsystem, or block should be assessed to determine if it can be credited as diverse from other elements in the block diagram. Guideline 2 should be followed in determining diversity and a summary should be prepared describing which groups of blocks are considered vulnerable to common mode failure, groups of elements are not subject to common mode failure, and the reasons behind these judgments. One way of performing this analysis is to describe each block and discuss the reasons why there is or is not a potential for common mode failure affecting both the selected block and other blocks in the diagram. This analysis may sometimes be done at a higher level than a block. An example would be an ATWS (Anticipated Transient Without Scram) mitigation system that is implemented in a way that the whole systems can be judged to be diverse from the protection system using Guideline 2. In this case, the diversity may be established for the entire system without needing to address each block of the ATWS system individually. In any case, there is no need to make the analysis more detailed than is needed to support the analytical conclusions.

6.3.3 Determine the system effects of common mode failure

For each set of identical blocks in the protection system, concurrent failure should be postulated in accordance with Guidelines 5, 6, and 7. The effect on the protection system should be determined using the criteria of Guidelines 8 and 9. This analysis should consider each type of failure described by Guideline 3.

Once the effect of common mode failure on system elements is understood, the effect on the protection system response to design basis events is examined. Each event analyzed in the plant safety analysis is examined and each set of common mode failures that can affect the assumed response to the event is considered. For every event/common mode failure pair, the remaining operable functions are examined to determine if at least one diverse protection mechanism is available. Tables such as those shown in Fig. 6.7 and Fig. 6.9 can be helpful in organizing this analysis.

6.3.4 Determine the plant response in the presence of common mode failures

The diverse protection mechanisms identified by the analysis of system effects must be shown to limit consequences within the criteria provided in Guidelines 10, 11, and 12. Furthermore, diversity among echelons of defense must be demonstrated in accordance with Guideline 12.

For anticipated operational occurrences as described in Guideline 10 (in combination with primary protection system failure), the goal of defense-in-depth analysis using best-estimate (realistic assumptions) methodology is to show that no more than a small fraction (10%) of the 10 CFR 100 dose limit is exceeded, and that the integrity of the primary coolant pressure boundary is not violated.

For design basis accidents as described in Guideline 11 (in combination with primary protection system failure), the goal of defense-in-depth analysis using best-estimate methodology is to show that any credible failure does not result in exceeding the 10 CFR 100 dose limits, violation of the integrity of the primary coolant pressure boundary, or violation of the integrity of the containment.

Often consequences can be determined by examining the results of existing safety analyses. The assistance of design basis event analysts will be useful in interpreting the meaning of these analyses. In some cases additional analysis of design bases events may be needed to determine the consequences of the event when mitigated by the credited diverse system rather than the primary protections system.

Often there will be some interaction between the analysis of plant response and the determination of system effects. For example, many possible diverse responses may be possible in some cases. If it is difficult to determine plant response when mitigated by one diverse function, or if the selected function does not provide adequate mitigation, another diverse function may be examined. It is not necessary to analyze every possibility; it is only necessary to show that at least one diverse means is

available. Consequently, if the I&C analyst and event analyst can use their plant knowledge to pick successful diverse mitigation strategies at the first time, considerable work can be avoided.

6.3.5 Analyze diverse displays and manual controls

The diverse displays and manual controls provided as a backup to the automatic systems are analyzed for conformance with Guidelines 13 and 14. This should be possible by simple examination of the block diagram.

6.3.6 Summary findings on vulnerabilities

The vulnerabilities identified by the analysis should be summarized and discussed. Figures 6.7 and 6.9 provide example charts that may be used to summarize the findings. Each vulnerability should be examined and dispositioned. Disposition may involve making system modifications to remove the vulnerability or developing a technical justification for accepting the vulnerability.

6.3.7 Documentation in a report

The D-in-D&D analysis should be documented in a report. Section 6.4 (see also sections 5 through 9 of NUREG/CR-6303) describes elements of a recommended report. Ideally, much of the required information should be developed during the analysis and preparation of the final report should only involve assembling the information into final form.

6.4 Preparation of a CMF Analysis Document

This section summarizes the table of contents that should be in the analysis document that should be prepared by a vendor. The description in this section is based on the guidelines of NUREG/CR-6303 [N6303].

The document of a D-in-D&D analysis should explain why and how the analysis was done in sufficient detail that a competent evaluator can identify the underlying bases and assumptions and follow the reasoning to the document's conclusions. Normally an analysis will be presented as a report body, which describes significant features and results, to which is attached one or more appendices which contain the detailed work. The following suggested format, presented in brief outline as shown in Fig. 6.6, is a structure that accomplishes these purposes.

6.4.1 Introduction

Introduction identifies the design being evaluated and those doing the evaluation. Purpose describes the certification or approval for which the evaluation is being performed, or other reasons, if applicable. Background cites relevant regulatory or applicant history and places this particular analysis in historical context. New or unusual features of the analyzed design which may affect the analysis process or outcomes should be noted.

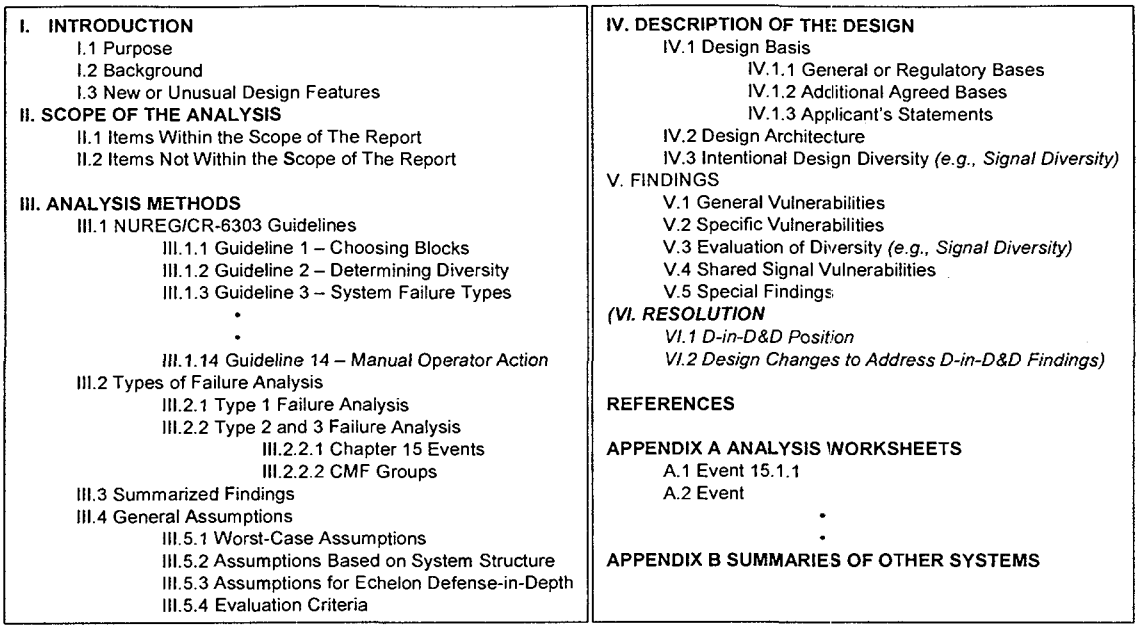


Fig. 6.6 An example of the table of contents for a D-in-D&D analysis document

6.4.2 Scope of the analysis

The scope of the current analysis is important to both analyst and evaluator to ensure appropriate coverage. .

- *What is in scope:* The subsystems and equipment being analyzed should be identified. The types of failure being postulated should be stated. The basis set of anticipated operational occurrences and accidents to be used should be stated or referenced.
- *What is not in scope:* Subsystems and equipment being excluded should be identified and reasons for the exclusion should be stated. Certain failures that are incredible or do not fit the definition of common-mode failure as used in the analysis should be described and reasons given for their exclusion.

6.4.3 Analysis methods

The analysis methods and their derivation from various authorities and guidelines should be described in detail. It is particularly important to discuss deviations from standard methods or assumptions made to clarify missing, incomplete, or inconsistent information provided by design descriptions (which usually accompany a Standard Safety Analysis Report).

Guidelines for performing the D-in-D&D analysis are given in section 6.2 and should be referred to in this section of the analysis document. However, these guidelines do not supplant or supersede the general design criteria of 10 CFR 50, Appendix A, or other standards or design bases required by regulation or practice. Such criteria or standards as are applicable to the design should also be stated here.

The types of failures to be considered in the analysis should be noted here. See section 6.2.3 for further detail.

6.4.3.1 Data required to the analysis

Besides the general guidelines and other background information, a D-in-D&D analysis requires certain specific information, some of which is unique to this analysis process. The following describes the main analysis data inputs.

i) *System Diagram and Logic Diagrams*—A system diagram showing one division (of multiple redundant divisions) of the protective system to be analyzed is required. Additional detailed logic diagrams or textual descriptions may be necessary so that system response to various events can be determined by the analyst. The system diagram is equivalent to a single-line electrical diagram in that it is an abstraction that presents the system architecture at a level of detail appropriate to "block" failure analysis.

ii) *Chapter 15 Events*—In most instances, the SAR Chapter 15 events form the basis set of anticipated operational occurrences and accidents which will be used to challenge the protective system design. The applicant usually presents simulation curves of reactor parameters during the Chapter 15 incidents which at least determine the primary trip variables, but often exclude secondary trips due to postulated prompt protection system action. This is appropriate, considering the goals of Chapter 15 analyses. Chapter 15 analyses are also performed under conservative, rather than best-estimate, assumptions.

iii) *Alternate Trips*—Because the purpose of D-in-D&D is to determine whether sufficient diversity or defense-in-depth exists to compensate for primary trip (or ESF initiation) failure, it is necessary to know which secondary trip or initiation signals will activate defenses, if any, if primary signals or signal paths fail. Sometimes this is possible if trip point values are known and simulation curves of an incident or a closely similar incident show that alternative reactor parameters exceed trip values. When such information exists, a secondary trip will occur under conservative assumptions, although it is sometimes not clear whether, for less severe event sequences, a secondary trip point will be reached. In cases in which secondary trips cannot be clearly determined, it may be necessary to perform simulations that assume the primary trip variable fails. A set of best-estimate (using realistic assumptions) secondary trip sequences for events lacking a clear secondary trip should be deduced or obtained. An alternative to secondary trip data is best-estimate analyses that demonstrate for each such event that all possible sequences lead to safe conclusions.

iv) *Required Mitigation*—Success or failure of protective system action, whether by primary or alternate trip variables, is determined solely by the actuation of, or failure to actuate,

appropriate mitigation measures. For this, the analyst needs to know the mitigation measures required for satisfactory response to the design basis incidents.

6.4.3.2 Assumptions to be stated

The assumptions made by the analyst are crucial to understanding the decisions made during analysis. Statement of the analysis assumptions is important because it permits easier and faster resolution of misunderstandings, should they arise.

i) *Worst-Case Assumptions* —The worst-case assumptions describe the particular application of Guideline 5 (Method of Evaluation) of NUREG/CR-6303 to the subject design. Failures are assumed to occur in the most limiting fashion possible consistent with hardware or software construction. The assumption on failure latency is the worst-case assumption applied to surveillance effectiveness. That is, failures are assumed to be latent and undetectable until stressed by event or accident, at which time the failure becomes manifest.

ii) *Assumptions Based on System Structure*—System structure is the crux of D-in-D&D analysis. The assumptions should describe what portions of the design have potential for common-mode failures and how and why block delineations were made. Guideline 7 (Use of Identical Hardware and Software Module) of NUREG/CR-6303 requires that "identical" blocks in a design be determined. Guideline 2 (Determining Diversity) provides criteria for determining diversity (or similarity) among subsystems, and this section should describe the precise application of Guideline 2 to the instant design so that it is clear which blocks are considered identical. The standard for independence between two subsystems as defined (in Guideline 2) is that they must differ significantly in parameters, dynamics, and logic. If two such subsystems perform similar functions but have different inputs (different parameters being sensed) combined by different logic, it is assumed that the two subsystems do not have a common failure mode. This assumption is reasonable since it is implicit that the programs being run will differ in timing and logic because of the different inputs and processing code.

iii) *Assumptions for Echelon Defense-in-Depth*—The equivalent of the four nominal echelons, *control*, *trip*, *ESFAS*, and *monitoring and indicator* must be identified in a design. Defense-in-depth assumptions describe the arrangement of echelons of defense (which equipment is part of which echelon), necessary mitigations or effectiveness of certain mitigation equipment, and any other assumptions necessary to clarify which combinations of equipment must function during the events studied. In some instances, this may require identifying and categorizing equipment such as that designed to satisfy the ATWS rule, but which crosses nominal echelon boundaries in its effects.

iv) *Evaluation Criteria*—The criteria for success are stated in Guidelines 10 through 12 of NUREG/CR-6303 for event responses and in Guideline 13 for plant monitoring. Any deviation or additional criteria should be stated here. By default, the applicant's list (if such a list exists) of necessary and sufficient mitigation actions for various events is considered sufficient protection system response to fulfill Guidelines 10 through 12.

