UCRL-ID-122514

Software Safety Hazard Analysis

Version 2.0

Prepared by
J. Dennis Lawrence

Prepared for
U.S. Nuclear Regulatory Commission

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California and shall not be used for advertising or product endorsement purposes.

This work was supported by the United States Nuclear Regulatory commission under a Memorandum of Understanding with the United States Department of Energy, and performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Software Safety Hazard Analysis

Manuscript date: October 1995

Prepared by
J. Dennis Lawrence

Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550

Prepared for
U.S. Nuclear Regulatory Commission

ABSTRACT

Techniques for analyzing the safety and reliability of analog-based electronic protection systems that serve to mitigate hazards in process control systems have been developed over many years, and are reasonably well understood. An example is the protection system in a nuclear power plant The extension of these techniques to systems which include digital computers is not well developed, and there is little consensus among software engineering experts and safety experts on how to analyze such systems.

One possible technique is to extend hazard analysis to include digital computer-based systems. Software is frequently overlooked during system hazard analyses, but this is unacceptable when the software is in control of a potentially hazardous operation. In such cases, hazard analysis should be extended to fully cover the software. A method for performing software hazard analysis is proposed in this paper.

CONTENTS

Acknowledgment vii
Abbreviations viii
1. Introduction 1
1.1. Purpose 1
1.2. Report Structure 1
1.3. Terminology 2
2. Introduction to the Software Hazard Analysis Process 3
2.1. Software Hazard Analysis as Part of System Safety Analysis 3
2.2. Software Hazard Analysis as Part of Software Design 4
2.3. General Approach to Software Hazard Analysis 4
2.4. Prerequisites to Software Hazard Analysis 5
3. Requirements Hazard Analysis 8
3.1. Inputs to Software Requirements Hazard Analysis 14
3.2. Analysis Procedures 14
3.3. Outputs of Software Requirements Hazard Analysis 15
4. Architectural Design Hazard Analysis 15
4.1. Inputs to Software Architecture Hazard Analysis 16
4.2. Analysis Procedures 16
4.3. Outputs of Software Architecture Hazard Analysis 18
5. Detailed Design Hazard Analysis 18
5.1. Inputs to Software Detailed Design Hazard Analysis 19
5.2. Analysis Procedures 19
5.3. Outputs of Software Detailed Design Hazard Analysis 19
6. Code Hazard Analysis 19
6.1. Inputs to Software Code Hazard Analysis 20
6.2. Analysis Procedures 20
6.3. Outputs of Software Code Hazard Analysis 20
7. Summary and Conclusions 20
8. References 23
Appendix A. Background 25
A.1. Standards Review 25
A.2. General Discussion of Hazard Analysis 33
A.3. NIST Review of Software Hazard Analyses 38
A.4. Review of the Published Literature 39
Appendix B. Potential Software Safety Analysis Methods 41
Appendix C. Software Tools for Hazard Analysis 45
C.1. Fault Tree Analysis 45
C.2. FMEA, FMECA, HAZOP 46
C.3. Hazard Tracking 46
C.4. Markov Chain Modeling 47
C.5. Reliability Growth Modeling 47

FIGURES

Figure 1. Waterfall Life Cycle Model 6
Figure 2. Software Hazard Analysis within the Software Life Cycle 7
Figure 3. Hazard Severity Categories 9
Figure 4. Hazard Probability Levels 9
Figure 5. Example Matrix for Determining Risk 10
Figure 6. Software Qualities Relating to Potential Hazards 10
Figure 7. Guide Phrases for Software Qualities 11
Figure 8. Example of a Software Architecture 17
Figure 9. Determination of Architecture Risk Levels 18
Figure 10. Outline of a Software Safety Plan 26
Figure 11. IEEE 1228 Suggested Safety Analyses 27
Figure 12. Hazard Severity Categories (from Mil-Std 882C) 28
Figure 13. Hazard Probability Levels (from Mil-Std 882C) 28
Figure 14. Detailed Safety Tasks (from Mil-Std 882C) 29
Figure 15. Example Matrix for Residual Risk (from Mil-Std 882C) 29
Figure 16. Example Software Hazard Criticality Matrix (from Mil-Std 882C) 31
Figure 17. Summary of Safety System ACEs Identification 35
Figure 18. Summary of Software Requirements ACEs Identification 35
Figure 19. Summary of Software Design ACEs Identification 36
Figure 20. Summary of Software Code ACEs Identification 36
Figure 21. Summary of General Guidelines for ACE Resolution 37
Figure 22. Classes of Hazards (Hammer 1972) 38
Appendix C Figures: FaulTrEase 48
Appendix C Figures: HAZOPtimizer 55
Appendix C Figures: HazTrac 61
Appendix C Figures: CARMS 70
Appendix C Figures: CASRE 76

ACKNOWLEDGMENT

The author thanks and acknowledges the efforts of Mr. John Gallagher from the Nuclear Regulatory Commission who reviewed this work and provided insights and comments.

ABBREVIATIONS

ACE	Abnormal Condition or Event
CHA	Component Hazard Analysis
COTS	Commercial Off-the-Shelf
DOD	Department of Defense
ETA	Event Tree Analysis
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
FSE	Functions, Systems and Equipment
FTA	Fault Tree Analysis
HAZOP	Hazard and Operability Analysis
I&C	Instrumentation and Control
IEEE	Institute of Electronic and Electrical Engineers
MOD	Ministry of Defense
NPP	Nuclear Power Plant
NSCCA	Nuclear Safety Cross-Check Analysis
O&SHA	Operating and Support Hazard Analysis
PHA	Preliminary Hazard Analysis
PHL	Preliminary Hazard List
PIE	Postulated Initiating Event
PRA	Probabilistic Risk Assessment
RPS	Reactor Protection System
SAD	Software Architecture Description
SDD	Software Design Description
SHA	System Hazard Analysis
SRS	Software Requirements Specification
SSP	Software Safety Plan
SwHA	Software Hazard Analysis
V&V	Verification and Validation

SOFTWARE SAFETY HAZARD ANALYSIS

1. INTRODUCTION

1.1. Purpose

Techniques for analyzing the safety and reliability of analog-based electronic protection systems that serve to mitigate hazards in process control systems have been developed over many years, and are reasonably well understood. An example is the protection system in a nuclear power plant. The extension of these techniques to systems which include digital computers is not well developed, and there is little consensus among software engineering experts and safety experts on how to analyze such systems.

One possible technique is to extend hazard analysis to include digital computer-based systems. If safety is considered to be a measure of the degree of freedom from risk, then *software safety* ensures that the software in the computer system will execute within the application system without resulting in unacceptable risk. Hazard analysis is a method of identifying portions of a system which have the potential for unacceptable hazards; the purpose is to (1) encourage design changes which will reduce or eliminate hazards, or (2) carry out special analyses and tests which can provide increased confidence in especially vulnerable portions of the system.

Software is frequently overlooked during system hazard analyses, but this is unacceptable when the software is in control of a potentially hazardous operation. In such cases, hazard analysis should be extended to fully cover the software. A method for performing software hazard analysis is proposed in this paper.[1]

The report considers only those hazards affected by software. Only the software portion of the digital computer system is considered. In particular, it is assumed that the computer hardware operates without failure.[2]

As a consequence of the above assumptions, the report concentrates on two questions.

- If the software operates correctly (i.e., follows its specifications), what is the potential effect on system hazards?
- If the software operates incorrectly (i.e., deviates from specifications), what is the potential effect on system hazards?

This report does not discuss how to determine whether a software item is correct or not. Software analyses, reviews and tests directed at finding faults in the software are not considered to be a direct part of software hazard analysis. See Lawrence (1993) for a discussion of these Verification and Validation (V&V) topics within the software life cycle.

Although V&V is not considered to be part of hazard analysis, the results of a V&V effort may well be of use. For example, the use of testing to estimate the reliability of a software item might be used within a fault tree analysis to estimate the probability of a hazard occurring.

The performance of software hazard analysis can be facilitated by the use of automated or semi-automated tools. Examples of such tools are considered in Appendix C.

1.2. Report Structure

Software hazard analysis is discussed in general terms in Chapter 2. This chapter includes a list of desirable prerequisites to software hazard analysis, and a general discussion of the approach proposed in the remainder of the report.

Chapters 3-6 provide the details of the proposed software hazard analysis process. Considerable emphasis is placed on the requirements and

[1] Neither this proposed method of hazard analysis nor any other specific method has been endorsed by the U. S. Nuclear Regulatory Commission.

[2] A separate hardware hazard analysis and—for complex computer systems—a separate computer system hazard analysis, are recommended to supplement the software hazard analysis.

architecture design phases of the software life cycle to reflect the belief that faults in requirements and architecture design specifications have a greater potential impact on system hazards than faults in the detailed design or coding phases.

Tool support can be very helpful when performing hazard analyses. A representative set of tools is discussed briefly in Appendix C. The goal here is to indicate the availability of different types of tools. The tools were selected for discussion based on availability on a PC platform and on price. No endorsement of specific tools is implied.

The software hazard analysis process proposed in this report is based on a variety of standards and technical papers described in Appendix A. The report continues with a list of possible safety analysis techniques taken from a System Safety Society report (Appendix B).

1.3. Terminology

Safety engineering has special terminology of its own. The following definitions, based primarily on those contained in IEEE Standard 1228, are used in this report. They are reasonably standard definitions that have been specialized to computer software in a few places.

- An *accident* is an unplanned event or series of events that result in death, injury, illness, environmental damage, or damage to or loss of equipment or property. (The word *mishap* is sometimes used to mean an accident, financial loss or public relations loss.)

  Accidents generally can be divided into two categories: those that involve the unplanned release of energy and those that involve the unplanned release of toxicity.

- A *system hazard* is an application system condition that is a prerequisite to an accident.

  That is, the system states can be divided into two sets. No state in the first set (of *nonhazardous states)* can directly lead to an accident, while accidents may result from any state in the second set (of *hazardous states*). Note that a system can be in a hazardous state without an accident occurring. It is the potential for causing an accident that creates the hazard, not necessarily the actuality, because conditions that convert the hazard to an accident are not concurrently present. A hazard is a potential for an accident that can be converted to actuality by encountering a triggering event or condition within the foreseeable operational envelope of the system.

- The term *risk* is used to designate a measure that combines the likelihood that a system hazard will occur, the likelihood that an accident will occur and an estimate of the severity of the worst plausible accident.

  The simplest measure is to simply multiply the probability that a hazard occurs, the conditional probability that a triggering event or condition will occur while the hazard is present, and the estimated worst-case severity of the accident.

- *Safety-critical software*[3] is software whose inadvertent response to stimuli, failure to respond when required, response out-of-sequence, or response in unplanned combination with others can result in an accident or the exacerbation of an accident. This includes software whose operation or failure to operate can lead to a hazardous state, software intended to recover from equipment malfunctions or

[3]The word “critical,” as used in this report, refers to software criticality, not nuclear criticality.

external insults, and software intended to mitigate the severity of, or recover from, an accident.[4]

- A *critical system* is a system whose failure may lead to unacceptable consequences. The results of failure may affect the developers of the system, its direct users, their customers or the general public. The consequences may involve loss of life or property, financial loss, legal liability, regulatory actions or even the loss of good will if that is extremely important. The term *safety critical* refers to a system whose failure could lead to loss of life, injury, or damage to the environment. For nuclear reactors, this includes radiation releases or exposure to the public or operators and reactor workers.
- The term *safety* is used to mean the extent to which a system is free from system hazard.
- *Hazard analysis* is the process of identifying and evaluating the hazards of a system, and then either eliminating the hazard or reducing its risk to an acceptable level. (NIST 1993)
- *Software hazard analysis* ". . . eliminates or controls software hazards and hazards related to interfaces between the software and the system (including hardware and human components). It includes analyzing the requirements, design, code, user interfaces and changes." (NIST 1993)

2. INTRODUCTION TO THE SOFTWARE HAZARD ANALYSIS PROCESS

2.1. Software Hazard Analysis as Part of System Safety Analysis

Software hazard analysis should be performed within the context of the overall system design, for both those attributes of the system design that contribute to the system's ability to perform the assigned tasks that are derived from the plant's safety mission as well as the assigned tasks 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 also in the performance of the system control and monitoring functions, and the effect of the software acting within the system with respect to its potential impact on the accomplishment of the plant's safety mission. This understanding is obtained from the system safety analysis; in particular, the system's hazard analysis. IEEE Standard 1228 presents the relationship between the system safety analysis and the software safety analysis in more detail. The following discussion provides an overview of the safety case for a nuclear power plant.

The safety properties of a nuclear reactor design are fundamentally affected by three broad design principles: quality, diversity and defense-in-depth. These principles may be applied at various levels of the design; determining where and how to apply the principles is one of the more important activities of the design process. All three principles should have wide applicability to other forms of process control systems.

The main hazards in a nuclear reactor are the possibility of a rapid, energetic fission reaction (e.g., Chernobyl) and the release of radioactive fission products which are the waste products of the fission reaction. In the U.S. (and many other countries), regulations mandate that the physics of the core design make rapid reactions self limiting. This leaves the prevention of the release of fission products as the main hazard to be controlled.

Three levels of defense-in-depth are provided to control the release of fission products. Each is sufficient to prevent public exposure to any significant level of radiation. First, each element of the fuel is surrounded by a barrier. In light water reactors, the fuel is composed of

[4] There are cases in which software other than control system software can cause a hazard. An example is a fire suppression modeling program which makes incorrect recommendations for fire suppression equipment, causing a hazard in case of fire due to lack of the necessary fire fighting equipment. Such cases are outside the scope of this report.

numerous metal tubes, each tube containing fuel pellets and associated fission products. Second, fission products that might be released from the fuel are further contained by the reactor coolant systems. Should some event cause breach of both of these barriers, a third barrier, the containment building, surrounds the reactor coolant system. Each of these barriers is fundamentally different in design, providing diversity at each level of defense-in-depth.

Barrier integrity is maintained first by building in a high degree of quality, and second by ensuring the barriers are not exposed to environmental conditions that exceed design assumptions. Active systems are provided to enforce these environmental limits. For example, the most important environmental considerations for the fuel are that the heat generated by the fuel be limited, and that the heat that is generated be taken away. These are the safety functions that must be accomplished to ensure the barrier (fuel clad) immediately surrounding the fuel and fission products remains intact. Diversity and defense-in-depth are provided for these functions. For example, power can be limited by dropping solid neutron absorbers (control rods) or injecting liquid absorber into the coolant system.

Each function can be actuated by multiple independent systems. For example, the control rods may be inserted automatically by the control system, the reactor protection system, the ATWS (Anticipated Transient Without Scram) Mitigation System, the Engineered Safety Actuation System (ESFAS), or the reactor operator. In proposed U. S. advanced reactor designs that involve computer-based control and protection systems, at least two diverse, automatic systems must be capable of initiating each safety function such that the consequences of each postulated accident are acceptable. Furthermore, sufficient information and manual controls must be provided to allow the operator to start and control each safety function.

This diversity may be accomplished via diversity in the computer systems. Hardware diversity may include multiple CPU types and multiple communication paths. Software diversity could involve independent calculations of the process state using different types of information—temperature and pressure calculations in one piece of software compared to pressure and neutron flux in another piece—either of which is sufficient to determine, in time, if backup systems must be started to achieve safety functions.

Finally, quality parts and design are used to reduce the probability of any individual failures from occurring.

2.2. Software Hazard Analysis as Part of Software Design

The ultimate objectives of any hazard analysis program are to identify and correct deficiencies and to provide information on the necessary safeguards. This is certainly true of Software Hazard Analysis. There is no point to the analysis unless appropriate action is taken. At least four types of actions may be appropriate, depending on the circumstances:

1. The system design may be changed to eliminate identified hazards which are affected by software or are not adequately handled by software, or to reduce the hazards to acceptable levels, or to adjust the system architecture so that identified hazards are compensated by defense-in-depth.
2. The software design may be changed to eliminate identified hazards, or reduce them to acceptable levels.
3. The quality of the software may be improved sufficiently to reduce the probability of a hazard to an acceptable level.
4. The application system may be rejected if it is considered too hazardous.

2.3. General Approach to Software Hazard Analysis

Software hazard analysis should be a defined aspect of the software life cycle. No specific life

cycle is endorsed here (see Lawrence (1993) for a discussion of life cycles). To provide some specificity to the discussion, a waterfall life cycle is assumed, as shown in Figure 1. Not all the phases in the figure are included in the hazard analysis.

Hazard analysis begins with analyses of the reactor design which identify parameter limits of the safe operating region for the thermal-hydraulic properties of the reactor. This provides a variety of documents which serve as the starting point for the software hazard analysis. The general approach is shown in Figure 2, which shows the technical development activities (requirements, architecture, design, code), the V&V activities, and the hazard analysis activities. Results of the various software hazard analyses are used, as appropriate, to change the protection system design, change the software architecture or design, and to identify portions of the software which require increased attention to quality.

This report does not discuss methods or techniques for performing the recommended hazard analyses. Little extensive experience with analysis techniques has been reported in the literature. Hazard and Operability Analysis (HAZOP), Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) are possibilities (see Appendix A). Other potential possibilities are listed in Appendix B.

2.4. Prerequisites to Software Hazard Analysis

Considerable work is required before a software hazard analysis process can begin. The following list will generally require some modifications to fit specific projects. Since iterations of analyses are necessary as the software development proceeds, no strict chronological rigidity 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 steps 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 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 (see Appendix A.1.4 for a discussion). This method assigns a level code of A, B or C to each hazard, where “A” is assigned to the most critical software.

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 (see IEEE 1228, discussed in Appendix A.1.1). 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.