6.4.4 Description of the design

This section, the assumptions section (see section 6.4.3.2), and a system diagram (see section 6.4.3.1) combine to provide an accurate description of the design to be analyzed. This section should be a high-level text description that, combined with the system diagram, lays out the system architecture and the details of the design that are material to the analysis. The sources from which design information was taken should be cited.

The design being analyzed should be described with emphasis on factors that are important to D-in-D&D. This is not merely a repetition of the applicant's SAR submission, which may be detailed, but is a critical selection from what may be voluminous material of that part that forms the basis for analytic decision making. A design description consists of *three* parts: the *design basis* which any successful design must satisfy, a *description* of the design architecture (probably supported by a number of drawings), and a *description* of intentional diversity in the design.

6.4.4.1 Design basis

Design bases are the rules under which a design is executed and they specify general qualities that the resulting design will satisfy. In cases where it may not be clear from details how to decide a particular analytic question, the design basis may provide guidance sufficient to make the decision. Certain design qualities are mandated by regulation and these are listed below under "General or Regulatory Bases." An applicant may also have agreed or may have volunteered to use certain standards or techniques.

i) *General or regulatory bases*—Design basis requirements pertinent to D-in-D&D for the light water reactor (LWR) designs include the regulations and standards (which is summarized in section 6.1.7.1) as follows:

- 10 CFR 50 Appendix A, "General Design Criteria,"
- 10 CFR 50.55a(h) requires that protection systems meet the requirements of IEEE Std 279 [AI279], and
- IEEE Std 603 [I603] includes criteria substantially similar to the foregoing IEEE Std 279 [I279] requirements, and is endorsed by Regulatory Guide 1.153 [R1153] as an alternative.

ii) *Additional agreed bases*—The applicant may have agreed to use additional standards or conform to regulations or design techniques that are not directly required by the sources mentioned above. These bases should be identified.

iii) *Applicant's statements*—The applicant may have made statements in SAR text that certain standards would be used or that certain design techniques would be used or would be avoided.

6.4.4.2 Design architecture

i) The points of the applicant's design that are salient to defense-in-depth and to the separation of the design into independent, diverse subsystems should be described.

ii) A system block diagram, such as that demonstrated in the Appendix (Block Examples) of NUREG/CR-6303, is extremely helpful here. The echelons of defense should be identified, and any division into redundant, independent divisions should be described. Relations between the echelons, and between the echelons and subsystems such as diverse ATWS mitigation equipment or the remote shutdown panel, should be detailed with attention to aspects important to the analysis.

iii) In parts of the design that use redundancy, the voting scheme should be described, particularly where it may have asymmetries that could be single-failure vulnerabilities.

iv) It should also be noted where the applicant's design commitments are being used to decide significant design issues rather than using applicant-supplied design details.

6.4.4.3 Intentional design diversity

Any specific diversity (e.g., signal diversity) or design features (e.g., a diverse backup system) intended by the applicant to improve protection system performance in the face of CMF should be acknowledged.

6.4.5 Findings

Findings should be presented in a sensible organization that could be used directly by license applicants to reduce discovered vulnerabilities in their reactor protection systems. Certain graphical aids to presenting results, such as analysis chart (see section 6.4.7.1), system block diagrams (see section 6.4.7.2) and vulnerability summary charts (see section 6.4.7.3), are suggested in section 9 of NUREG/CR-6303. Previous analyses have used an organization similar to the following.

6.4.5.1 General vulnerabilities

These are vulnerabilities that appear in a majority of cases studied under Guidelines 10 and 11 (Diversity for AOOs and Accidents) of NUREG/CR-6303. Reducing these would probably be considered a higher priority than reducing isolated, specific vulnerabilities.

6.4.5.2 Specific vulnerabilities

Vulnerabilities found under Guidelines 10 and 11 of NUREG/CR-6303 that occur only during one or a few SAR Chapter 15 events are reported here. These might be considered lower priority than general vulnerabilities, depending upon the event consequences.

6.4.5.3 Evaluation of diversity

This section contains an evaluation of how many events are potentially detected only by one sensor. Since reactor trip and various ESF functions are initiated by different logical combinations of sensor signals, this findings section discusses diversity for all mitigation functions.

6.4.5.4 Shared signals

This section reports the results of the analysis required by Guideline 12 (Diversity among Echelons of Defense).

6.4.5.5 Special findings

Any other findings that a responsible reviewer may have noted during perusal of the design should be reported here.

6.4.6 References

Standards, regulations, and publications used during the preparation of the analysis should be cited in this section.

6.4.7 Appendices

The appendices contain the actual analysis worksheets and narratives. Section 9 of NUREG/CR-6303 describes an analysis worksheet that may be useful for systematic documentation of analysis details.

Graphical aids can be used to enhance the intelligibility of a report and their use is encouraged. Previous workers have found two kinds of charts and a drawing to be particularly useful both in doing the analysis and in presenting the results.

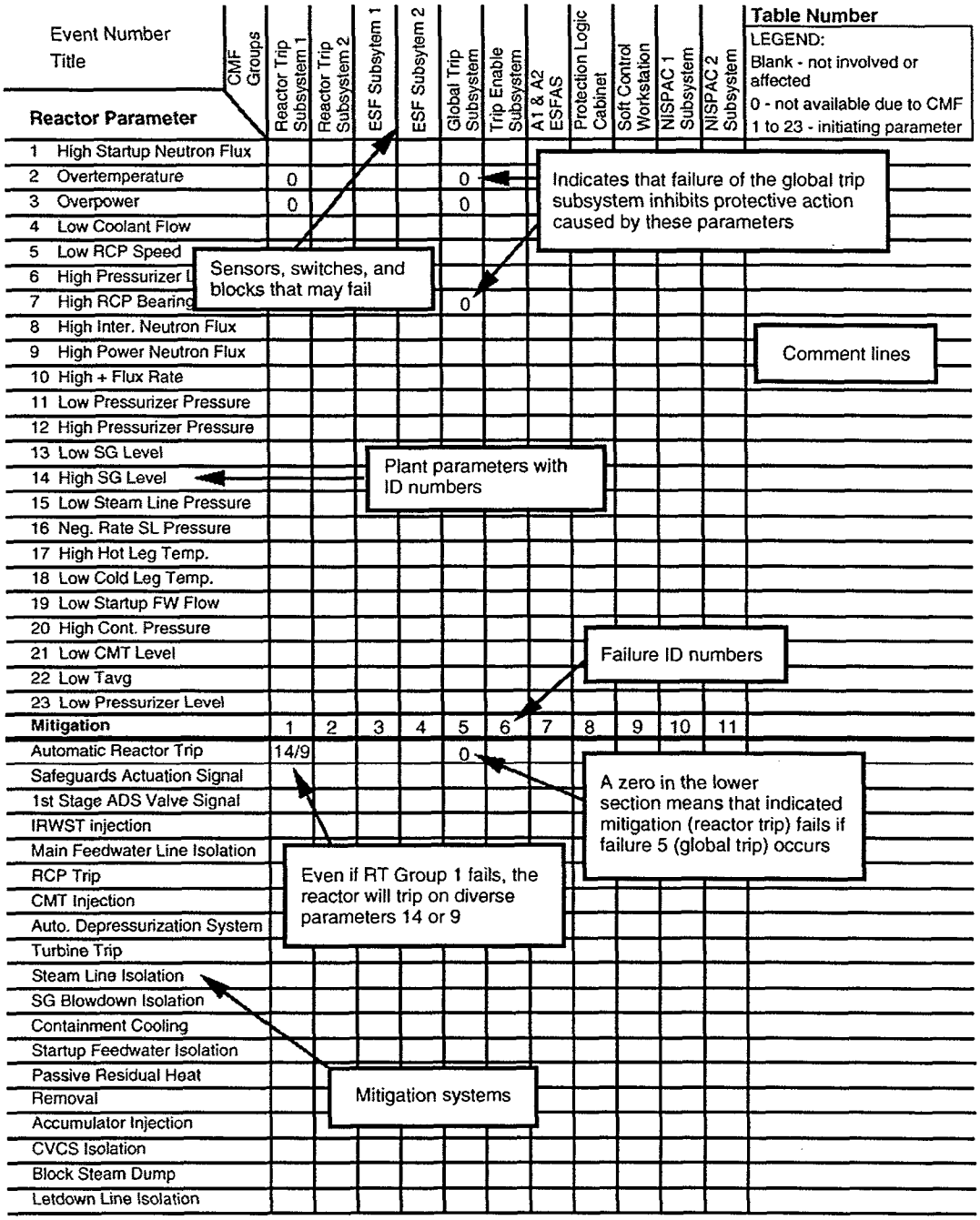
- *Analysis charts* aid the analyst by presenting analytic decisions in a matrix format that permits failure-by-failure determination of system response.

- *The system block diagram* presents the system architecture with only the interconnections of interest to the analyst being displayed.
- *Vulnerability summary charts* show analysis results consolidated in matrix form by design basis event versus signals and blocks.

6.4.7.1 Analysis charts

The illustrative format of analysis charts is shown in Fig. 6.7 (for more information, see section 9 of NUREG/CR-6303). These charts differ in detail, but their common purpose is to record failed signals or blocks ("CMF groups") systematically and to indicate the results of each failure.

One analysis chart is prepared for each Chapter 15 event studied. The top portion of the chart consists of lines labeled with reactor parameters and columns labeled with sensors or blocks. The failure of a sensor or a block will prevent a reactor parameter signal from passing through the sensor or the block, and this is indicated by placing a zero in the appropriate intersection of row and column in the upper half of the chart.



A single common-mode failure and its consequences are represented by a column of the chart. The sensor channel or block that fails is indicated at the top of the column. The failure is indicated by at least one 0 in the column on the upper half of the chart. A consequential failure of a mitigation system is indicated by a 0 in the column on the lower half of the chart, where the mitigation system is at the left of the row. If a number (not 0) appears in the lower half of the chart, it means that the mitigation system of that row will initiate with the plant parameter indicated by the number, despite the CMF of the column.

Fig. 6.7 An illustrative format of analysis charts for a pressurized water reactor

If the column is followed to the lower half of the chart, the mitigation means required for this event and for the CMF represented in this column are marked either with a zero (meaning no diverse initiation) or the number of the reactor parameter that does cause initiation. The chart is marked for each sensor, block, parameter, and mitigation means relevant to the event and then examined for zeros in the lower half.

The lower-half zeros represent a failure to initiate mitigation for the columnar CMF in conjunction with the Chapter 15 event, and if insufficient mitigation is initiated, a vulnerability has been found. Insufficient mitigation exists if the sum of mitigation means with non-zero initiators in a column is less than the applicant's required mitigation for the Chapter 15 event, reduced by the effects of portions of the control system that are postulated to continue operation.

The analysis chart provides a stepwise method of considering common-mode failures and their effects. However, it may be difficult for others to interpret the analyst's work solely from the chart, so it should be accompanied with a short narrative describing the reasoning behind the chart marking.

6.4.7.2 System block diagram

System block diagrams (see Fig. 6.8 for a sample pressurized water reactor (PWR) system block diagram) are contained in the Appendix of NUREG/CR-6303, along with a discussion of how blocks were selected and what level of detail is appropriate.

Arrangement into blocks also aids the analyst's perceptions and presents the significant interconnections graphically. Guidelines 6 through 9 require that input and output connections be determined for each CMF taken in conjunction with SAR Chapter 15 events, and it is tedious and inefficient to have to search through several drawings each time. Also, the system block diagram makes the analyst's view of the system clear to readers of the analysis results, so that analyst misperceptions can be corrected by knowledgeable evaluators.

6.4.7.3 Vulnerability summary charts

There are potentially about 20 or 30 events to analyze, which result in an equal number of analysis charts. Vulnerabilities documented on the analysis charts can be transferred to a vulnerability summary chart (see Fig. 6.9). The example in Fig. 6.9 is from analyses in progress, before resolution of vulnerabilities has occurred, a process beyond the scope of this discussion.