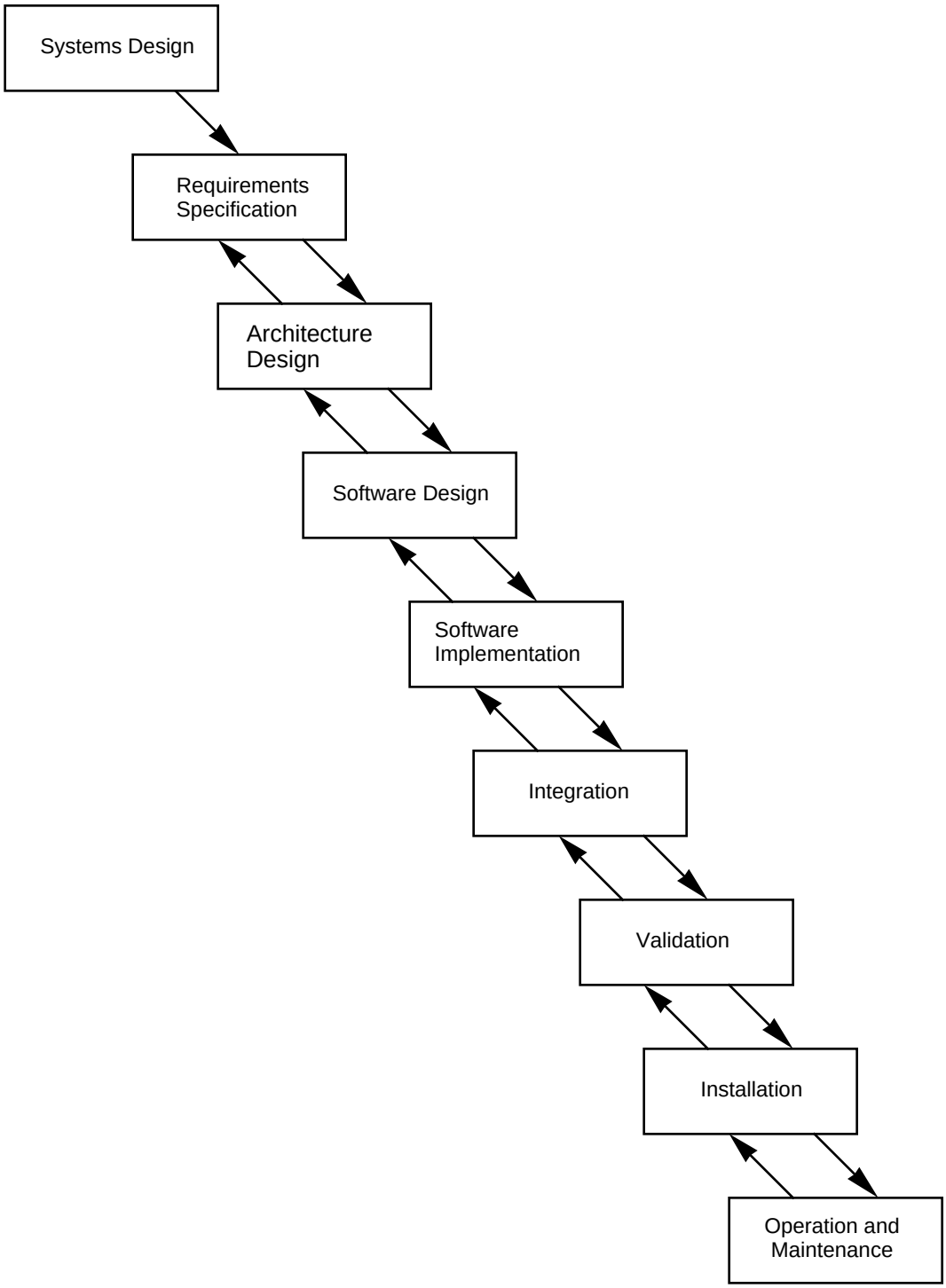


Figure 1. Waterfall Life Cycle Model

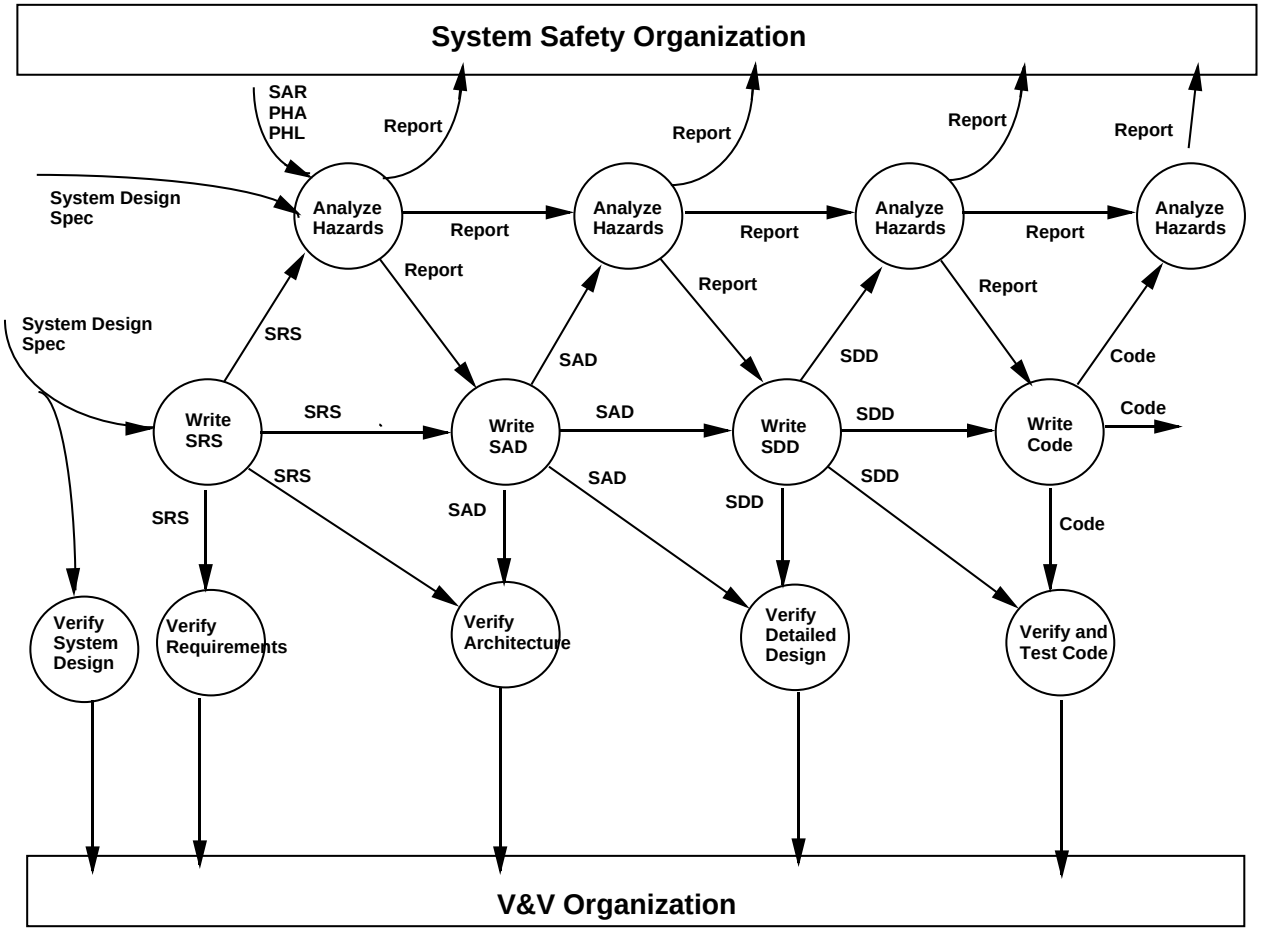


Figure 2. Software Hazard Analysis within the Software Life Cycle

Abbreviations

PHA	Preliminary Hazard Analysis
PHL	Preliminary Hazard Analysis
SAD	Software Architecture Description
SAR	Safety Analysis Report
SDD	Software Design Description
SRS	Software Requirements Specification

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 of this system 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 tables shown in Figures 3 and 4 can be used as a basis for this. These tables are based on IEC 1226 and MilStd 882C, and are discussed in Appendix A.1.4 and A.1.2, respectively.

5. Prepare a table like that in Figure 5 from the tables created in step 4. This table can be used to derive an estimate of risk for each hazard.

   This table matches the hazard severity categories of Figure 3 to the hazard probability levels of Figure 4 to obtain a measure of overall risk. Thus, events with critical severity and occasional probability of occurrence are judged to have high risk.

6. For each hazard identified in the PHL, PHA or other hazard analyses, identify its risk level using the table prepared in step 5.

7. Prepare an application system requirements specification.

8. 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 coordinate to address the hazards discovered in previous analyses.

9. Prepare the remaining documents to the extent required in order to specify, design, implement, verify and analyze the software component of the RPS. 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 are elaborated.

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* here. A recommended list of qualities to be considered during software hazard analysis is given in Figure 6. 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. McDermid et al. (1994, 1995) suggest the use of guide words to assess impact; this idea is adapted here. A set of guide phrases is supplied for each quality that can be used to help assess the impact on hazards of each requirement associated with the quality.

These guide phrases are shown in Figure 7. This figure suggests concepts to be examined for each requirement that relates to specific software qualities. In some cases, a requirement may affect more than one quality. The figure lists the various qualities; in some cases, these are further divided into aspects of the quality. The third column contains a code for the life cycle phase during which use of the guide phrase is recommended.

Letters are:

R Requirements

A Architectural Design

D Detailed Design

C Coding

The last column contains the guide phrases. In addition to the phrases listed, the analyst should examine the impact on hazards if the requirement is actually met.

Description	Category	Definition
Catastrophic	A	Death, system loss, or severe environmental damage
Critical	B	Severe injury, severe occupational illness, major system or environmental damage
Marginal	C	Minor injury, minor occupational illness or minor system or environmental damage
Negligible	--	Less than minor injury, occupational illness or less than minor system or environmental damage

Figure 3. Hazard Severity Categories (based on IEC 1126)

Description	Level	Estimate of Probability
Frequent	A	Likely to occur frequently
Probable	B	Will occur several times in the life of an item
Occasional	C	Likely to occur some time in the life of an item
Remote	D	Unlikely but possible to occur in the life of an item
Improbable	E	So unlikely, it can be assumed occurrence may not be experienced

Figure 4. Hazard Probability Levels (based on Mil-Std 882C)

	Hazard Category			
Frequency	Catastrophic	Critical	Marginal	Negligible
Frequent	High	High	High	Medium
Probable	High	High	Medium	Low
Occasional	High	High	Medium	Low
Remote	High	Medium	Low	Low
Improbable	Medium	Low	Low	Low

Figure 5. Example Matrix for Determining Risk

Quality	Description of Quality
Accuracy	The term *accuracy* denotes the degree of freedom from error of sensor and operator input, the degree of exactness possessed by an approximation or measurement, and the degree of freedom of actuator output from error.
Capacity	The terms *capacity* denotes the ability of the software system to achieve its objectives within the hardware constraints imposed by the computing system being used. The main factors of capacity are Execution Capacity (timing) and Storage Capacity (sizing). These refer, respectively, to the availability of sufficient processing time and memory resources to satisfy the software requirements.
Functionality	The term *functionality* denotes the operations which must be carried out by the software. Functions generally transform input information into output information in order to affect the reactor operation. Inputs may be obtained from sensors, operators, other equipment or other software as appropriate. Outputs may be directed to actuators, operators, other equipment or other software as appropriate.
Reliability	The term *reliability* denotes the degree to which a software system or component operates without failure. This definition does not consider the consequences of failure, only the existence of failure. Reliability requirements may be derived from the general system reliability requirements by imposing reliability requirements on the software components of the application system which are sufficient to meet the overall system reliability requirements.
Robustness	The term *robustness* denotes the ability of a software system or component to function correctly in the presence of invalid inputs or stressful environmental conditions. This includes the ability to function correctly despite some violation of the assumptions in its specification.
Safety	The term *safety* is used here to denote those properties and characteristics of the software system that directly affect or interact with system safety considerations. The other qualities discussed in this table are important contributors to the overall safety of the software-controlled protection system, but are primarily concerned with the internal operation of the software. This quality is primarily concerned with the affect of the software on system hazards and the measures taken to control those hazards.
Security	The term *security* denotes the ability to prevent unauthorized, undesired and unsafe intrusions. Security is a safety concern in so far as such intrusions can affect the safety-related functions of the software.

Figure 6. Software Qualities Relating to Potential Hazards

Quality	Aspect	Phase	Guide Phrases
Accuracy	Sensor	RADC	Stuck at all zeroes
		RADC	Stuck at all ones
		RADC	Stuck elsewhere
		RADC	Below minimum range
		RADC	Above maximum range
		RADC	Within range, but wrong
		RADC	Physical units are incorrect
		RADC	Wrong data type or data size
	Actuator	RADC	Stuck at all zeroes
		RADC	Stuck at all ones
		RADC	Stuck elsewhere
		RADC	Below minimum range
		RADC	Above maximum range
		RADC	Within range, but wrong
		RADC	Physical units are incorrect
		RADC	Wrong data type or data size
	Operator Input & Output	RA	Numerical value below acceptable range
		RA	Numerical value above acceptable range
		RA	Numerical value within range, but wrong
		RA	Numerical value has wrong physical units
		RA	Numerical value has wrong data type or data size
		RA	Non-numerical value incorrect
	Calculation	RDC	Calculated result is outside acceptable error bounds (too low)
		RDC	Calculated result is outside acceptable error bounds (too high)
		RDC	Formula or equation is wrong
		RDC	Physical units are incorrect
		RDC	Wrong data type or data size

Figure 7. Guide Phrases for Software Qualities

Capacity	Message	RADC	Message volume is below stated minimum
		RADC	Message volume exceeds stated maximum
		RADC	Message volume is erratic
		RADC	Message rate is below stated minimum
		RADC	Message rate exceeds stated maximum
		RADC	Message rate is erratic
		RADC	Message contents are incorrect, but plausible
		RADC	Message contents are obviously scrambled
	Timing	RADC	Input signal fails to arrive
		RADC	Input signal occurs too soon
		RADC	Input signal occurs too late
		RADC	Input signal occurs unexpectedly
		RADC	System behavior is not deterministic
		RADC	Output signal fails to arrive at actuator
		RADC	Output signal arrives too soon
		RADC	Output signal arrives too late
		RADC	Output signal arrives unexpectedly
		R	Insufficient time allowed for operator action
		AD	Processing occurs in an incorrect sequence
		DC	Code enters non-terminating loop
		DC	Deadlock occurs
		C	Interrupt loses data
		C	Interrupt loses control information
Functionality		RA	Function is not carried out as specified (for each mode of operation)
		RA	Function is not initialized properly before being executed
		RA	Function executes when trigger conditions are not satisfied
		RA	Trigger conditions are satisfied but function fails to execute
		RA	Function continues to execute after termination conditions are satisfied
		RA	Termination conditions are not satisfied but function terminates
		RA	Function terminates before necessary actions, calculations, events, etc. are completed
		R	Function is executed in incorrect operating mode
		R	Function uses incorrect inputs
		R	Function produces incorrect outputs

Figure 7. Guide Phrases for Software Qualities, continued

Reliability	RA	Software is less reliable than required
	RA	Software is more reliable than required
	RA	Software reliability is not known when the system goes into production use
	RA	Software does not degrade gracefully when required (crashes instead)
	RA	Software fault tolerance requirements (if any) are not met
	RA	Reliability varies among the different modes of operation
	R	Software fails in-service test
	R	Software fails
	A	Hardware unit fails
	A	Software failure propagates to uninvolved processes
	A	Software fails to recover from failure
	A	Hardware or software failure is not reported to operator
	A	Software fails to detect inappropriate operation action
	AD	Data is passed to incorrect process
Robustness	RA	Software fails in the presence of unexpected input data
	RA	Software fails in the presence of incorrect input data
	RA	Software fails when anomalous conditions occur
	RA	Software fails to recover itself when required
	RA	Software fails during message overload
	RA	Software fails when messages missed
Safety	RA	Software causes system to move to a hazardous state
	RA	Software fails to move system from hazardous to nonhazardous state
	RA	Software fails to initiate emergency shutdown when required to do so
	RA	Software fails to recognize hazardous reactor state
Security	RA	Unauthorized person has access to software system
	RA	Unauthorized changes have been made to software
	RA	Unauthorized changes have been made to plant data

Figure 7. Guide Phrases for Software Qualities, continued

Numerous traditional qualities generally considered necessary to an adequate software requirements specification are not included in Figure 7. 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.