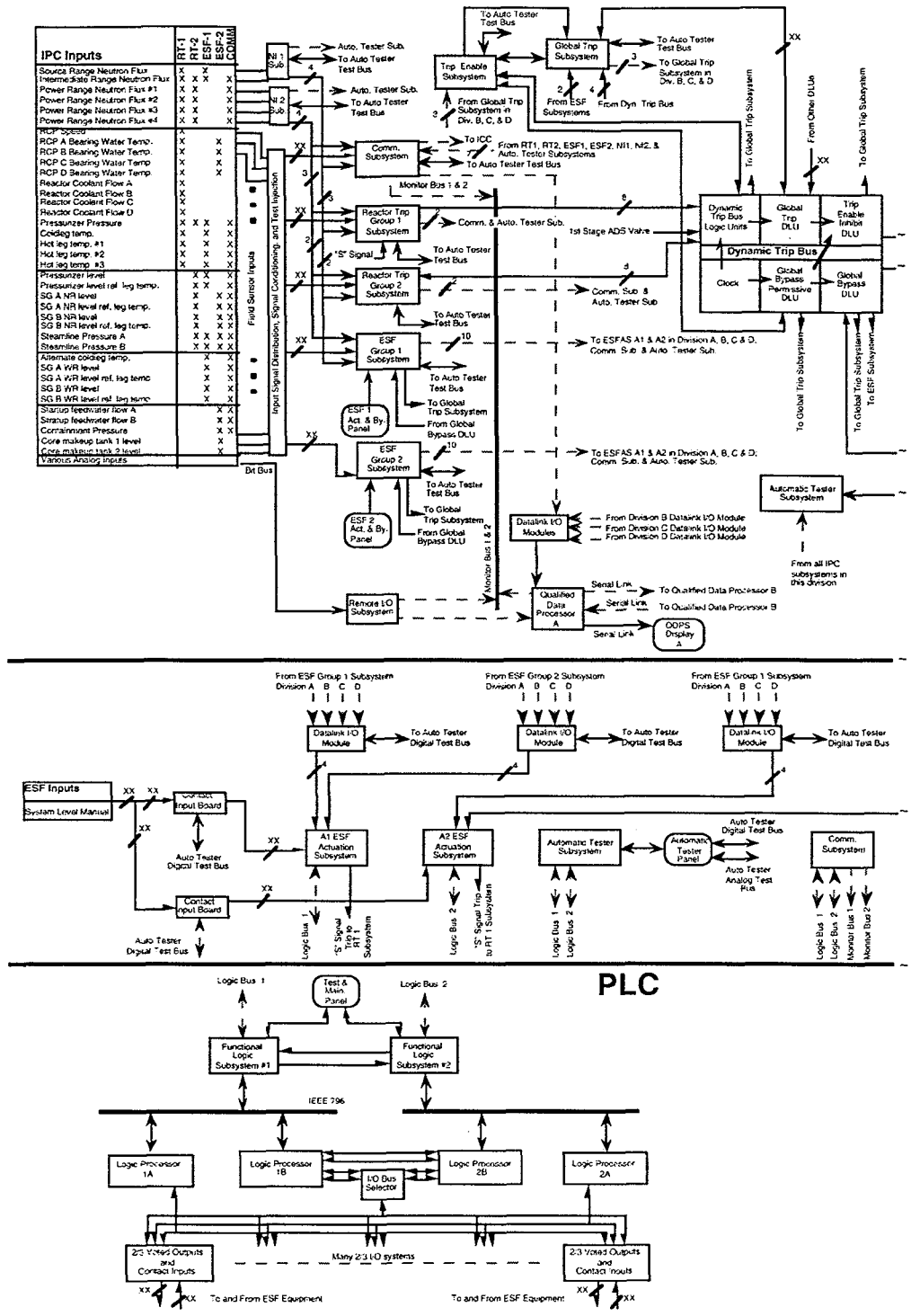


Fig. 6.8 Sample pressurized-water-reactor system block diagram

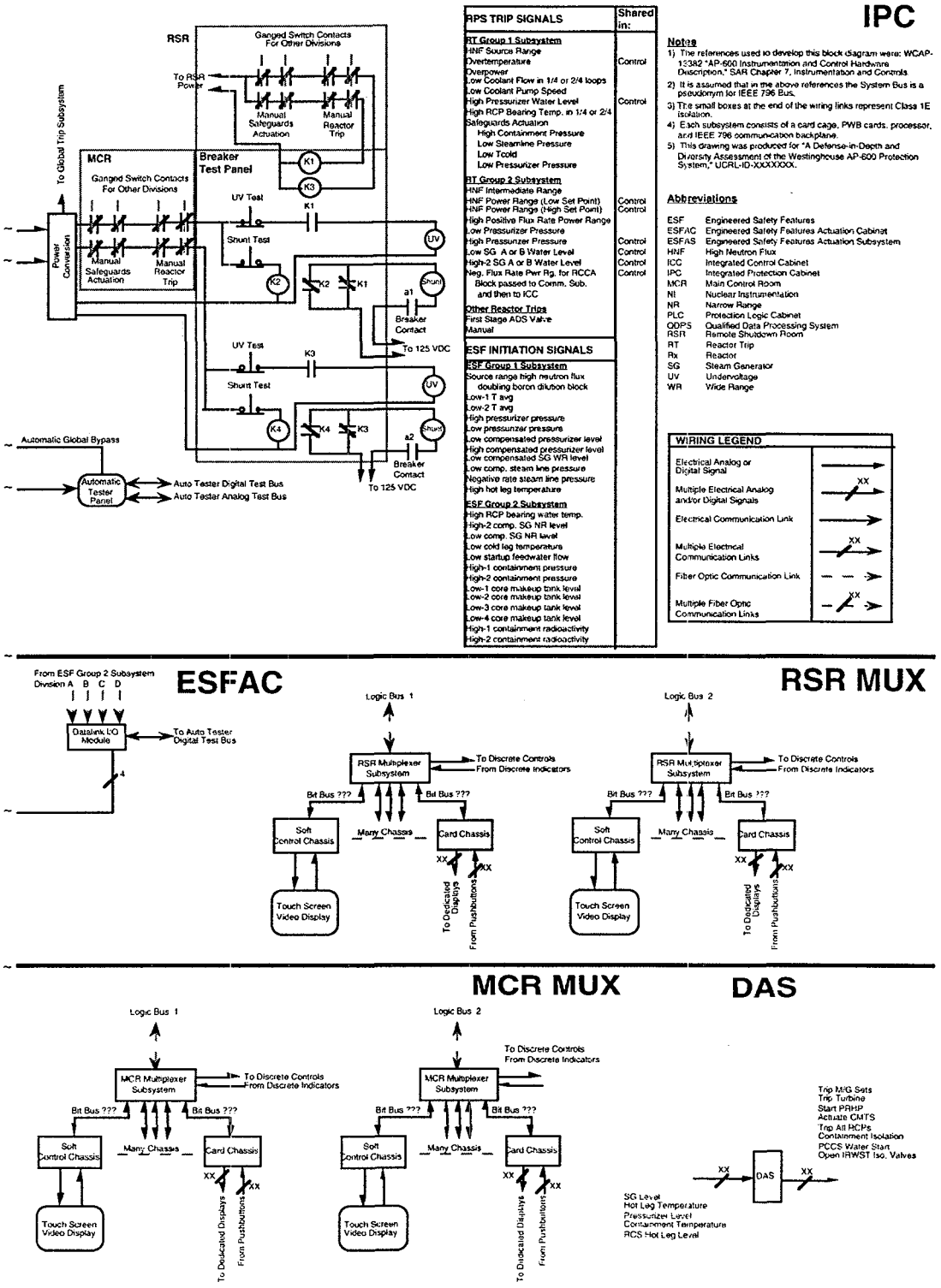


Fig. 6.8 Sample pressurized-water-reactor system block diagram (continued)

CMF Vulnerability Summary Chapter 15 Event	CMF Groups	Reactor Trip Subsystem 1	Reactor Trip Subsystem 2	ESF Subsystem 1	ESF Subsystem 2	Global Trip Subsystem	Trip Enable Subsystem	A1 & A2 ESFAS	Protection Logic Cabinet	Soft Control Workstation	NISPAC 1 Subsystem	NISPAC 2 Subsystem	CMF Vulnerability Summary **Legend:** blank - not involved T - Trip vulnerability E - ESFAS vulnerability
15.1.2 - FW Sys. Malfunction that Results in an Inc. in FW Flow				E	E	T		E	E	E			
15.1.4 - Inadvertent Opening of a SG Relief or Safety Valve					E	T		E	E	E			
15.1.5 Steam System Piping Failure					E	T		E	E	E			
15.1.6 - Inadvertent Operation of the PRHR System				E	E	T		E	E	E			
15.2.2 through 15.2.5 - Turbine Trips						T							
15.2.6 - Loss of AC Power to the Plant Auxillaries				E		T		E	E	E			
15.2.7 - Loss of Normal Feedwater Flow						T		E	E	E			
15.2.8 - Feedwater System Pipe Break				E	E	T		E	E	E			
15.3.1 - Partial Loss of Forced Reactor Coolant Flow						T							
15.3.2 - Complete Loss of Forced Reactor Coolant Flow						T							
15.3.3 RCP Shaft Seizure (Locked Rotor)						T							
15.3.4 RCP Shaft Break						T							
15.4.1 - Unc. RCCA Bank Wd. from a Subc. or LP. Startup			T			T						T	
15.4.2 - Unc. RCCA Bank Withdrawal at Power						T							
15.4.3 RCCA Misalignment						T							
15.4.4 - Startup of an Inactive RCP at Inc. Temp.			T			T						T	
15.4.6 - CVCS Malfunction that Res. in Dec. in Boron Conc.				E				E	E	E	E	E	
15.4.8 - Spectrum of RCCA Ejection Accidents			T			T		E	E	E		T	
15.5.2 - CVCS Malfunction that Inc. Reactor Coolant Inventory				E	E	T		E	E	E			
15.6.1 - Inad. Opening of a Przr SRV or Inad. Op. of the ADS				E		T		E	E	E			
15.6.3 SG Tube Rupture				E	E	T		E	E	E			
15.6.5 LOCA				E	E	T		E	E	E			

Fig. 6.9 An illustrative format of a vulnerability summary chart for a pressurized water reactor

7. CONCLUSIONS

This report has described the software safety analysis techniques and the engineering guidelines for developing safety critical software to identify the state of the art in this field and to give the software safety engineer a trail map between the code & standards layer and the design methodology and documents layer. We have surveyed the management aspects of software safety activities during the software lifecycle in order to improve the safety. After identifying the traditional safety analysis techniques for systems, we have surveyed in details the software safety analysis techniques, software FMEA, software HAZOP, and software FTA. We also surveyed the state of the art in the software reliability assessment techniques. However, the most important results from the reliability techniques are not the specific probability numbers generated, but the insights into the risk importance of software features.

Software cannot be proven to be error-free, and therefore is considered susceptible to common-mode failures because identical copies of the software are present in redundant channels of safety-related systems. To defend against potential common-mode failures, high quality, defense-in-depth, and diversity are considered to be key elements in digital I&C system design. Implementation of software diversity should use independent systems with functional diversity. The use of system diversity, diverse software features and diverse design approaches should be considered. To minimize the possibility of CMFs and thus increase the plant reliability, we have provided D-in-D&D analysis guidelines.

This report will provide a possibility to fill a gap between the mandatory requirements (*what*) and the work practices (*how*) when conducting the processes and activities intended to improve the reliability of safety-critical software in digital I&C systems.

8. REFERENCES

[Andow] Andow P. Guidance on HAZOP Procedures for Computer Controlled Plants, KBC Process Technology, ISBN 0-7176-0367-9

[ANS86] ANS 8.3. "Criticality Accident Alarm System," ANSI/ANS 8.3, 1986.

[ANS87] ANS 10.4. "Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry," ANSI/ANS 10.4, 1987.

[ASM90] ASME Std NQA-2a-1990 Part 2.7. "Quality Assurance Requirements of Computer Software for Nuclear Facility Applications." ASME, 1990.

[ASM94] ASME Std NQA-1-1994. "Quality Assurance Requirements for Nuclear Facility Applications." ASME 1994.

[Bonda] Bondavalli, A. & Simoncini, L. Failure Classification With Respect To Detection, in First Year report, TASK B : Specification and Design for Dependability, Volume 2. ESPRIT BRA Project 3092: Predictably Dependable Computing Systems.

[Bor90] Boris Beizer, "Software Testing Techniques," 2nd Edition, Van Nostrand Reinhold, 1990.

[BTP19] NUREG-0800, BTP HICB-19. "Guidance for Evaluation of Defense-in-Depth and Diversity in Digital Computer-Based Instrumentation and Control Systems," Rev. 4, U.S. Nuclear Regulatory Commission, June 1997.

[BTP97] Standard Review Plan, Appendix 7-A BTP HICB-14-27 Rev. 4-June 1997.

[Bur93] Burns, D. J. & Pitblado, R. M., A Modified Methodology For Safety Critical Systems Assessment, in F. Redmill & T. Anderson, eds, Directions in safety critical systems: SCS Symposium, Bristol, 1993 (Springer-Verlag)

[Cha88] Cha S. S, Leveson N. G, Shimeall T. J. Safety Verification in Murphy using Fault Tree Analysis. ICSE-10, IEEE Computer Society Press, 1988. p. 377-386.

[Cha91] Cha S. S. A Safety-Critical Software Design and Verification Technique. Ph.D. Dissertation, Information and Computer Science, UC Irvine, 1991.

[Chu93] Chudleigh, M., Hazard Analysis Using HAZOP: A Case Study, in Proceedings of SAFECOMP 93.

[Cla93] Clarke S. J, McDermid J. A. Software Fault Trees and Weakest Preconditions: A Comparison and Analysis. IEE Software Engineering. 1993. p. 225-236.

[Del97] Delic, K., Mazzanti, M. and Stringini, L., "Formalizing engineering judgement on software dependability via belief networks," DCCA-6, Germany, 1997.

[Ear92] Earthy, J. V., Hazard and Operability Studies As An Approach To Software Safety Assessment, in Proceedings of IEE Computing and Control Division Colloquium on Hazard Analysis, IEE, November 1992

[EPR87] EPRI NP-4924, "An Approach to the Verification of a Fault Tolerant, Computer-Based Reactor Safety System: A Case Study Using Automated Reasoning," EPRI, January 1987.

[EPR92] EPRI NP-7343, "Integrated Instrumentation and Control Upgrade Plan," EPRI, February 1992.

[EPR94] EPRI Topical Report TR-102323. "Guidelines for Electromagnetic Interference Testing in Power Plants." 1994.

[EPR96] EPRI Topical Report TR-106439 (Draft). "Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications." Electric Power Research Institute, April 1996.

[Ezhil] Ezhilchelvan, P. D. & Shrivastava, S. K., A Classification Of Faults In Systems, University of Newcastle upon Tyne.

[Fen93] Fenelon P, McDermid J. A. An Integrated Tool Set for Software Safety Analysis. J. Systems Software, vol. 21, 1993. p. 279-290.

[Fri95] Friedman M. A, Voas J. M. Software Assessment: Reliability, Safety, Testability. John Wiley & Sons Publishing, 1995.

[Gla79] Glanford J. Myers, "The Art of Software Testing," John Wiley & Sons, Inc., 1979.

[Gor95] Gorski J, Wardzinski A. Formalizing Fault Trees. Proc. of the Safety-Critical Systems Symposium. Eds. F. Redmill, and T. Anderson. Springer-Verlag. Brighton, UK. February 7-9, 1995. p. 311-327.

[Hal77] Halstead, M.H.: *Elements of Software Science*. Prentice-Hall, Inc., New York, 1977.

[Han94] Hansen K. M, Ravn A. P, Stavridou V. From Safety Analysis to Formal Specification. ProCos II Technical Report, ESPRIT, Department of Computer Science, Technical University of Denmark, 1994.

[I1008] IEEE Std 1008. "IEEE Standard for Software Unit Testing," ANSI/IEEE Std 1008, 1987.

[I1012] IEEE Std 1012. "IEEE Standard for Software Verification and Validation Plans," ANSI/IEEE Std 1012, 1986.

[I1016] IEEE Std 1016. "Recommended Practice for Software Design Descriptions," ANSI/IEEE Std 1016, 1987.

[I1028] IEEE Std 1028. "IEEE Standard for Software Reviews and Audits." IEEE Std 1028, 1988.

[I1042] IEEE Std 1042. "IEEE Guide to Software Configuration Management," ANSI/IEEE Std 1042, 1987.

[I1058] IEEE Std 1058.1. "IEEE Standard for Software Project Management Plans," ANSI/IEEE Std 1058.1, 1987.

[I1063] IEEE Std 1063. "IEEE Standard for Software User Documentation." ANSI/IEEE Std 1063, 1987.

[I1074] IEEE Std 1074-1991. "IEEE Standard for Developing Software Life Cycle Processes." IEEE, 1991.

[I1219] IEEE Std 1219-1992. "IEEE Standard for Software Maintenance." IEEE, 1992.

[I1228] IEEE Std 1228. "IEEE Standard for Software Safety Plans," IEEE Std 1228, 1994.

[I1298] IEEE Std 1298. "IEEE Standard Software Quality Management System, Part 1: Requirements," IEEE Std 1298 (1992) and AS 3563.1, 1991.

[I279] ANSI/IEEE Std 279-1971. "Criteria for Protection Systems for Nuclear Power Generating Stations."

[I352] IEEE/ANSI 352-1987, "IEEE Guide for General Principles of Reliability Analysis of Nuclear Power Generation Station Safety System," IEEE, October 13, 1987.

[I379] ANSI/IEEE Std 379-1988. "Standard Application of the Single-Failure Criterion to Nuclear Power Generating Station Safety Systems."

[I384] IEEE-384-1992. "IEEE Standard Criteria for Independence of Class 1E Equipment and Circuits."

[I603] IEEE Std 603-1991. "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations."

[I603] IEEE Std 603. "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations," IEEE Std 603, 1991.

[I60880] Draft Amendment 1 to IEC 60880. "Software for Computers Important to Safety For Nuclear Power Plants – First Supplement to IEC Publication 880," Ed. 1, International Electrotechnical Commission, April 1999.

[I610] IEEE Std 610.12. "IEEE Standard Glossary of Software Engineering Terminology," IEEE STD 610.12, 1990.

[I61513] Draft IEC 61513. "Nuclear Power Plants – Instrumentation and Control for Systems Important to Safety – General Requirements for Computer-based Systems," International Electrotechnical Commission, April 1997.

[I7301] IEEE Std 730.1-1989. "IEEE Standard for Quality Assurance Plans." IEEE, 1989.

[I7302] IEEE Std 730.2. IEEE Guide to Software Quality Assurance Planning,: IEEE Draft 730.2 (1993).

[I7432] IEEE Std 7.4.3.2-1993. "IEEE Standard for Digital Computers in Safety Systems of Nuclear Power Generating Stations." IEEE, 1993.

[I828] IEEE Std 828-1990. "IEEE Standard for Software Configuration Management Plans." IEEE, 1990.

[I829] IEEE Std 829. "IEEE Standard for Software Test Documentation, ANSI/IEEE 829, 1983.

[I830] IEEE Std 830-1993. "IEEE Recommended Practice for Software Requirements Specifications." IEEE, 1993.

[I9821] IEEE Std 982.1. "IEEE Standard Dictionary of Measures to Produce Reliable Software," IEEE 982.1 (1988).

[IAE94] IAEA Technical Reports Series No. 367, Software Important to Safety in Nuclear Power Plants, 1994.

[IAE95] IAEA TR Draft-1995, "V&V of Software related to NPP Control and Instrumentation," IAEA, 1995.

[IAE96] IAEA Draft Safety Practice on Computer Based System Important to Safety in Nuclear Power Plants, Nov. 1996.

[IC371] IEEE Std C37.1. "IEEE Standard Definition, Specification and Analysis of Systems Used for Supervisory Control, Data Acquisition and Automatic Control," ANSI/IEEE C37.1 (1987).

[ICI88] An Introduction to Software Hazard and Operability Procedures, ICI 1988.

[IRC86] IEC 880. "Software for Computers in the Safety Systems of Nuclear Power Stations," EC Publication 880, 1986.

[ISA84] ISA S5.1. "Instrumentation Symbols and Identification," ANSI/ISA Std. S5.1 (1984).

[KINS98] KINS/AR-541. "Development of the Technology for D-I-D and Diversity Regulations of Computer-based Reactor Protection Systems," the Korea Institute of Nuclear Safety, April 1998.

[Lee00] J. S. Lee and S. D. Cha, "Software Safety Analysis of Hybrid Real-time Systems using Qualitative Formal Method," J. of RESS, December 2000. (submitted)

[Lev83a] Leveson N. G, Harvey P. R. Analyzing Software Safety. IEEE Trans. Software Eng. Vol. SE-9, No. 5, September 1983.

[Lev83b] Leveson N. G, Stolzy J. L. Safety Analysis of Ada programs using Fault Trees. IEEE Transactions on Reliability, Vol R-32, No-5. December 1983.

[Lev87] Leveson N. G, Stolzy J. L. Safety Analysis using Petri Nets. IEEE Trans. Software Eng. Vol. 13, 1987. p. 386-397.

[Lev95] Leveson N. G. SAFEWARE: System Safety and Computers. Addison-Wesley Publishing, 1995.

[Liu96] Liu S, McDermid J. A. Model-Oriented Approach to Safety Analysis Using Fault Trees and a Support System. J. Systems Software, vol. 35, 1996. p. 151-164.

[MaC76] McCabe, T.J.: *A Complexity Measure.* IEEE Transactions on Software Engineering, 2(1976)4, pp. 308-320.

[McD94] McDermid, J. A & Pumfrey, D. J., A Development of Hazard Analysis To Aid Software Design, COMPASS '94

[MoD89] Defense Standard 00-55, Ministry of Defense, Glasgow, May 1989.

[MoD94] Feasibility study for MoD PES HAZOP, MoD Ref NSM 13C/1063, 1994.

[MSE93] Staff Requirements Memorandum. "SECY-93-087, Policy, Technical, and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor (ALWR) Designs," U.S. Nuclear Regulatory Commission, July 21, 1993.

[N0493] NUREG-0493. "A Defense-in-Depth & Diversity Assessment of the RESAR-414 Integrated Protection System." U.S. Nuclear Regulatory Commission, March 1979.

[N0694] NUREG-0694. "TMI-Related Requirements for New Operating Reactor Licenses," 1980.

[N0737] NUREG-0737. "Clarification of TMI Action Plan Requirements." 1980.

[N1462] NUREG/CR-1462, "Draft Safety Evaluation Report Related to the Design Certification of Combustion Engineering System 80+," U.S. NRC, September 1992.

[N4640] NUREG/CR-4640, "Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry," Pacific Northwest Laboratory, August 1987.

[N5930] NUREG/CR-5930, "High Integrity Software Standards and Guidelines," NIST, U.S. DoC, September 1992.

[N6101] NUREG/CR-6101. "Software Reliability and Safety in Nuclear Reactor Protection Systems." 1993.

[N6263] NUREG/CR-6263, "High Integrity Software for Nuclear Power Plants," 1996.

[N6303] NUREG/CR-6303. "Method for Performing Diversity and Defense-in-Depth Analyses of Reactor Protection Systems," U.S. Nuclear Regulatory Commission, December 1994.

[N6421] NUREG/CR-6421, "A Proposed Acceptance Process for Commercial Off-the-Shelf (COTS) Software in Reactor Applications." March 1996.

[N6430] NUREG/CR-6430 Software Safety Hazard Analysis, 1996.

[N6463] NUREG/CR-6463. "Review Guidelines on Software Languages for Use in Nuclear Power Plant Safety Systems." June 1996.

[Nei96] Neil, M. and Fenton, N., "Predicting software quality using Bayesian belief networks," Proceedings of 21st Annual Software Engineering Workshop, NASA Goddard Space Flight Center, pp. 217-230, 1996.

[NRC97] Digital Instrumentation and Control Systems in Nuclear Power Plants – Safety and Reliability Issues, Section 5: Common-Mode Software Failure Potential, Final Report, Committee on Application of Digital Instrumentation and Control Systems to Nuclear Power Plant Operations and Safety, Board on Energy and Environmental Systems, Commission on Engineering and Technical Systems, National Research Council, National Academy Press, Washington, D.C., 1997.

[Pra92] Prather, R.E.: *Hierarchical Software Metrics*. Tutorial on the Second International Confernce on Software Quality, Research Triangle Park, NC, October 1992.

[R1105] Regulatory Guide 1.105. "Instrument Spans and Setpoints." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1996.

[R1118] Regulatory Guide 1.118. "Periodic Testing of Electric Power and Protection Systems." Office of Nuclear Regulatory Commission, 1995.

[R1151] Regulatory Guide 1.151. "Instrument Sensing Lines." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1983.

[R1152] Regulatory Guide 1.152. "Criteria for Digital Computers in Safety Systems of Nuclear Power Plants." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1996.

[R1153] Regulatory Guide 1.153. "Criteria for Power, Instrumentation, and Control Portions of Safety Systems." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1996.

[R1168] Regulatory Guide 1.168. "Verification, Validation, Reviews and Audits for Digital Computer Software Used in Safety Systems of Nuclear Power Plants." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1997.

[R1169] Regulatory Guide 1.169. "Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1997.

[R1170] Regulatory Guide 1.170. "Software Test Documentation for Digital Computer Software Used in Safety Systems of Nuclear Power Plants." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1997.

[R1171] Regulatory Guide 1.171. "Software Unit Testing for Digital Computer Software Used in Safety Systems of Nuclear Power Plants." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1997.