For example, the first quality is sensor accuracy. Suppose there were an accuracy requirement for a particular sensor that "The value from sensor 123 shall be between 100 and 500, with an error of no more than 5%." Then, the following questions should be asked:

- What is the effect on hazards if the sensor reading satisfies the requirement? In particular, what if the reading is 5% away from the actual value?
- What is the effect on hazards if the sensor is stuck at all zeros?
- What if the sensor is stuck at all ones?
- What if the sensor value is less than 100?
- What if the sensor value is greater than 500?
- What if the sensor value is between 100 and 500, but is not within 5% of the actual value?

It is important that this analysis not be sidetracked into asking how such conditions might occur, or into arguments on the impossibility of the conditions. For hazard analysis, assume that the conditions can occur, and examine the consequences.

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
- Protection System Design Description
- Software Requirements Specification

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, using the table in Figure 5.
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. Analyze each requirement using the guide phrases in Figure 7 which are marked with an "R." These guide phrases are meant to initiate discussion and suggest possibilities to consider, not to bound the analysis.

   There are a great many phrases in Figure 7. For any particular requirement, most of these will not apply. For example, only about eight of the phrases would apply to the example given at the beginning of Section 3. Part of the analysis of this step is to select the quality or qualities that apply to the requirement, so that only applicable phrases are used.
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 and the guide phrase analysis provides information on 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. In particular, 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 the following analysis steps.

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) (see Appendix A.4 for a discussion). Event Tree Analysis (ETA) should also be considered, using the guide phrases 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. 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.

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 in this chapter.

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.

The analysis here builds on the 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 (hardware, software and mapping).

For example, suppose there is a timing requirement that a certain signal be sent to a particular valve actuator within five seconds of receiving an overload signal from a particular sensor. This requirement would have been analyzed as part of the software requirements hazard analysis. Now, suppose the logical data path is as shown in Figure 8. Among other guide phrases that apply, the analyst should consider the effect on the hazard if the message to be sent on path “c” never arrives. In this instance, a hazard that did not exist previously has been added by the decision to implement the logical data path “c.”

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

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. Figure 9 shows one method of doing this. The figure uses the risk levels taken from Figure 5, and considers the number of requirements of various risk levels affected by the element in order to assign a risk to the element. The suggested algorithm is as follows:

   a. Pick one requirement. Assign the architectural element severity level to be the same as that of the requirement. If the requirement has medium severity, for example, then the initial element level is also “medium.”

   b. 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.

   c. 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. Analyze each safety-critical architectural element using the guide phrases in Figure 7 marked “A.” These phrases are meant to initiate discussion and suggest possibilities to consider, not to bound the analysis.

   As with the requirements discussion in Section 4.2, there are a great many phrases in Figure 7 marked “A.” For any particular architectural element, many of these will not apply. Part of the analysis of this step is to select the quality or qualities that apply to the architectural element, so that only applicable phrases are used.

4. Document the results of the analysis.

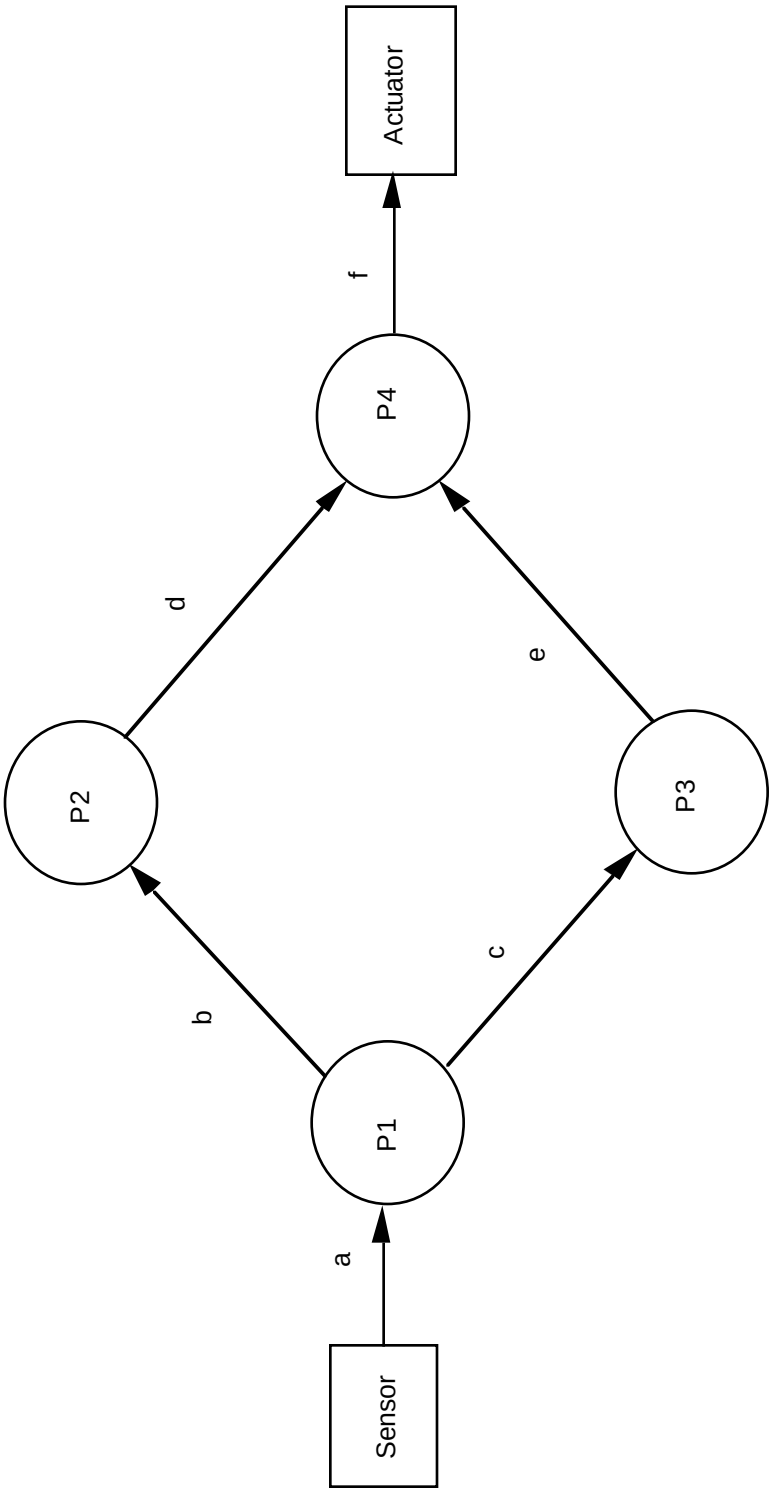


Figure 8. Example of a Software Architecture

Architecture Element Risk Level	Risk Level from Adding a Requirement		
	High	Medium	Low
Very High	Very High	Very High	Very High
High	Very High	High	High
Medium	High	Medium	Medium
Low	High	Medium	Low

Figure 9. Determination of Architecture Risk Levels

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.

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 which are required 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.

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.

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

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, use the guide phrases in Figure 7 that are marked “D” to 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 analyze 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.

5.3. Outputs of Software Detailed Design Hazard Analysis

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

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.

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

6.2. Analysis Procedures

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

1. For each code element, use the guide phrases in Figure 7 that are marked “C” 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 analyze 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.

6.3. Outputs of Software Code Hazard Analysis

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

7. SUMMARY AND CONCLUSIONS

The software hazard analysis described in Sections 3-6 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 recognized 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. A detailed discussion of the life cycle activities for the development of safety-critical software is given in Lawrence (1993). A summary of design factors important to the realization of high-quality software that is “fit for duty” in safety-critical applications is given in Lawrence (1994).

With the above view in mind, one can then consider where software hazards analysis offers a unique capability to improve the integrity of safety-critical software. Section 2 provides an overview of the objectives of the hazards analysis activities for both system hazards and software hazards and the relation between hazards analysis activities and other principal software development life cycle activities. 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 defense-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 discussed in Section 5. The emphasis on correctness is even more true for the software code. The discussion in Section 6 provides guidance on matters that are more effectively dealt with through correctness concerns.

The more detailed presentation of the software hazards analysis in Section 3, Requirements Hazards Analysis, includes an approach to guide the assessment of the impact on hazards of each requirement as it is related to the qualities given in Figure 6. The guide phrases for this assessment are presented in Figure 7.

The selection of applicable guide phrases to particular requirements must be governed by the potential impact of each software hazard on the system, as presented in item 3 of section 2.4 and the accompanying risk associated with that hazard, as given in Figure 5. Similar selection considerations are applicable for the architectural design hazards analysis described in Section 4.

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.

The process outlined in Chapters 3-6 is based on the documents listed in the References. It has not been tested or evaluated in the field.

Section 7. Summary and Conclusions

8. REFERENCES

Air Force Inspection and Safety Center. 1985. *Software System Safety*. Headquarters Air Force Inspection and Safety Center, AFISC SSH 1-1, September 5.

Bowman, William C., Glenn H. Archinoff, Vijay M. Raina, David R. Tremaine and Nancy G. Leveson. 1991. "An Application of Fault Tree Analysis to Safety Critical Software at Ontario Hydro." *Probabilistic Safety Assessment and Management*, G. Apostolakis, ed. (1991): 363-368.

Brown, Michael L. 1985. "Software Safety for Complex Systems. *IEEE Annual Conf. of the Eng. in Medicine and Biology Society* (1985): 210-216.