[R1173] Regulatory Guide 1.173. "Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1997.

[R153] Regulatory Guide 1.53. "Application of the Single-Failure Criterion to Nuclear Power Plant Protection Systems." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1973.

[R197] Regulatory Guide 1.97. "Instrumentation for Light-Water-Cooled Nuclear Power Plants to Assess Plant and Environs Conditions During and Following an Accident." Rev. 3, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, May 1983.

[SEC91] SECY-91-292, "Digital Computer Systems for Advanced Light Water Reactors," U.S. Nuclear Regulatory Commission, Sept. 16, 1991.

[SEC93] SECY-93-087, "Policy, Technical, and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor (ALWR) Designs," U.S. Nuclear Regulatory Commission, April 2, 1993.

[SPM93] NPX80-SQP-0101.0, "Software Program Manual for NUPLEX 80+," ABB-CE, January 21, 1993.

[Sub95] Subramanian S, Vishnuvajjala R. V, Mojdebakhsh R, Tsai W.T, Elliott L. A. Framework for Designing Safe Software Systems. COMPSAC'95, 1995. p. 409-414.

[Vog94] Udo Voges, "Software diversity," Reliability Engineering and System Safety, Vol. 43, 1994, pp. 103-110.

[Wym97] Wyman, R. H. and Johnson, G. L., "Defense Against Common-Mode Failures in Protection System Design," IAEA Committee Meeting on Advanced Technologies for Improving Availability and Reliability of Current and Future Water Cooled Nuclear Power Plants, Argonne, Illinois, Sept. 8-11, 1997.

APPENDIX A. DEFINITIONS AND TERMS

Accident. Accidents are defined as those conditions of abnormal operation that result in limiting faults: These are occurrences that are not expected to occur but are postulated because their consequences would include the potential for the release of significant amounts of radioactive material. Limiting faults are further defined as those accidents whose effects circumscribe or bound the effects of similar faults of lesser magnitude. For the purposes of the analysis described in this document, a basis set of limiting faults, identical to those considered in the Standard Safety Analysis Report, Chapter 15, should be identified.

Anticipated Operational Occurrences: For the purposes of the analysis described in this document, a basis set of anticipated operational occurrences should be identified by the following criteria. "Anticipated operational occurrences" mean those conditions of normal operation which are expected to occur one or more times during the life of the nuclear power unit and include but are not limited to loss of the turbine generator set, isolation of the main condenser and loss of offsite power" (10 CFR 50, Appendix A, Definitions and Explanations). Such occurrences are further categorized as to frequency:

- Incidents of moderate frequency—these are incidents, any one of which may occur during a calendar year for a particular plant.
- Infrequent incidents—these are incidents, any one of which may occur during the lifetime of a particular plant.

Architectural Design. The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

Baseline. A defined configuration on which a design freeze decision has been implemented, thereafter requiring formal configuration

Block. Generally, a system is described as an arrangement of components or black boxes interconnected by communication, electrical connections, pipes, or physical effects. This kind of description, often called a "system architecture," may be too complex or may not be partitioned conveniently for diversity and defense-in-depth analysis. A more convenient description may be obtained by restricting the portion of the system under consideration to instrumentation and control equipment and partitioning the restricted portion into "blocks." A "block" is the smallest portion of the system under analysis for which it can be credibly assumed that internal failures, including the effects of software errors, will not propagate to other equipment. The objective of choosing blocks is to reduce the need for detailed examination of internal failure mechanisms while examining system behavior under reasonable assumptions of failure containment.

Examples of typical software-containing blocks are computers, local area networks or multiplexers, or programmable logic controllers (PLCs). A block can be solely hardware, but there are no solely software blocks; software-containing blocks suffer the distinction that both hardware or software faults (and sometimes both acting together) can cause block failure. Consequently, it is difficult to separate the effects of software from the machine that executes that software. For example, a software defect in one small routine can cause an entire computer to fail by corruption of other data or software. Guideline 1 and Guidelines 6 through 9 (see section 6.2) provide additional direction on block choice and failure propagation limits.

Categorization of Functions Important to Safety. The Assignment of a category for each I&C function and for systems and equipment important to safety. Note - The process of assignment according to categories A, B, and C is described in IEC 1226.

Channel. A channel is defined as a set of interconnected hardware and software components that processes an identifiable sensor signal to produce a single protective action signal in a single division when required by a generating station condition. A channel includes the sensor, data acquisition, signal conditioning, data transmission, bypasses, and logic up to voters or actuating device inputs. The objective of the channel definition is to define subsets of a reactor protection system that can be unambiguously tested or analyzed from input to output.

Common-Mode (or -Cause) Failure. Common-mode failures (CMFs) are causally related failures of redundant or separate equipment. For example, (1) A CMF of identical subsystems across redundant divisions defeats the purpose of redundancy, or (2) A CMF of different subsystems or echelons of defense defeats the use of defense-in-depth. CMF embraces all causal relations, including severe environments, design errors, calibration and maintenance errors, and consequential failures.

Configurable Software. Consists of a set of basic functional elements and a set of rules describing how these elements can combined. The user configure the basic elements into the specified system. This software can be treated as existing software with either accessible or proprietary documentation. It is possible that the base software has already been rigorously validated and in this case only the application specific extension has to be validated.

Defense-in-Depth. Defense-in-depth is a principle of long standing for the design, construction and operation of nuclear reactors, and may be thought of as requiring a concentric arrangement of protective barriers or means, all of which must be breached before a hazardous material or dangerous energy can adversely affect human beings or the environment. The classic three physical barriers to radiation release in a reactor—cladding, reactor pressure vessel, and containment—are an example of defense-in-depth.

Design Diversity. Design diversity is the use of different approaches, including both software and hardware, to solve the same or similar problem. Software diversity is a special case of design diversity and is mentioned separately because of its potential importance and its potential defects. The rationale for design diversity is that different designs will have different failure modes and will not be susceptible to the same common influences. A factor that weakens this argument is that different designs may nonetheless use similar elements or approaches.

Design Fault. A fault due to the inadequate design of an item.

- Note 1 - A design fault is caused by a human error during system development.
- Note 2 - Design faults in software are usually latent, transient, recurrent and systematic.
- Note 3 - Design faults give rise to failure during operation when activated by a certain combination of conditions referred to as the trigger. Since these conditions are encountered at random during operation, design facilities are random events.

Detailed Design. The process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented.

Diverse System. An independent system designed and developed with the intention of ensuring that design failures occur independently.

Diversity. Diversity is a principle in instrumentation systems of sensing different parameters, using different technologies, using different logic or algorithms, or using different actuation means to provide several ways of detecting and responding to a significant event. Diversity is complementary to the principle of defense-in-depth and increases the chances that defenses at a particular level or depth will be actuated when needed. Defenses at different levels of depth may also be diverse from each other. There are six important types of diversity to consider: human diversity, design diversity, software diversity, functional diversity, signal diversity, and equipment diversity.

Dynamic Analysis. The process of evaluating a system or component traced on its behavior during execution. Contrast with static analysis.

Echelons of Defense. "Echelons of defense" are specific applications of the principle of defense-in-depth to the arrangement of instrumentation and control systems attached to a nuclear reactor for the purpose of operating the reactor or shutting it down and cooling it. Specifically, the echelons are the control system, the reactor trip or scram system, the Engineered Safety Features actuation system (ESFAS), and the monitoring and indicator system. The echelons may be considered to be concentrically arranged in that when the control system fails, the reactor trip system shuts down reactivity; when both the control system and the reactor trip system fail, the

ESFAS continues to support the physical barriers to radiological release by cooling the fuel, thus allowing time for other measures to be taken by reactor operators to reduce reactivity. All four echelons depend upon sensors to determine when to perform their functions, and a serious safety concern is to ensure that no more than one echelon is disabled by a common sensor failure or its direct consequences.

Equipment Diversity. Equipment diversity is the use of different equipment to perform similar safety functions, in which ``different'' means sufficiently unlike as to significantly decrease vulnerability to common failure. The fact that equipment is made by different manufacturers does not guarantee diversity; many computer designs use the same semiconductor chips, and in the most extreme cases, two suppliers may acquire, re-label, and sell the same printed circuit boards from a single manufacturer. The use of diverse computer equipment may have an effect on software diversity; using a different computer architecture forces the use of diverse compilers, linkers, and other support software.

Error. A discrepancy between a computed. observed or measured value or condition and the true, specified or theoretical value or conditions.

Erroneous State. An incorrect internal state of a system, due to a component failure or an external failure.

Failure. The event of an item not providing its full required service.

Note 1 - A failure is an event in time. A fault is a state in the system.

Note 2 - A failure may be due to a physical failure of a hardware component, to activation of a latent design fault, or to an external failure.

Note 3 - After a failure, an item may recover and resume its required service after a break. partially recover and continue to provide some of its required functions (fail degraded) or it may remain down (complete failure) until repaired.

Fault. The state of an item characterized by inability to perform a required function, excluding the inability during preventive maintenance or other planned action, or due to lack of external resources.

Fault Activation. The event in which a latent fault give rise to a failure in response to a trigger.

Fault Mode. An observable state of an item, which can give rise to a failure under certain operating conditions. (Note - This covers both the state of a hardware component following a physical failure and a design fault.)

Fault Tolerance. The attribute of an item which makes it able to perform a required function in the presence of certain given sub-item faults.

Formal Proof. A complete mathematical proof constructed to discharge proof obligations.

Functional Diversity. Two systems are functionally diverse if they perform different physical functions though they may have overlapping safety effects. For example, cooling systems normally intended to function when containment is isolated are functionally different from other liquid control systems intended to inject coolant or borated water for other reasons. However, the other liquid control systems may have a useful cooling effect, while the isolation cooling systems may have useful coolant makeup side effects. Functional diversity is often useful when determining if sufficient mitigation means have been employed in a postulated accident; a combination of alternative systems in the face of primary system failure may be enough to mitigate the effects of an accident. A type of functional diversity, called "aspect" diversity, was applied to systems using relays, specifically to distinguish "de-energize to trip" arrangements from "energize to trip" arrangements.

Functional Isolation. Means for preventing the functioning of a circuit or a system from being influenced by the failure of another circuit or system.

Functional Requirement. A system requirement that specifies a function that a system or system component must be capable of performing. Functional requirements define the behavior of the system, that is, the fundamental process of transformation that software and hardware components of the system perform on inputs to produce outputs.

Functional Specification. A document that specifies the functions that a system or component must perform. Often a pan of a requirements specification.

Guide word. A word or phrase which expresses and defines a specific type of deviation from design intent.

Hazard. A hazard is a state or a set of conditions of a system that, together with other conditions in the environment of the system, will lead inevitably to an accident. A hazard is defined with respect to the environment of the system or component. What constitute a hazard depends upon where the boundaries of the system are drawn.

Hazard analysis. An analysis for the purpose of exploring the hazards which may be caused by the system or which may affect the system.

HAZOP Study. A formal systematic examination, by a team under the management of a trained leader, of the design intentions of a new or existing system or parts of a system, to identify hazards, mal-operation or mal-function of individual entities within the system and the consequences on the system as a whole and on its environment. It typically includes several HAZOP Study Meetings.

Human Diversity. The effect of human beings on the design, development, installation, operation, and maintenance of safety systems is known to be extremely variable, and has been a factor in several serious accidents. Used in a positive way, human diversity can be a plus for system safety. For instance, using different maintenance personnel to calibrate separate, redundant divisions of safety instrumentation may provide some assurance that the same, systematic error is not made in all divisions. Using separate designers to design functionally diverse safety systems may reduce the possibility of similar design errors.

Instrumentation System. A plant instrumentation system is that equipment which senses various plant parameters and transmits appropriate signals to control systems, to the reactor trip system, to the engineered safety features actuation system, and to the monitoring and indicator system for use in determining the actions these systems or reactor operators will take. Independence is required between control systems, safety-related monitoring and display systems, the safety systems, and between redundant divisions of the safety systems.

Latent Fault. An existing, fault located in a software component.

Maintainability. The probability that a given active maintenance action to an item under given conditions of use can he carried out within a stated time interval when the maintenance is performed under stated conditions and using stated procedures and resources.

Maintenance. The combination of all technical and administrative actions, including supervision actions, intended to retain an item in, or restore it to, a state in which it can perform a required function.

Monitoring and Indicator System. The monitoring and indication echelon is the slowest and also the most flexible echelon of defense. Like the other three echelons, operators are dependent upon accurate sensor information to perform their tasks, but, given information, time, and means, can perform previously unspecified logical computations to react to unexpected events. The monitoring and indication echelon includes both Class 1E and non-Class 1E manual controls, monitors, and indicators required to operate equipment nominally assigned to the other three echelons.