Center for Chemical Process Safety. 1992. 1992. *Guidelines for Hazard Evaluation Procedures*, American Institute of Chemical Engineers.

Clarke, Stephen J. and John A. McDermid. 1993. "Software Fault Trees and Weakest Preconditions: a Comparison and Analysis." *Soft. Eng. J*. 8, no. 4 (July): 225-236.

Connolly, Brian. 1989. "Software Safety Goal Verification Using Fault Tree Techniques: A Critically Ill Patient Monitor Example." *Computer Assurance Conf. (Compass)* (June): 18-21.

Elahi, B. John. 1993. "Safety and Hazard Analysis for Software Controlled Medical Devices." *Sixth Annual IEEE Symp. On Comp.-Based Med. Syst* (June): 10-15.

Froome, P. K. D. 1992. Interim Def Stan 00-56: Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defense Equipment," *Centre for Software Reliability, Ninth Annual Conf. on Soft. Safety*, Luxembourg (April): 1-14.

Gowen, Lon D., James S. Collofello and Frank W. Calliss. 1992. "Preliminary Hazard Analysis for Safety-Critical Software Systems," *Phoenix Conf. Comp. and Comm.* (April): 501-508.

Hammer, Willie. 1972. *Handbook of System and Product Safety*. Prentice-Hall.

IEC 1226. 1993. *Nuclear Power Plants—Instrumentation and Control Systems Important for Safety—Classification*. International Electrotechnical Commission.

IEEE 7-4.3.2. 1993. *IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*. Institute of Electronic and Electrical Engineers.

IEEE 1228. 1994. *IEEE Standard for Software Safety Plans.* Institute of Electronic and Electrical Engineers.

Lal-Gabe, Anshoo. 1990. "Hazards Analysis and its Application to Build Confidence in Software Test Results." *Third Annual IEEE Symp. On Comp.-Based Med. Syst.* (June): 129-134.

Lawrence, J. Dennis. 1993. *Software Reliability and Safety in Nuclear Reactor Protection Systems*. NUREG/CR-6101, UCRL-ID-114839, Lawrence Livermore National Laboratory, November.

Lawrence, J. Dennis and G. Gary Preckshot. 1994. *Design Factors for Safety-Critical Software*, NUREG/CR-6294, Lawrence Livermore National Laboratory, December.

Levan, David G. 1992. "Preliminary Procedure for Software Hazard Analysis of Safety-Critical Software." Prepared for Ontario Hydro Nuclear Safety Department by DLSF Systems, Inc., January.

Leveson, Nancy G. and Peter R. Harvey. 1983. "Analyzing Software Safety." *IEEE Trans. Soft. Eng.* 9, no. 5 (September): 569-579.

Leveson, Nancy G. 1991a. "Safety Assessment and Management Applied to Software." *Probabilistic Safety Assessment and Management*, G. Apostolakis, ed. 377-382.

Leveson, Nancy G. 1991b. "Safety." *Aerospace Software Engineering*. Christine Anderson and Merlin Dorfman, ed., AIAA, 319-336.

Leveson, Nancy G., Stephen S. Cha and Timothy J. Shimeall, 1991c. “Safety Verification of Ada Programs Using Software Fault Trees.” *IEEE Software* (July): 48-59.

Levinson, Stanley H. and H. Tazewell Doughtrey. 1993. “Risk Analysis of Software-Dependent Systems.” *Prob. Safety Assessment Int’l. Topical Meeting* (January).

McDermid, J. A., and D. J. Pumfrey. 1994.”A Development of Hazard Analysis to Aid Software Design.” *Computer Assurance (Compass)* (June): 17-25.

McDermid, John A., M. Nicholson, D. J. Pumfrey and P. Fenelon. 1995. “Experience with the Application of HAZOP to Computer-Based Systems.” *Computer Assurance (Compass)* (June): 37-48

McKinlay, Archibald. 1991. “The State of Software Safety Engineering.” *Probabilistic Safety Assessment and Management*, G. Apostolakis, ed. (1991): 369-376.

Mil-Hdbk-764. 1990. *System Safety Engineering Design Guide for Army Materiel*. U.S. Dept. of Defense (January 12).

Mil-Std 882C. 1993. *System Safety Program Requirements*. U.S. Dept. of Defense (January).

MOD 00-56. 1991. *Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defense Equipment*. Defence Standard 00-56, Ministry of Defence, Great Britain (April).

Mojdehbakhsh, Ramin, Satish Subramanian, Ramakrishna Vishnuvajjala, Wei-Tek Tsai and Lynn Elliott. 1994. “A Process for Software Requirements Safety Analysis.” *Int’l Symp. on Software Reliability Engineering*, (November): 45-54.

NIST 1993. *Review of Software Hazard Analyses*. National Institutes of Standards and Technology. Draft (June 4).

Stephans, Richard A. and Warner W. Talso. 1993. *System Safety Analysis Handbook*. System Safety Society, New Mexico Chapter, Albuquerque, NM, July.

APPENDIX A. BACKGROUND

A.1. Standards Review

A.1.1. IEEE 1228, Standard for Software Safety Plans

IEEE 1228 Standard, *Standard for Software Safety Plans*, "describes the minimum acceptable requirements for the content of a Software Safety Plan." A Software Safety Plan developed to satisfy this Standard will contain information on both management and technical aspects of the development activity. The recommended contents of a Software Safety Plan, as given by the Standard, are shown in Figure 10.

Only the analyses which are required in Sections 4.2-4.4 of the Safety Plan (sections 4.4.2-4.4.4 of the Standard) are relevant to the scope of this report. The Standard itself does not require any particular types of analyses. It does contain an appendix which lists some suggested analyses.

The Standard distinguishes between the Application System and the Software System. In the context of reactors, for example, the application system might be the entire reactor, or the entire reactor protection system, and the software system is the software which is contained in the reactor control system or reactor protection system, respectively. The word "system," used here without modification, will always refer to the entire application system.

The Standard assumes that certain information will be available prior to performing any safety analyses. This information is listed next.

1. A Preliminary Hazard Analysis (PHA) and any other hazard analyses which have been performed on the entire application system or any portion of it should be available. These analyses must include the following information:
    a. Hazardous application system states.
    b. Sequences of actions that can cause the application system to enter a hazardous state.
    c. Sequences of actions intended to return the application system from a hazardous state to a nonhazardous state.
    d. Actions intended to mitigate the consequences of an accident.
2. A high-level application system design should exist and specify:
    a. The functions which will be performed by the software contained in the application system.
    b. Any actions that will be required of the software in order to prevent the application system from entering a hazardous state.
    c. Any actions that will be required of the software in order to move the application system from a hazardous state to a nonhazardous state.
    d. Any actions that will be required of the software to mitigate the consequences of an accident.
3. The interfaces between the software and the rest of the application system should be completely defined.
4. A software safety plan (SSP) should exist. It will describe fully the means by which software safety analyses will be carried out for the application system. IEEE Standard 1228 may be used as a model for the SSP. If the developer prefers, the software safety plan may be included in the general system safety plan.

These will be referred to as the System Hazard Analyses, the System Design, the System Interface Specification and the Software Safety Plan, respectively.

The Appendix to the Standard suggests certain analyses which may be performed during software development. These are shown in Figure 11.

1. Purpose
2. Definitions, Acronyms and References
3. Software Safety Management
- 3.1. Organization and Responsibilities
- 3.2. Resources
- 3.3. Staff Qualifications 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. Startup and Transition
  - 5.2.3. Operations Support
- 5.3. Monitoring
- 5.4. Maintenance
- 5.5. Retirements and Notification

6. Plan Approval

Figure 10. Outline of a Software Safety Plan

Software Safety Requirements Analysis
Criticality Analysis
Specification Analysis
Timing and Sizing Analysis
Software Safety Design Analysis
Logic Analysis
Data Analysis
Interface Analysis
Constraint Analysis
Functional Analysis
Module Analysis
Revised Timing and Sizing Analysis
Software Safety Code Analysis
Logic Analysis
Data Analysis
Interface Analysis
Constraint Analysis
Programming Style Analysis
Non-critical Code Analysis
Revised Timing and Sizing Analysis
Software Safety Test Analysis
Unit Test Analysis
Interface Test Analysis
Subsystem Test Analysis
System-level Test Analysis
Stress Test Analysis
Regression Test Analysis

Figure 11. IEEE 1228 Suggested Safety Analyses

A.1.2. Mil-Std 882C, System Safety Program Requirements

This Standard applies to all military systems in which safety is a factor. The Standard is directed at DoD program managers, and is meant to assist them in overseeing contractors. The contractors are expected to carry out the safety program.

The Standard defines hazard severity categories which provide a qualitative measure of the worst credible accident. These are shown in Figure 12. A second table, reproduced here in Figure 13, categorizes the probability that a hazard will be created during the planned life expectancy of the system. This latter table is also qualitative, and is given both in terms of specific individual items, and in terms of all items in the inventory. The Standard points out that the two tables may need to be modified in some cases to fit individual situations.

The Standard presents detailed requirements as tasks. These are organized into four sections, with specific tasks in each section. This grouping is intended to facilitate understanding, and does not imply that the tasks are to be carried out in the order listed. The task sections and individual tasks are listed in Figure 14.

It is possible to combine Figures 12 and 13 to show a hazard risk level. One way of doing this is shown in Figure 15. This latter table can be used in a hazard analysis in order to manage risk. For example, if a hazard falls in the “high” risk category, it might be possible to redesign the system or use better quality parts in order to move to a “medium” risk category. Figure 15 can also be used to determine where assessment resources should be concentrated.

Description	Category	Definition
Catastrophic	I	Death, system loss, or severe environmental damage
Critical	II	Severe injury, severe occupational illness, major system or environmental damage
Marginal	III	Minor injury, minor occupational illness or minor system or environmental damage
Negligible	IV	Less than minor injury, occupational illness or less than minor system or environmental damage

Figure 12. Hazard Severity Categories (from Mil-Std 882C)

Description	Level	Specific Individual Item	Fleet or Inventory
Frequent	A	Likely to occur frequently	Continuously experienced
Probable	B	Will occur several times in the life of an item	Will occur frequently
Occasional	C	Likely to occur some time in the life of an item	Will occur several times
Remote	D	Unlikely but possible to occur in the life of an item	Unlikely but can reasonably be expected to occur
Improbable	E	So unlikely, it can be assumed occurrence may not be experienced	Unlikely to occur, but possible

Figure 13. Hazard Probability Levels (from Mil-Std 882C)

Task Number	Task Title
100	**Program Management and Control**
101	System Safety Program
102	System Safety Program Plan
103	Integration / Management of Associate Contractors, Subcontractors and Architect and Engineering Firms
104	System Safety Program Reviews and Audits
105	System Safety Group / System Safety Working Group Support
106	Hazard Tracking and Risk Resolution
107	System Safety Progress Summary`
200	**Design and Integration**
201	Preliminary Hazard List
202	Preliminary Hazard Analysis
203	Safety Requirements / Criteria Analysis
204	Subsystem Hazard Analysis
205	System Hazard Analysis
206	Operating and Support Hazard Analysis
207	Health Hazard Assessment
300	**Design Evaluation**
301	Safety Assessment
302	Test and Evaluation Safety
303	Safety Review of Engineering Change Proposals, Specification Change Notices, Software Problem Reports and Requests for Deviation / Waiver
400	**Compliance and Verification**
401	Safety Verification
402	Safety Compliance Assessment
403	Explosive Hazard Classification and Characteristics Data
404	Explosive Ordnance Disposal Data

Figure 14. Detailed Safety Tasks (from Mil-Std 882C)

	Hazard Category			
Frequency	Catastrophic	Critical	Marginal	Negligible
Frequent	High	High	High	Medium
Probable	High	High	Medium	Low
Occasional	High	High	Medium	Low
Remote	High	Medium	Low	Low
Improbable	Medium	Low	Low	Low

Figure 15. Example Matrix for Residual Risk (from Mil-Std 882C)

An additional assessment of risk is recommended for software, which considers the potential hazard severity and the degree of control that the software exercises over the application. Four control categories are defined, as follows.