N-Version Software. A set of different programs. known as versions, are developed to meet a common requirement and common acceptance test. Concurrent and independent execution of these versions takes place. generally in redundant hardware. Identical inputs in test systems or corresponding inputs in redundant systems are used. A predetermined strategy such as voting is used to decide between conflicting outputs in different versions.

Operability. The capacity of a system to function. In this Standard, the ability of the system to perform its functions is implied, but in the military context there is often an additional implication of the ability of the operators to use the system effectively.

Operational System Software. Parts of the system software used by the application software and which run on the target processor during operation.

Performance specification. A document that specifies the performance characteristics that a system or component must possess. These characteristics typically include speed, accuracy and memory usage. Often a part of a requirements specification.

Plant Functional Safety Analysis. The process of identification of the I&C functions important to safety of nuclear power plant.

Pre-Developed Software. An software which has not been specifically developed and verified to meet the requirements of the application for which it will be used. It may have been developed to satisfy a general market need or meet the requirements of another application.

Quality Assurance. All the planned and systematic activities implemented within the quality system and demonstrated as needed, to provide adequate confidence that an entity will fulfill. requirements for quality.

Reactor Trip or Scram System. The reactor trip echelon is that safety equipment designed to reduce reactivity rapidly in response to an uncontrolled excursion. It consists of instrumentation for detecting potential or actual excursions, means for rapidly and completely inserting the reactor control rods, and may also include certain chemical neutron moderation systems.

Recovery Block Technique. Alternative software versions for the same function are organized such that an acceptance test is used to check the results found by the versions in the same equipment If the test is not passed. recovery is implemented by initial state restoration, followed by the execution of the alternate version.

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

Re-Usable Software Modules. Basic software modules implementing application functions that can be configured .

Risk. Risk is the hazard level combined with (1) the likelihood of the hazard leading to an accident and (2) hazard exposure or duration. Risk is the combination of the frequency, or probability, and the consequence of an accident.

Safety. Safety is freedom from accident or losses.

Safety analysis. An analysis carried out with the purpose of assessing and examining the safety of the system and its surroundings. A hazard analysis is a necessary element of a safety analysis.

Safety-critical Software: Software that falls into one or more of the following categories:

a) Software whose inadvertent response to stimuli, failure to respond when required, response out-of-sequence, or response in combination with other responses can result in an accident.

b) Software that is intended to mitigate the result of an accident.

c) Software that is intended to recover from the result of an accident.

Safety Functions Requirements Specifications. This specification contains the requirements for the safety functions that have to be performed by the systems important to safety.

Signal Diversity. Signal diversity is the use of different sensed parameters to initiate protective action, in which any of the parameters may independently indicate an abnormal condition, even if the other parameters fail to be sensed correctly. For example, in a BWR, neutron flux increase due to void reduction is a diverse parameter to reactor pressure excursion for events that cause a reactor pressure pulse.

Software Diversity. Software diversity is the use of different programs designed and implemented by different development groups with different key personnel to accomplish the same safety goals—for example, using two separately designed programs to compute when a reactor should be tripped. It has been suggested that sufficient diversity can be obtained by implementing the same specification through intentionally diverse designs (possibly by the same programming team); however, the bulk of significant reported experience concerns independent software teams. The great hope of software diversity is that different programmers will make different mistakes. Unfortunately, some (very sparse) data suggest that different programmers designing to the same requirements too often make similar mistakes.

Software Failure. System failure due to the activation of a design fault in a software component.

Software Fault. Design fault located in a software component.

Software Hazard: A software condition that is a prerequisite to an accident.

Software Release. A version of the software product that has been extended with new functionality as compared with the previous version

Software Reliability. The component of the system reliability which depends on software failures.

Software Safety: Freedom from software hazards.

Software Safety Integrity. A qualitative measure that signifies the likelihood of software in a computer-based system achieving its safety functions under all stated conditions within a stated period of time.

Software Safety Integrity Level. One of a number of possible discrete levels for specifying the safety integrity of software in safety systems according to pre-defined categories.

Software Safety Program: A systematic approach to reducing software risks.

System Hazard: A system condition that is a prerequisite to an accident.

System Safety: Freedom from system hazards.

APPENDIX B. SOFTWARE SAFTY TECHNIQUES AND METHODS

The Software Safety Plan requires the specification of the types of analyses that will be performed during the software life cycle. The information in this annex may be helpful in preparing the Software Safety Plan.

B.1 Software safety requirements analyses

The software safety requirements analysis evaluates software and interface requirements and identifies errors and deficiencies that could contribute to a hazard. It is the basis for subsequent software safety analyses. Analyses may include, but are not limited to, those listed below.

a) Criticality analysis identifies all software requirements that have safety implications. Each requirement in the Software Requirements Specification (SRS) is evaluated against the various system hazard analyses (including the PHA) to assess its potential for unacceptable risk. Each requirement in the SRS is evaluated against the system design to ensure that each safety-critical software requirement imposed by the system design is satisfied in the SRS. Requirements that satisfy either portion of this analysis are termed safety-critical requirements. A criticality level is assigned to each safety-critical requirement based on the estimated risk.

b) Specification analysis evaluates each safety-critical software requirements. This evaluation is with respect to a list of qualities, such as, completeness, correctness, consistency, testability, robustness, integrity, reliability, usability, flexibility, maintainability, portability, interoperability, accuracy, auditability, performance, internal instrumentation, security, and training. (Items are listed in no particular order.)

c) Timing and sizing analysis evaluates safety implications of safety-critical requirements that relate to execution time, clock time, and memory allocation. During requirements analysis, timing analysis identifies conditions, events, and time intervals that satisfy one or more of the following criteria:

    i) If Condition C becomes true, then Event A must occur within T seconds.

    ii) If Condition C becomes true, then Event A must not occur until T seconds have elapsed.

    iii) Event B must not occur until T seconds after Event A has occurred.

    A preliminary performance analysis may be performed.

d) Different software system analyses may be required if more than one software system is being integrated. Such integration will significantly expand the amount of analysis required in the software safety requirements analysis. Specific analysis of the allocation of software requirements to the separate systems can reduce subsequent integration and interface errors

related to safety. This is particularly true when a system hazard results in a requirement that is partially implemented in two or more software systems.

B.2 Software safety design analysis

The software safety design analysis verifies that the safety-critical portion of the software design correctly implements the safety-critical requirements and introduces no new hazards. Analyses may include, but are not limited to, those listed below.

a) Logic analysis evaluates the safety-critical equations, algorithms, and control logic of the software design.

b) Data analysis evaluates the description and intended use of each data item in the software design. This analysis ensures that the structure and intended use of data will not result in a hazard. Data structures should be assessed for data dependencies that circumvent isolation, partitioning, data aliasing, and fault containment issues affecting safety, and the control or mitigation of hazards.

c) Interface analysis verifies the proper design of a software component's safety-critical interfaces with other components of the system, both internal and external. The major areas of concern with interfaces are properly defined protocols and control and data linkages. External interfaces should be analysed to demonstrate that communication protocols in the design are compatible with interfacing requirements. Hazards associated with an interface are also related to the system context and the environmental context as defined by their state at any point in time. The interface analysis must document this system and environment contexts. Interface analysis is also a tool that indicates the source of a system-level hazard and areas where further analyses are required.

d) Constraint analysis evaluates the safety of restrictions imposed on the selected design by the requirements and by real-world restrictions. The impacts of the environment on this analysis can include such items as the location and relation of clocks to circuit cards, the timing of a bus latch when using the longest safety-related timing to fetch data from the most remote circuit card, interrupts going unsatisfied due to a data flood at an input, and human reaction time.

e) Functional analysis ensures that each safety-critical software requirement is covered and that an appropriate criticality level is assigned to each software element.

f) Software element analysis examines software elements that are not designated safety-critical and ensures that these elements do not cause a hazard.

g) Based on the results of the timing and sizing analysis conducted for software safety requirements analysis, timing and sizing estimates can be established to allow evaluation of the operating environment.

h) Reliability predictions may be made for safety-critical software elements. Acceptable risk levels as defined in the software safety requirements may set reliability goals (see IEEE Std 982.1-1988 for additional information).

B.3 Software safety code analysis

The software safety code analysis verifies that the safety-critical portions of the design are correctly implemented in the code. Analyses may include, but are not limited to, those listed below.

a) Logic analysis evaluates the sequence of operations represented by the coded program and detects programming errors that might create hazards.

b) Data analysis evaluates the data structure and usage in the code to ensure each is defined and used properly by the program. Analysis of the data items used by the program is usually performed in conjunction with logic analysis.

c) Interface analysis ensures compatibility of program modules with each other and with external hardware and software.

d) Constraint analysis ensures that the program operates within the constraints imposed upon it by requirements, the design, and the target computer. Constraint analysis is designed to identify these limitations, to ensure that the program operates within them, and to ensure that all interfaces have been considered for out-of-sequence and erroneous inputs.

e) Programming style analysis ensures that all portions of the program follow approved programming guidelines.

f) Non-critical code analysis examines portions of the code that are not considered safety-critical code to ensure that they do not cause hazards. As a general rule, safety-critical code should be isolated from non-safety-critical code. The intent of this analysis is to prove that this isolation is complete and that interfaces between safety-critical code and non-safety-critical code do not create hazards. If isolation is not provable, the Plan should discuss how to handle the risk.

g) Timing and sizing analysis is further refined to ensure that no hazards due to timing or sizing factors have been added by the process of writing the code.

B.4 Software safety test analysis

Software safety test analysis demonstrates that safety requirements have been correctly implemented and the software functions safely within its specified environment. Tests may include, but are not limited to, the following:

a) Computer software unit level testing that demonstrates correct execution of critical software elements.

b) Interface testing that demonstrates that critical computer software units execute together as specified.

c) Computer software configuration item testing that demonstrates the execution of one or more system components.

d) System-level testing that demonstrates the software's performance within the overall system.

e) Stress testing that demonstrates the software will not cause hazards under abnormal circumstances, such as unexpected input values or overload conditions.

f) Regression testing that demonstrates changes made to the software did not introduce conditions for new hazards.

B.5 Software safety change analysis

The starting point of the change analysis is the safety-critical design elements that are affected directly or indirectly by the change. The purpose of software safety change analysis is to show that the change does not create a hazard, does not impact on a previously resolved hazard, does not make a currently existing hazard more severe, and does not adversely affect any safety-critical software design element.

APPENDIX C. POTENTIAL SOFTWARE SAFETY ANALYSIS METHODS

The New Mexico chapter of the System Safety Society issued a report on safety analysis in 1993. The relevant portion of that report is a 312-page discussion of hazard analysis techniques. Ninety techniques are discussed to varying levels of detail. The following topics are included for each technique:

- alternate names
- purpose
- method
- application
- thoroughness
- mastery required
- difficulty of application
- general comments and references

Many of the techniques do not apply directly to software (for example, Tornado Analysis). Some of the remaining analyses could have indirect application to software. Bent Pin Analysis, for example, applies to connector pins in a cable connection. If the cable carries computer data, a bent pin could affect software functions. However, the analysis is performed on the cable, not the software, so it is considered to be indirect.

The 47 techniques that might potentially apply to software are listed below. The word "potential" means that it is conceivable that the technique could be used, not that there is any evidence of use. For each of these techniques, the list gives its name and an extract of the purpose. In some cases, the purpose sections were not very complete.

- **Accident Analysis** evaluates the effect of scenarios that develop into credible and incredible accidents. This is expected to be performed at the system level, but could be extended to software safety by considering the effect of software on the prevention, initiation or mitigation of accidents identified in the system accident analysis.
- **Cause-Consequence Analysis** combines the inductive reasoning features of Event Tree Analysis with deductive reasoning features of Fault Tree Analysis. The result is a technique that relates specific accident consequences to their many possible causes. Computer codes exist to assist in the performance of this analysis. GENII, RSAC4, MACCS, ARA, EPA-AIRDOS and HOTSPOT are examples.
- **Change Analysis** examines the potential effects of modifications from a starting point or baseline. The Change Analysis systematically hypothesizes worst-case effects from each modification from that baseline.