"C1. Software exercises autonomous control over potentially hazardous hardware systems[5], subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazard's occurrence.

"C2a. Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.

"C2b. Software item displays information requiring immediate operator action to mitigate a hazard. Software failures will allow or fail to prevent the hazard's occurrence.

"C3a. Software item issues commands over potentially hazardous hardware systems, subsystems or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.

"C3b. Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.

"C4. Software does not control safety critical hardware systems, subsystems or components and does not provide safety critical information."

From this list and the list of hazard categories, a software hazard criticality matrix can be defined. This is shown in Figure 16. Risks range from 1 to 5, which may be interpreted as follows:

1 High risk—significant analysis and testing is required.

2 Medium risk—requirements and design analysis and in-depth testing is required.

3,4 Moderate risk—high level analysis and testing is acceptable with management approval.

5 Low risk—acceptable; no additional analysis is required.

This scheme does not easily fit reactor protection systems. It addresses a primary control system which controls potentially hazardous equipment. A reactor protection system is an "independent safety system" in the sense of item C1A.

A.1.3. AFISC SSH 1-1, Software System Safety Handbook

This is a companion document to Mil-Std 882, and is designed to specifically address software.[6] Software hazards fall into four broad categories:

1. Inadvertent/unauthorized event. An unexpected/unwanted event occurs.
2. Out-of-sequence event. A known and planned event occurs but not when desired.
3. Failure of event to occur. A planned event does not occur (e.g., a hazard is allowed to propagate because the program does not detect the occurrence of the hazard or fails to act).
4. Magnitude or direction of event is wrong. This is normally indicative of an algorithm error.

[5] In this list, "hardware" refers to all forms of equipment, not just computer hardware. For example, a missile is considered to be hardware here.

[6] Another handbook, Mil-Hdbk-764, briefly discusses software hazards analysis.

	Hazard Category			
Control Category	Catastrophic	Critical	Marginal	Negligible
C1	5	5	3	1
C2	5	4	2	1
C3	4	3	1	1
C4	3	2	1	1

Figure 16. Example Software Hazard Criticality Matrix (from Mil-Std 882C)

The software hazard analysis effort should begin early in the software life cycle. It is intended to ensure that the software complies with safety requirements and to identify potentially hazardous conditions. Software hazard analysis (SwHA) must be fully integrated with the overall system hazard analysis. Two phases are identified: preliminary software hazard analysis (PSwHA) and follow-on software hazard analysis (FSwHA). However, it is probably better to view these as a single analysis which starts with the PSwHA and is revised as needed during software development.

The PSwHA is based on an analysis of the following documents:

1. System and subsystem PHAs
2. System and subsystem specifications
3. System allocation and interface documents
4. Functional flow diagrams and related data
5. Flow charts or their functional equivalent
6. Storage allocation and program structure documents
7. Background information related to safety requirements associated with the contemplated testing, manufacturing, storage, repair and use
8. System energy and toxic or hazardous event sources which are controlled or influenced by software

The combination of the PHA and the allocation of system functions to software can be used to identify the software components which are critical to safety. These must be investigated in depth; other components must be analyzed to ensure that their operation or failure cannot impact or influence safety-critical components.

The software hazard analyses should be revised from time to time during the development process. The handbook recommends revision after the critical design review, during coding, and during integration. Special attention should be placed on changes to requirements, design, and coding.

The handbook lists several methods of software hazard analysis; the list is not meant to be exhaustive. Software fault tree analysis, software sneak circuit analysis, nuclear safety cross-check analysis and Petri net analysis are discussed.

A.1.4. IEC 1226, Classification of Safety Systems in Nuclear Power Plants

This Standard also uses a hazard severity classification scheme, but does not weight it by probability of occurrence. Three categories are used, labeled A, B and C. The Standard is specific to nuclear reactors, so is particularly appropriate to this report. See Figure 3.

The following notations are used:

FSE Functions and the associated Systems and Equipment that implement them

I&C Instrumentation and Control

NPP Nuclear Power Plant

PIE Postulated Initiating Event

A.1.4.1. Category A

Category A “denotes the FSE which play a principal role in the achievement or maintenance of NPP safety.” An I&C FSE falls into this category if it meets any of the following criteria:

- It is required to mitigate the consequence of a PIE to prevent it from leading to an accident.
- Its failure when required to operate in response to a PIE could result in an accident.
- A fault or failure in the FSE would not be mitigated by another category A FSE, and would lead directly to an accident.
- It is required to provide information or control capabilities that allow specified manual actions to be taken to mitigate the consequences of a PIE to prevent it from leading to an accident.

Typical functions of a category A FSE are the following:

- Reactor shutdown and maintenance of subcriticality
- Decay heat transport to an ultimate heat sink
- Isolation of containment
- Information for essential operator action

Examples of such FSE are the reactor protection system, the safety actuation system and safety system support features. Key instrumentation and displays that permit operator actions defined in the operations manual, and required to ensure safety are also examples of category A FSEs.

A.1.4.2. Category B

Category B "denotes FSE that play a complementary role to the category A FSE in the achievement or maintenance of NPP safety." An I&C FSE falls into this category if it meets any of the following criteria:

- It controls the plant so that process variables are maintained within safe limits.
- A requirement for operation of a category A FSE in order to avoid an accident would result from faults or failures of the category B FSE.
- It is used to prevent or mitigate a minor radioactive release, or minor degradation of fuel.
- It is provided to alert control room personnel to failures in category A FSE.
- It is provided to continuously monitor the availability of category A FSEs to accomplish their safety duties.
- It is used to reduce considerably the frequency of a PIE.

Typical functions of a category B FSE are:

- Automatic control of the reactor primary and secondary circuit conditions, keeping variables within safe limits, and prevention of events from escalating to accidents.
- Monitoring and controlling performance of individual systems and items of equipment during the post-accident phase to gain early warning of the onset of problems.
- Limiting the consequences of internal hazards.
- Monitoring or controlling the handling of fuel where a failure could cause a minor radioactive release.

Examples of category B FSE are the reactor automatic control system, control room data processing systems, fire suppression systems and safety circuits and interlocks of fuel handling systems used when the reactor is shut down.

A.1.4.3. Category C

Category C "denotes FSE that play an auxiliary or indirect role in the achievement or maintenance of NPP safety." An I&C FSE falls into this category if it meets any of the following criteria:

- It is used to reduce the expected frequency of a PIE.
- It is used to reduce the demands on, or to enhance the performance of, a category A FSE.
- It is used for the surveillance or recording of conditions of FSE, to determine their safety status.

- It is used to monitor and take mitigating action following internal hazards within the reactor design bases, such as fire and flood.
- It is used to ensure personnel safety during or following events that involve or result in release of radioactivity within the reactor, or risk radiation exposure.
- It is used to warn personnel of a significant release of radioactivity in the reactor or of a risk of radiation exposure.
- It is used to monitor and take mitigating action following natural events such as earthquakes and tornadoes.
- It is the NPP internal access control.

Typical functions of a category C FSE are:

- Those necessary to warn of internal or external hazards, such as fire, flood, explosions, earthquakes.
- Those for which operating mistakes could cause minor radioactive releases, or lead to radioactive hazard to the operators.
- Access control systems.

Examples include alarm systems, waste stream monitoring, area radiation monitoring, access control systems, and emergency communications systems.

A.1.4.4. Effect of the Classification Scheme

The primary effect is to concentrate development and assurance efforts on the most important FSEs—those of category A. An example is the use of redundancy to achieve reliability. A category A FSE is required to have redundancy so that no single point of failure exists. Redundancy is encouraged for category B FSEs, but is not required if the reliability goals can be met without it. No redundancy is generally needed for category C FSEs, though it can be used if necessary to meet reliability goals.

A.1.5. IEEE 7-4.3.2, Annex F, Abnormal Conditions and Events

Annex F of IEEE 7-4.3.2 discusses the identification and resolution of abnormal conditions and events (ACEs).

ACEs are grouped into two categories, depending on their source. Some are caused by conditions or events that occur outside the computer system—a failure of a system component is an example. Others are caused by failures within the computer system.

Section F.2.3 of the Annex describes a process for identifying ACEs based on the software life cycle. It begins at the system design phase, and proceeds through computer system design, software requirements, software design, software implementation, computer integration testing and computer validation testing. The Standard lists various considerations for most of the life cycle phases; these are summarized in Figures 17-20.

A general procedure is provided for resolving ACEs. The intent is to eliminate ACEs or reduce the associated risk where possible. A summary of the procedure is given in Figure 21.

A.2. General Discussion of Hazard Analysis

Hammer (1972) lists six functions of hazard analysis:

1. The investigation and evaluation of the interrelationships of primary, initiating and contributory hazards that may be present.
2. The investigation and evaluation of the circumstances, conditions, equipment, personnel and other factors involved in the safety of the system and its operation.
3. The investigation and evaluation of the means of avoiding or eliminating any specific hazard by using suitable designs, procedures, processes or materials.

4. The investigation and evaluation of the controls that may be required to prevent possible hazards and the best methods for incorporating those controls in the product or system.
5. The investigation and evaluation of the possible damaging effects resulting from lack or loss of control of any hazard that cannot be avoided or eliminated.
6. The investigation and evaluation of the safeguards for preventing injury or damage if control of the hazard is lost.

Initial hazards analyses must be carried out for the entire application system. This report assumes that the five forms of system-level hazards analyses identified in Mil-Std 882C have been carried out, except for software components. The following is a brief list of the types of hazard analysis given in the Standard:

1. Preliminary Hazard List (PHL) identifies hazards that may require safety design consideration or special analyses. It occurs upon completion of the concept definition phase of the system life cycle.
2. Preliminary Hazard Analysis (PHA) identifies and evaluates all system hazards. It starts in the concept definition phase of the system life cycle, and ends when the component-level System Hazard Analysis is able to begin. The PHA is the foundation for future system and software hazard analyses.
3. System Hazard Analysis (SHA) examines the entire system to identify hazards and assess the risk of the entire system design, including software. It starts as the system design matures, close to the design review, and is updated until the system design is complete.
4. Component SHA identifies hazards associated with the design of the components, and how those hazards will affect the entire system. It begins as each component is designed and is updated as the component design matures.
5. Operating and Support Hazard Analysis (O&SHA) identifies and evaluates hazards related to the environment, personnel, procedures and equipment during a system operation performed by humans. It begins before the system test and integration life cycle phase. O&SHA identifies safety requirements necessary to eliminate hazards or mitigate the risk of hazards.

These hazard analyses will identify certain hazards. The table in Figure 22 suggests broad classes of hazards that may be present. The various system hazard analyses will attempt to eliminate as many hazards as possible, reduce the probability of occurrence of those that remain, and reduce the potential damage which may result from accidents. In the latter two cases, responsibility will be assigned to specific system components for the control of occurrences and consequences. In some cases, software components may be assigned such responsibility. If this occurs, software hazard analysis is a form of component hazard analysis.

a.	Occurrence of design bases conditions identified in the Safety Analysis Report.
b.	Possible independent, dependent and simultaneous ACE events considering failures of safety equipment.
c.	Interface considerations among various elements of the system.
d.	Environmental constraints.
e.	Operating, test, maintenance and emergency procedures.
f.	Design and use of test and maintenance equipment that has potential for introducing damage, software errors or interrupts.
g.	Safety equipment design and possible alternate approaches.
h.	Degradation in a subsystem or the total system from normal operation of another subsystem including non-safety systems.
i.	Modes of failure, including reasonable human errors as well as single point failures, and ACEs created when failures occur in subsystem components.
j.	Potential contribution of software, events, faults and occurrences on safety of the system.
k.	Potential common mode failures.
l.	The method of implementation of the software design requirements and corrective actions will not impair or degrade the safety system nor introduce new ACEs.
m.	The method of controlling design changes during and after system acceptance will not degrade the safety system nor introduce new ACEs.

Figure 17. Summary of Safety System ACEs Identification

a.	Software requirements should be evaluated to identify those that are essential to accomplishing the safety function. These critical requirements should be evaluated against the ACE to assess their significance.
b.	Requirements for timing and sizing should be included to ensure adequate resources for execution time, clock time and memory allocations are provided to support the critical requirements.
c,	In designs involving the integration of multiple software systems, consideration should be given for interdependencies and interactions between the components of the system.
d.	Existing software should be evaluated to ensure adequate confidence that no “unintended functions” detrimental to the operation of the safety system are introduced.

Figure 18. Summary of Software Requirements ACEs Identification

a.	Equations, algorithms and control logic should be evaluated for potential problems.
b.	Potential computational problems should be evaluated.
c.	Evaluation of data structure and intended use should be performed.
d.	Potential data handling problems should be evaluated.
e.	Interface design considerations should be reviewed.
g.	Adequate confidence that the design fits within the identified system constraints.
h.	Software modules that implement critical functions should be identified.
i.	Non-safety modules should be evaluated to provide adequate confidence that they do not adversely affect safety software.

Figure 19. Summary of Software Design ACEs Identification

a.	Evaluate equations, algorithms and control logic for potential problems.
b.	Confirm the correctness of algorithms, accuracy, precision and equation discontinuities, out of range conditions, breakpoints, erroneous inputs, etc.
c.	Evaluate the data structure and usage in the code to provide adequate confidence that the data items are defined and used properly.
d.	Provide adequate interface compatibility of software modules with each other and with external hardware and software.
e.	Provide adequate confidence that the software operates within the imposed constraints.
f.	Examine non-critical code to provide adequate confidence that it does not adversely affect the function of critical software.
g.	Provide adequate confidence that the results of coding activities are within the timing and sizing constraints.

Figure 20. Summary of Software Code ACEs Identification

a.	Eliminate identified ACEs or reduce associated risk through design, if possible.
b.	Ensure that the safety functions are protected from identified ACEs, and that non-safety functions do not create ACEs for the safety functions.
c.	Identify, evaluate and eliminate ACEs associated with each system throughout the entire life cycle of a system.
d.	Minimize risk resulting from excessive environmental conditions.
e.	Design to minimize risk created by human error in the operation and support of the system.
f.	Create unambiguous requirements definitions to minimize the probability of misinterpretation by developers.
g.	Consider and use historical ACEs data, including lessons learned from other systems.
h.	Minimize risk by using existing designs and test techniques whenever possible.
i.	Analyze for ACEs and document changes in design, configuration or system requirements.
j.	Document identified ACEs and their resolution.

Figure 21. Summary of General Guidelines for ACE Resolution

Acceleration and motion	Leakage
Chemical reactions	Moisture
Dissociation	High humidity
Oxidation	Low humidity
Replacement	Power source failure
Contamination	Pressure
Corrosion	High pressure
Electrical	Low pressure
System failure	Changes
Inadvertent activation	Radiation
Shock	Thermal
Thermal	Electromagnetic
Explosion	Ionizing
Fire	Ultraviolet
Heat and temperature	Structural damage or failure
High temperature	Stress concentrations
Low temperature	Stress reversals
Changes	Toxicity
Impact and shock	Vibration and noise

Figure 22. Classes of Hazards (Hammer 1972)

A.3. NIST Review of Software Hazard Analyses

This draft report, based primarily on Mil-Std 882B (the predecessor of 882C), lists three requirements for software hazard analysis. The Software Hazard Analysis should:

1. Respond to every hazard identified in the System Hazard Analysis.
2. Ensure that the operation of the software does not interfere with the safety goals or operation of the system.
3. Evaluate and mitigate how software could hinder the safety goals or operation of the system.

The report describes six different software hazard analyses. The following description is taken from the report.

1. Software Requirements Hazard Analysis (SwRHA) ensures that system safety requirements have been properly defined, and that they can be traced from the system requirements to the software requirements; software design; and operator, user and diagnostic manuals. It begins during the requirements phase of the system life cycle. The PHL and PHA are inputs to this analysis. SwRHA examines the system requirements, software requirements and software design by reviewing system and software requirements documentation and program documentation. Recommendations and design and test requirements are incorporated into the Software Design

Documents and the Software Test Plan. The results of the SwRHA are presented at the System Requirements Review (draft), System Design Review (update) and Software Requirements Review (final).

2. Software Design Hazard Analysis (SwDHA) identifies safety-critical software components that require analysis beyond design. It starts after the Software Requirements Review and should be mostly completed before starting software coding. The PHA, SHA and SwRHA are inputs to this analysis. SwDHA defines and analyzes safety critical software components (e.g., assessing their degree of risk and relationships to other components) and the design and test plan (e.g., ensuring safety requirements are properly defined in the design). Changes are made to the Software Design Document (to eliminate hazards or mitigate the risk of hazards), and safety requirements are integrated into the Software Test Plan. Recommendations are made for coding. The results of the SwDHA are presented at the Software Design Review.

3. Software Code Hazard Analysis (SwCHA) identifies how to eliminate hazards or mitigate the risk of hazards during the coding phase of the life cycle. It starts at the beginning of that phase and continues until after system testing has been completed. The SwDHA is the input to this analysis. SwCHA analyzes the actual source and object code, system interfaces, and software documentation (to ensure safety requirements are included). Recommendations are made to change the software design, code and software testing. The results of the SwCHA are presented at the Test Readiness Review. (SwCHA results for lower level units are given to programmers during coding.)

4. The purpose of Software Safety Testing is to determine that all hazards have been eliminated or that each hazard's risk has been mitigated. Software Safety Testing of lower-level units starts very soon after their coding is completed. Software Safety Testing tests safety-critical software components under normal conditions and under abnormal environment and input conditions. It also ensures that the software performs correctly and safely under system testing. Software Safety Testing includes testing of any commercial or government furnished software present in the system. The results of Software Safety Testing is to identify corrections to the software which, when implemented, will eliminate hazards or mitigate the risk of hazards. Retests are then performed on the corrected software under the same conditions. Testing of the software at the system level starts following a successful Test Readiness Review.

5. The Software/User Interface Analysis manages hazards that were not eliminated or controlled in the system design phase. For example, change recommendations are made to the design that provide hazard detection and operator warning, hazard recovery, and event or process termination.

6. Software Change Hazard Analysis analyzes all changes made to the software to determine their impact on safety. Software hazard analysis and testing is performed on all changes made to the requirements, design, code, systems, equipment, and test documentation to ensure that the changes do not create new hazards or affect existing hazards and that the change is properly incorporated into the code.

A.4. Review of the Published Literature

The published literature on software hazard analysis is sparse and recent, except for the application of fault trees to software. Some general background can be found in (Brown 1985; Leveson 1991b; Gowen 1992; and Elahi 1993).

Leveson (1991a) proposes to augment traditional software engineering by a form of hazard analysis; this idea forms the basis for the approach proposed in this report.

The use of fault trees to analyze software has received considerable attention. The following may be consulted: Leveson 1983; Fryer 1985; Connolly 1989; Lal-Gabe 1990; Bowman 1991; Leveson 1991a; Leveson 1991c; McKinlay 1991; Levan 1992; Clarke 1993; and Levinson 1993. Much of this, however, uses fault trees to analyze code for correctness. There are two difficulties with this approach, and it is not recommended in this report. First, the most important decisions that may affect hazards occur early in the life cycle, when the requirements are specified and the basic computer system architecture is chosen. A fault tree constructed after this is done is likely to overlook the early decisions, resulting in missed opportunities for improvement. Second, a fault tree carried to the program language level is likely to be very large, making it hard to analyze. There is also the temptation to concentrate on the leaves (statements), missing the important intermediate nodes of the tree that capture combinations of events that can lead to failures.

The use of fault trees early in the software development process can be quite useful, particularly if they are an extension of fault trees developed for the overall reactor. They should probably be restricted to analysis, since the assignment of failure probabilities to software architectural elements is very problematic.

A few articles discuss other techniques. Leveson (1991a) also includes a discussion of Petri nets and state charts, and Mojdehbakhsh (1994) includes Statemate charts within a discussion of fault trees. Levinson (1993) includes fault trees, failure modes and effects analysis (FMEA) and Parts Stress Analysis (PSA).

Two articles were most influential on the development of this report, both by McDermid (1994, 1995). The first proposes the use of HAZOP and guide words to structure software hazard analysis, while the latter describes experiences in carrying out the technique. This report extends the approach by McDermid, placing it into a broader plan for software hazard analysis, extending the list of guide words to cover many aspects of software, and specializing somewhat to the nuclear reactor industry.

APPENDIX B. 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 C. SOFTWARE TOOLS FOR HAZARD ANALYSIS

Hazard analysis in general, and software hazard analysis in particular, can be much assisted by the use of convenient software tools. Many tools are available that address different aspects of hazard analysis and run on different platforms. Costs vary from free to tens of thousands of dollars. Capabilities and quality also vary considerably. Platforms include PC, Macintosh, Sun and other Unix, VAX and other systems.