- **Checklist Analysis** uses a list of specific items to identify known types of hazards, design deficiencies and potential accident situations associated with common equipment and operations. The identified items are compared to appropriate standards.
- **Common Cause Analysis** identifies any accident sequences in which two or more events could occur as the result of a common event or causative mechanism. . Comparison-To-Criteria (CTC) Analysis provides a formal and structured format that identifies all safety requirements for a (software) system and ensures compliance with those requirements.
- **Contingency Analysis** is a method of preparing for emergencies by identifying potential accident-causing conditions and respective mitigating measures to include protective systems and equipment.
- **Critical Incident Technique** uses historical information or personal experience in order to identify or determine hazardous conditions and high-risk practices.
- **Criticality Analysis** ranks each potential failure mode identified in a Failure Modes and Effects Analysis (FMEA) according to the combined influence of severity classification and its probability of occurrence based on the best available data. It is often combined with FMEA, forming a Failure Modes, Effects and Criticality Analysis (FMECA).
- **Digraph Utilization Within System Safety** is used to model failure effect scenarios within large complex systems, thereby modeling FMEA data. Digraphs can also be used to model hazardous events and reconstruct accident scenarios. As a result, both hazard analysis and accident investigation processes can be improved via modeling event sequences.
- **Event and Casual Factor Charting** reconstructs the event and develops root cause(s) associated with the event.
- **Event Tree Analysis** is an analytical tool that can be used to organize, characterize and quantify potential accidents in a methodical manner. An event tree models the sequence of events that results from a single initiating event.
- **Failure Modes and Effects Analysis (FMEA)** determines the result or effects of sub-element failures on a system operation and classifies each potential failure according to its severity.
- **Failure Modes, Effects and Criticality Analysis (FMECA)** tabulates a list of equipment in a process along with all of the possible failure modes for each item. The effect of each failure is evaluated.
- **Fault Hazard Analysis** is a basic inductive method of analysis used to perform an evaluation that starts with the most specific form of the system and integrates individual examinations into the total system evaluation. It is a subset of FMEA.

- **Fault Isolation Methodology** is applied to large hardware/software systems that are unmanned and computer-controlled. There are five specific methods: half-step search, sequential removal or replacement, mass replacement, lambda search and point of maximum signal concentration.
- **Fault Tree Analysis (FTA)** assesses a system by identifying a postulated undesirable end event and examines the range of potential events that could lead to that state or condition.
- **Hazard and Operability Study (HAZOP)** is a group review method that assesses the significance of each way a process element could malfunction or be incorrectly operated. The technique is essentially a structured brainstorming session using specific rules.
- **Hardware/Software Safety Analysis** examines an entire computer system so that the total system will operate at an acceptable level of risk.
- **Human Error Analysis** is used to identify the systems and the procedures of a process where the probability of human error is of concern. This technique systematically collects and analyzes the large quantities of information necessary to make human error assessments.
- **Human Factors Analysis** allocates functions, tasks and resources among humans and machines.
- **Interface Analysis** identifies potential hazards that could occur due to interface incompatibilities.
- **Maximum Credible Accident/Worst-Case Analysis** determines the upper bounds on a potential accident without regard to the probability of occurrence of the particular accident identified.
- **Nuclear Safety Cross-Check Analysis (NSCCA)** verifies and validates software designs. It is also a reliability hazard assessment method that is traceable to requirements-based testing.
- **Petri Net Analysis** provides a technique to model system components at a wide range of abstraction levels. It is particularly useful in modeling interactions of concurrent components. There are many other applications.
- **Preliminary Hazard Analysis (PHA)** can be used in the early stages of system design (possibly including software design), thus saving time and money which could have been required for major redesign if the hazards were discovered at a later date.
- **Preliminary Hazard List (PHL)** creates a list of hazards to enable management to choose any hazardous areas to place management emphasis.
- **Probabilistic Risk Assessment (PRA)** provides an analysis technique for low probability, but catastrophically severe events. It identifies and delineates the combinations of events

that, if they occur, will lead to an accident and an estimate of the frequency of occurrence for each combination of events, and then estimates the consequences.

- **Production System Hazard Analysis** identifies (1) potential hazards that may be introduced during the production phase of system development which could impair safety and (2) their means of control. This could apply to software if ``production" is replaced by ``operation."
- **Prototype Development** provides a modeling/simulation analysis technique that constructs early pre-production products so that the developer may inspect and test an early version.
- **Repetitive Failure Analysis** provides a systematic approach to address, evaluate and correct repetitive failures.
- **Root Cause Analysis** identifies causal factors relating to a mishap or near-miss incident. The technique goes beyond the direct causes to identify fundamental reasons for the fault or failure.
- **Safety Review** assesses a system or evaluates operator procedures for hazards in the design, the operations, or the associated maintenance.
- **Scenario Analysis** identifies and corrects potentially hazardous situations by postulating accident scenarios where credible and physically possible events could cause the accident.
- **Sequentially-Timed Events Plot (STEP) Investigation System** is a multi-linear events sequence-based analytical methodology used to define systems; analysis system operations to discover, assess and find problems; find and assess options to eliminate or control problems; monitor future performance; and investigate accidents. STEP results are consistent, efficiently produced, non-judgmental, descriptive and explanatory work products useful over a system's entire life cycle.
- **Single-Point Failure Analysis** identifies those failures that would produce a catastrophic event if they were to occur by themselves.
- **Sneak-Circuit Analysis** identifies unintended paths or control sequences that may result in undesired events or inappropriately timed events.
- **Software Failure Modes and Effects Analysis (SFMEA)** identifies software-related design deficiencies through analysis of process flow charting. It also identifies interest areas for verification/validation and test and evaluation.
- **Software Fault Tree Analysis** applies FTA to software. It can be applied to design or code.
- **Software Hazard Analysis** identifies, evaluates and eliminates or mitigates software hazards by means of a structured analytical approach that is integrated into the software development process.

- **Software Sneak Circuit Analysis (SSCA)** is used to discover program logic that could cause undesired program outputs or inhibits, or incorrect sequencing/timing.
- **Subsystem Hazard Analysis (SSHA)** identifies hazards and their effects that may occur as a result of the design of a subsystem.
- **System Hazard Analysis (SHA)** concatenates and assimilates the results of Subsystem Hazard Analyses into a single analysis to ensure that hazards or their controls or monitors are elevated to a system level and handled as intended.
- **Systematic Inspection** uses checklists, codes, regulations, industry consensus standards and guidelines, prior mishap experience and common sense to methodically examine a design, system or process in order to identify discrepancies representing hazards.
- **Uncertainty Analysis** identifies the incertitude of a result based on the confidence levels (or lack thereof) and variability associated with the inputs.
- **What-If Analysis** is a brainstorming approach in which a group of experienced individuals asks questions or voices concerns about possible undesired events in a process.
- **What-If/Checklist Analysis** is a combination of What-If Analysis and Checklist Analysis.

APPENDIX D. WOLSONG SOFTWARE HAZARD ANALYSIS PROCEDURE

AECL, the Canadian supplier of Wolsong nuclear power plant, developed a procedure of software hazard analysis for the safety critical software of Wolsong SDS2 [AEC93], and conducted a successful analysis using the procedure. The result is in Wolsong SDS2 software hazard analysis report [AEC94]. Although there are a number of types of hazard analysis ranging from FMEA, FMECA, and HAZOP to FTA, "Software Hazard Analysis" implies "Software Fault Tree Analysis" in the procedure for Wolsong nuclear power plants.

A hazard analysis consists of two analytical processes. The first process is "hazard identification" which defines the hazards. The second process is "hazard evaluation" which assesses the identified hazards. The most common "hazard evaluation" technique is the fault tree analysis. Practical applications reveal that hazard analysis is applied to the overall system instead of the software in isolation. Software itself typically cannot cause harm; however, hardware which it controls can cause damage.

System hazard analysis shall consist of two analytical processes, namely Preliminary System Hazard Analysis (PSHA) and System Fault Tree Analysis (FTA). System FTA shall encompass hardware, software, and human factors. Software FTA shall therefore represent a subset of the system FTA. Figure D.1 illustrates the complete system hazard analysis process [AEC93]. The system requirements definition and safety report are used to provide the information required to perform the PSHA. The system hazards, identified by the PSHA, provide the necessary input to the system hazard evaluation process (system FTA) since they define the top events of the system fault trees. Each system hazard must be evaluated using a separate system fault tree.

A fault tree is a symbolic logic diagram showing the cause and effect relationship between an undesired event and contributing causes. FTA is a backward process which attempts to determine all possible causes for the undesired event to occur. Forward approaches, such as testing or formal verification, try to ensure that all possible reachable states of the system are safe, whereas backward approaches try to ensure that unsafe states are not reachable.

The system hazards are defined by the preliminary system hazard analysis. They provide the necessary input to the system FTA process by defining the top events. As the development of the hardware portion of the system fault trees progresses, branches will be created defining software events. The identification of the software hazards requires the system hazard identification process to be performed followed by the partial completion of the system FTA process, that is, an attempt made to postulate undesired software events. There are following limitations in most software FTA techniques;

- It is labor intensive task with no present automatic generation of fault trees.

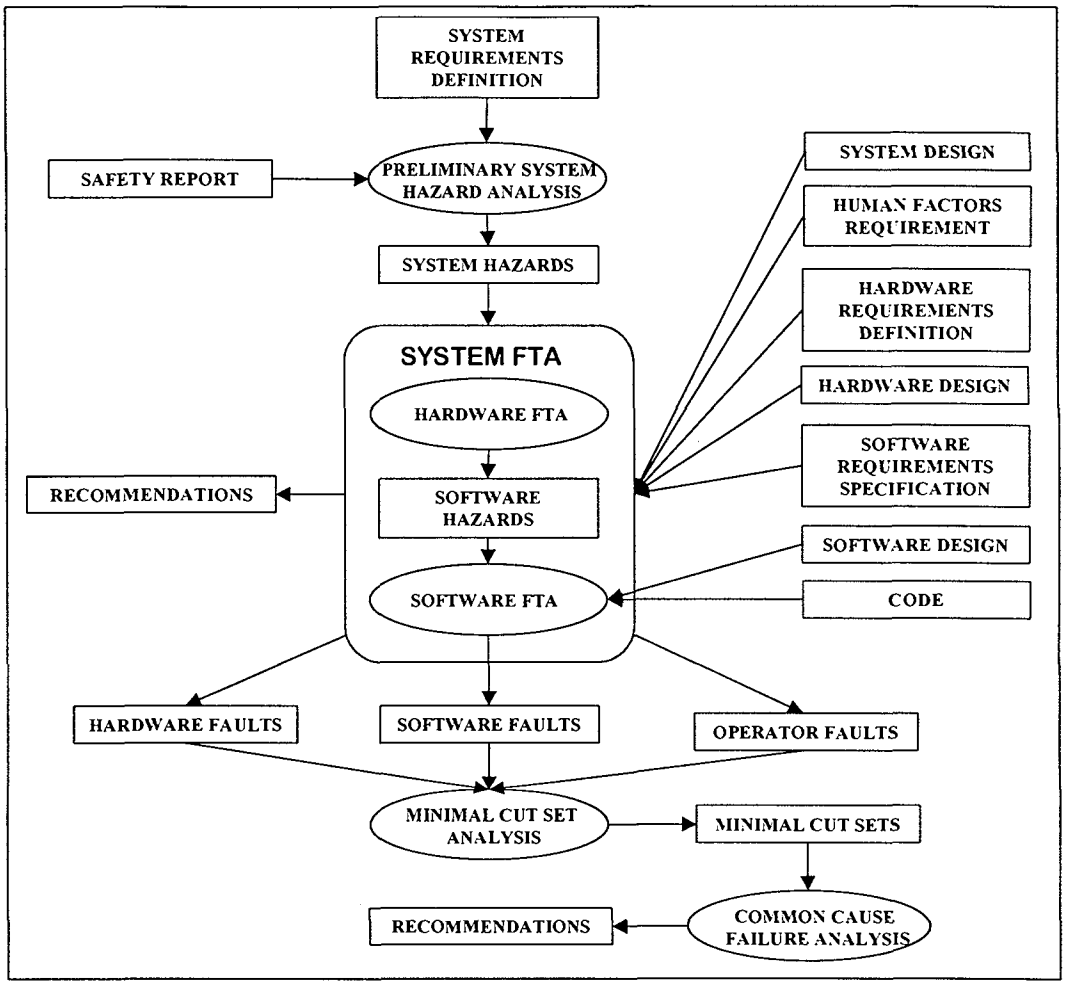


Fig. D.1 Wolsong System Hazard Analysis Process

- Software FTA is costly. A cost comparison with other software safety techniques is not practical as it is very application dependent.
- Safety-critical software, whose safety has been analyzed and validated using FTA techniques, cannot be re-used in other applications without performing a separate analysis unless its environment is identical.
- Within very large systems, it is often prohibitive to perform a complete FTA as the fault trees become huge and difficult to relate to the plant and its operation.
- Since software FTA is a static technique, it does not lend itself to situations where timing scenarios must be represented and analyzed.

There is no explicit causality information from logical faults of software to physical hazards of system. For example, in Fig. D.2 [Rav93], the logical contents of "Observation interval less than 30 sec" and "Gas leaks for more than 4 sec" must be the safety requirements for the controller, and will be implemented as software. However, when searching the cause of the upper event, it is thoroughly dependent on the analyst's knowledge to find the causal relation between the physical hazard (e.g.,

'Fire occurs' in Fig. D.2) and the logical fault of software. Here, software in requirement phase is the specification such as "Observation interval less than 30 sec" and "Gas leaks for more than 4 sec."

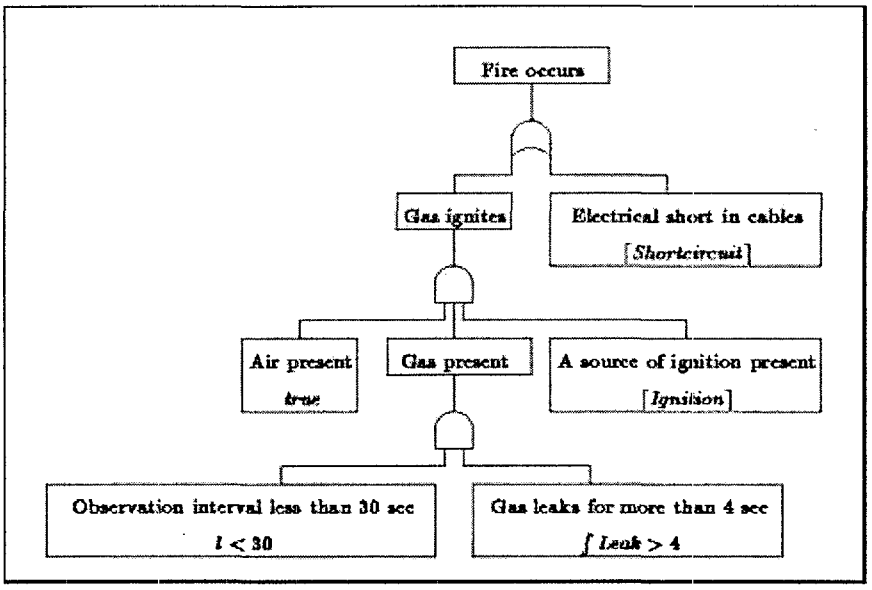


Fig. D.2 Fault Tree for a Gas Burner Controller

According to the Wolsong SDS2 software hazard analysis procedure [AEC93], there are two groups of outcome from software FTA. One group is the recommendations to improve the fault tolerance, and the other is the influence on testing. However, there is no detail guide to narrow down the testing domain from the software FTA. The analysis report [AEC94] by the procedure shows that they have identified about 50 software hazards, and provided the recommendations. The summary of the results by Wolsong software FTA is shown in Table D.1.

However, there was no explanation about how they found the software hazards from the software fault tree. That is, there is no mechanism to relate the logical aspects of software to the physical hazard of the system. For example, there must be an explanation why a specific IF-THEN-ELSE statement is hazardous. The explanation must not come from the program structures, but from the behavioral relations between the physical system and the logical software. In Wolsong SDS2 software FTA, more than 90% of software hazards are related to the program structure, and most of the hazards are expressed as "the current construct is not fully guarded against the hazard." The recommendations are also to change the program structure, not to change the logic or state of the program.

Table D.1 Hazards Identified from Wolsong SDS2 Software FTA

NAME OF SOFTWARE HAZARDS	NO	%	REMARKS
For construct hazard	4	7	
Initialization hazard	4	7	
IF-THEN-ELSE construct hazard	**38**	**67**	
CASE construct hazard	4	7	
Sequence checks hazard	1	2	
Main loop timer hazard	3	4	Hardware related
Wait in the main loop hazard	1	2	
Backup timers hazard	1	2	Hardware related
Common mode failure hazard	1	2	
Summary	57	**100**	

Figures D.3 and D.4 are a part of the fault trees for SDS2 of Wolsong nuclear power plant. When creating these fault trees, in addition to the general limitations of software FTA, there are also following drawbacks in the Wolsong software FTA approach;

- It is a static FTA based on the information of the program structure. In hardware FTA, the root cause of the top event is searched based on the structure of hardware. However, in software case, it is difficult to find the causes of software hazards from the structural information of the code, such as calling sequence, program structure, and module structure, because the I/O and the state changes of software determine whether the system could be hazardous or not.
- It is a code level FTA. It is difficult to relate the logical fault of software to the physical hazard of system because of the long conceptual distance between the logics of code and the physics of the system. It takes too much cost to fix the hazardous software because of the late phase in the lifecycle.

It depends on the ad hoc engineering judgments when deciding the cause of the upper event. The approach requires a lot of experiences of the analyst as not only a domain expert of the system, but also a software engineer.

Of course, there are many advantages of FTA itself and the Wolsong software FTA approach. For example, it provides the focus needed to give priority to catastrophic events and to determine the further verification. We show from next section that CRSA preserves all advantages of the FTA and also eliminates the above drawbacks.

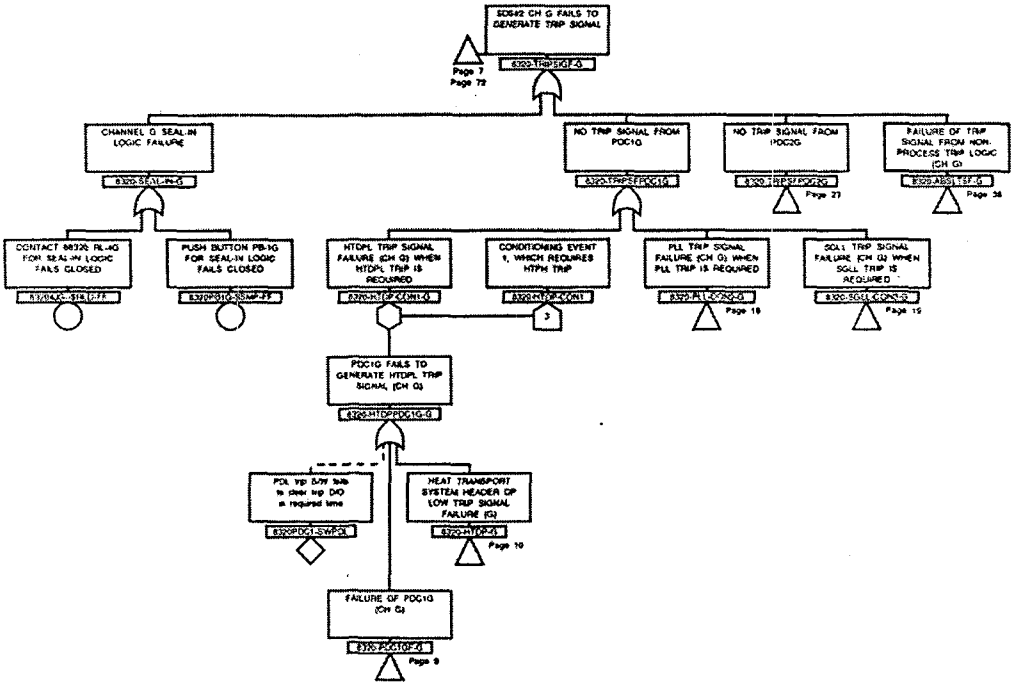


Fig. D.3 A System Fault Tree of Wolsong SDS2

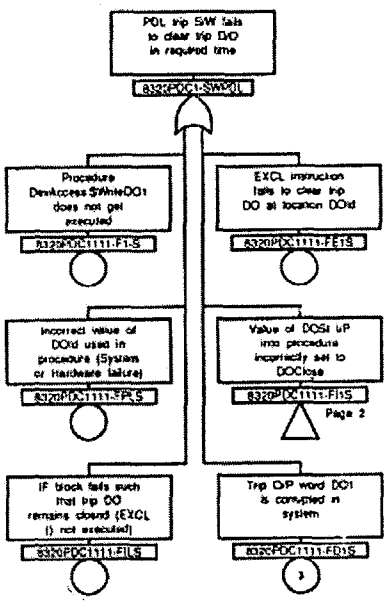


Fig. D.4 A Software Fault Tree of Wolsong SDS2

BIBLIOGRAPHIC INFORMATION SHEET

Performing Org. Report No.	Sponsoring Org. Report No.	Standard Report No.	INIS Subject Code
KAERI/AR-591/2001			

Title / Subtitle	Software Safety Analysis Techniques for Developing Safety Critical Software in the Digital Protection System of the LMR
Project Manager and Department	Jang-Soo Lee (MMIS Team)
Researcher and Department	Se-Woo Cheon, Chang-Hoi Kim (MMIS Team), Yun-Sub Sim (Kalimer Development Team)

Publication Place	Taejon	Publisher	KAERI	Publication Date	2001. 2
Page	119p.	Ill. & Tab.	Yes(O), No ( )	Size	21 x 29.6 cm.
Note	'00 Mid-and-Long Term Nuclear Research Project				
Classified	Open(O), Restricted( ), ___ Class Document			Report Type	State of the Art Report
Sponsoring Org.	MOST			Contract No.	

Abstract (15-20 Lines)

This report has described the software safety analysis techniques and the engineering guidelines for developing safety critical software to identify the state of the art in this field and to give the software safety engineer a trail map between the code & standards layer and the design methodology and documents layer. We have surveyed the management aspects of software safety activities during the software lifecycle in order to improve the safety. After identifying the conventional safety analysis techniques for systems, we have surveyed in details the software safety analysis techniques, software FMEA(Failure Mode and Effects Analysis), software HAZOP(Hazard and Operability Analysis), and software FTA(Fault Tree Analysis). We have also surveyed the state of the art in the software reliability assessment techniques. The most important results from the reliability techniques are not the specific probability numbers generated, but the insights into the risk importance of software features. To defend against potential common-mode failures, high quality, defense-in-depth, and diversity are considered to be key elements in digital I&C system design. To minimize the possibility of CMFs and thus increase the plant reliability, we have provided D-in-D&D analysis guidelines.

Subject Keywords (About 10 words)	Software Safety, Standards, Reliability, Common Mode Failures, Diversity, Safety-critical System

서 지 정 보 양 식

수행기관보고서번호	위탁기관보고서번호	표준보고서번호	INIS 주제코드
KAERI/AR-591/2001			

제목/부제	액금로 디지털 보호계통 소프트웨어 안전성 평가 기술
연구책임자 및 부서명	이장수(미래원자력기술개발단, MMIS팀)
연 구 자 및 부 서 명	천세우, 김창회 (MMIS팀) 심윤섭 (칼리머기술개발팀)

출 판 지	대전	발행기관	한국원자력연구소	발행년	2001. 2
페 이 지	119p.	도 표	있음(○), 없음( )	크 기	21*29.6 Cm.
참고사항	'00 원자력 중장기 계획사업				
비밀여부	공개(○), 대외비( ), __급 비밀		보고서종류	기술현황분석보고서	
연구위탁기관			계약 번호		

초록 (15-20줄내외)

전세계적으로 디지털 보호계통의 소프트웨어 안전성을 보장할 수 있는 확실한 기술이 없는 실정이다. 다만 품질보증, 수학적 검증, 안전성 평가, 형상관리, 확인 및 검증, 신뢰도 분석, 심층방어 및 다양성 기술 등 관리적 측면과 기술적 측면에서의 철저하고 다양한 노력을 통해 안전에 대한 믿음의 정도를 수용 가능하게 높임으로써 설계 인증 및 운전 허가를 받게 된다.

본 기술현황 분석보고서에서는 액체금속로(LMR: Liquid Metal Reactor) 보호계통 소프트웨어 안전성 현안을 파악하고 소프트웨어 안전성 보장을 위한 관리적 측면의 기술현황을 분석하였다. 3장에서 소프트웨어 개발공정 각 단계별로 필요한 소프트웨어 안전성 분석행위를 살펴보았다. 4장에서는 계통 측면과 소프트웨어 측면에서의 안전성 분석 기술을 비교 분석하였고 5장에서는 소프트웨어 신뢰도 평가 기술을 파악하였다. 아직까지 소프트웨어 신뢰도의 정량적 평가 가능성 자체가 논란이 되고 있는 소프트웨어 신뢰도 분석 관련기술의 현황을 파악하였다. 6장에서는 디지털 보호계통 개발 시 가장 심각하게 대두되고 있는 공통모드 고장(CMF: Common Mode Failure)의 방어 기술을 분석하였다. 차후 액금로 I&C 계통 설계시 공통모드 고장을 방지하기 위한 방법들에 대한 기술현황 분석과 아울러 분석 지침을 제시하였다. 소프트웨어 고장 허용(Fault Tolerance) 방법, 가능한 CMF 원인 및 파급 효과, 소프트웨어로 인한 CMF 방어 대책 및 Guideline 변천과정, 미국 NRC의 4가지 규제입장, 적용 가능한 규제 법안 및 가이드라인 등에 대해서 기술 현황을 조사하였다.

주제명키워드 (10단어내외)	액금로 보호계통, 소프트웨어 안전성, 신뢰성, 평가기술, 공통모드고장, 다양성