A small sample of tools was examined as an indication of the types of tools available. The sampling was restricted to tools that use the PC as a platform (running either under MS-DOS or Windows 3.x), and tools that cost less than $500.00. Only one example from each type of analysis was examined. Results are indicative, but are not necessarily representative of the tools available in the marketplace. Tool revisions are frequent, so the vendors should be contacted directly for current release information. No endorsement of any tool is implied by this study.

Six subject areas were used in the study:

- Fault tree analysis (FTA)
- Failure modes, effects and criticality analysis (FMEA and FMECA)
- HAZOP
- Hazard tracking
- Markov chain modeling
- Reliability growth analysis

The remainder of this appendix describes the various programs that were investigated. Each section begins with a brief description of the program: program name, vendor, platform and primary functions. A description of the program's capabilities follows, with illustrations of the reports that may be produced. No attempt is made here to discuss the various techniques (see Lawrence (1993) and the references given there for background). Opinions expressed are those of the author, and apply only to the software versions actually examined. Most of the versions examined had minor faults; these are not discussed. Some major problems are given for a few of the programs when these appeared to create considerable difficulty in using the program.

C.1. Fault Tree Analysis

Product Name:	FaulTrEase
Product Vendor:	Arthur D. Little, Inc., Cambridge, MA.
Platform:	Windows 3.1. (A Macintosh version is available)

FaulTrEase is a program for drawing and evaluating fault trees. The program provides considerable help in drawing the trees, since it calculates the proper position of the entire tree as nodes and branches are added or deleted. As a result, a fault tree can be constructed with considerable ease by concentrating on the contents of nodes and the connections between them. Building fault trees using this program is quite efficient.

Calculations include cut sets, probability calculations and calculation of importance.

The program is not able to handle n-out-of-m nodes, which hampers use for analysis of reactor protection systems, where it is desirable to include 2-out-of-4 and similar logics. Printing is limited to a single sheet, even though the program knows that several sheets are required in order to print the full tree. This makes large fault trees difficult to document. The solution is to divide the tree into separate subtrees, perform calculations on the latter, and manually insert the results into the main tree. This is subject to copying errors, and is quite inconvenient.

The figures show an example using fault trees for the AP600 reactor design. The probability data shown is that used in the AP600 fault trees when available; estimates are used when AP600 data was not given.

The approach used was to copy the fault trees from material provided by the NRC Program Monitor. Options in the program permit tree layouts to be compressed to save space, or

expanded for better appearance. Both options are illustrated in the examples. Probability values are assigned to leaves of the tree and the probability associated with the top node of the tree can be calculated by the program. If subtrees are used, they are evaluated first, and then the value associated with the top node of the subtree is manually entered into the corresponding off-page connector of the main tree. For example, the first page shows the top-level tree, as printed by the program. Off-page connecting nodes labeled 8, 11, 22 and 24 are all similar, and use the results of the second tree, "AP600 Failure to Actuate Valve" tree. This tree requires two printed pages since it's too large to fit on a single sheet; off-page connector "B" is used to connect the sheets.

Some additional lower-level charts are illustrated on succeeding pages.

C.2. FMEA, FMECA, HAZOP

Product Name: HAZOPtimizer

Product Vendor: Arthur D. Little, Inc., Cambridge, MA.

Platform: DOS 5.x or Windows 3.1.

HAZOPtimizer is used to organize, store and print tabular reports used for various types of hazard analysis. Report columns can be named, defined and arranged to suit the needs of the user. The product gives the appearance of a semi-structured interface to a database.

The primary unit of data collection is termed a *study*—which documents the results of a particular analysis on a particular problem. Typical analyses include FMEA, FMECA, HAZOP and PHA. Some pre-defined templates exist, but were not found useful.

Study results are organized into *sheets*. Each sheet has the same definition of data columns; the use of multiple sheets is for convenience. A sheet contains *lines*, upon which the study results are entered.

The figures show two sample studies. The first is an FMECA study from the book *Guidelines for Hazard Evaluation Procedures*, Center for Chemical Process Safety, 1992, page 207. The second uses data from a 1992 evaluation of the GE ABWR design performed by LLNL for NRC/NRR. Only a small portion of that study is included.

The examples illustrate the flexibility of the program, since different column definitions were used by the two different sources.

The program limits each box in the tables to 256 characters; this appears to be a limitation inherited from the underlying database management system, and was found to be extremely inconvenient.

C.3. Hazard Tracking

Product Name: HazTrac

Product Vendor: Hoes Engineering, Davis, CA.

Platform: DOS 5.x or Windows 3.1.

HazTrac assists in carrying out a hazard analysis. It is considerably more structured than HAZOPtimizer, and is organized to meet the requirements of Mil Std 882B. HazTrac can be used to support the analyses specified therein: PHA, SHA, SSHA, FMEA and OSHA. (There is also an option specific to the State of California, which is not discussed here.)

Information is organized into three levels: hazard, recommendations and status. There is an entry at the first level for each defined hazard. This depends on the type of analysis; a PHA, for example, records the scenario under which the hazard may occur, the potential effects and an assessment of risk. The latter use the Mil Std tables shown earlier in Figures 12 and 13.

The second level permits the recording of recommendations on eliminating or mitigating the hazard. The program records the recommendations and some associated information, including names of people and organizations responsible for the recommendation and due dates.

The third level permits tracking of changes in the status of each recommendation. This includes the current status (open, agreed,

dropped or verified) and a historical log of status events.

This program is considerably easier to use than HAZOPtimizer, but restricts the user to the built-in forms. That is, ease of use comes at the expense of flexibility. Text fields are also limited to 256 characters, which remains troublesome. A particularly irritating feature is the restriction that the program and data must reside on different disks. The computer used for examining the program has only a single hard disk, so the HazTrac data was placed on a floppy disk. No logical reason for this requirement was known.

An example of a PHA is shown in the later figures. It shows a hypothetical analysis of a chlorine plant, taken from the book *Guidelines for Hazard Evaluation Procedures*, Center for Chemical Process Safety, 1992, pages 270-271.

C.4. Markov Chain Modeling

Product Name: CARMS

Product Vendor: Daina

Platform: Windows 3.1

CARMS is a general Markov modeling program. It can be used to draw a Markov model, assign probabilities to transitions and run a simulation. The latter provides a graph of the calculated probabilities of the various states through time, which provides the user with knowledge of how the state probabilities change with time, and how fast they move to a steady state.

A model is constructed by defining states, transitions between states, initial probabilities for the states, and transition probabilities. The latter can be defined using equations. Drawing the model is quite easy using the built-in capabilities.

CARMS can show the model as a drawing or as a table. Several examples are shown, giving the diagram and the results of the simulation. The screen display of the simulation shows labels for the various lines in the graph; they are not printed, however. To show this, the lines in the graph were annotated by hand below.

There's not much more to write about this program. It does one thing, and does it very nicely.

C.5. Reliability Growth Modeling

Product Name: CASRE.

Product Vendor: Jet Propulsion Laboratory, Pasadena, CA.

Platform: Windows 3.1.

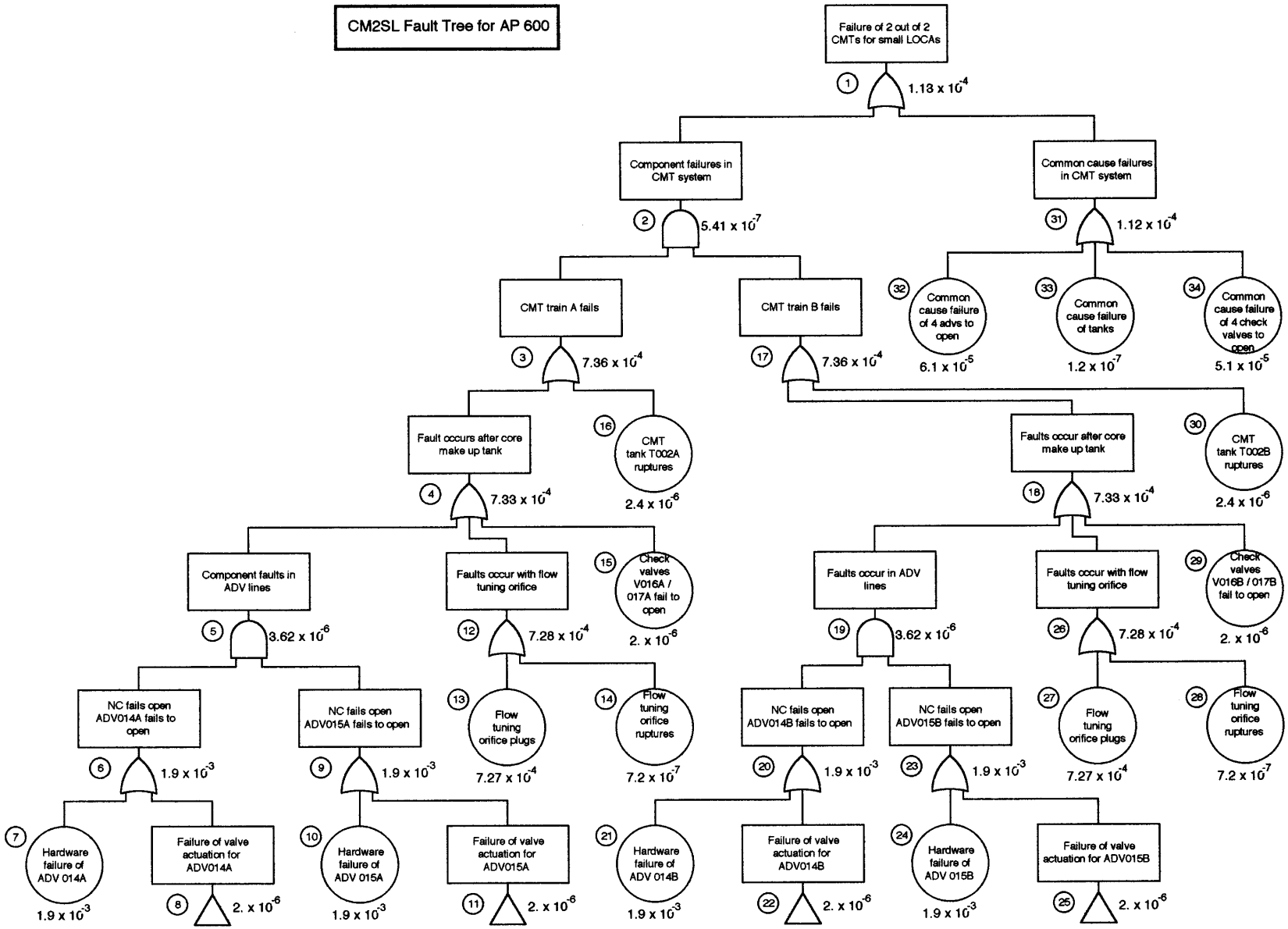
Reliability growth modeling is a technique for predicting the reliability of a product undergoing test when the product is re-engineered after each failure. This pattern is well suited to software testing when each failure causes the underlying fault to be repaired and testing to be resumed with the modified program.

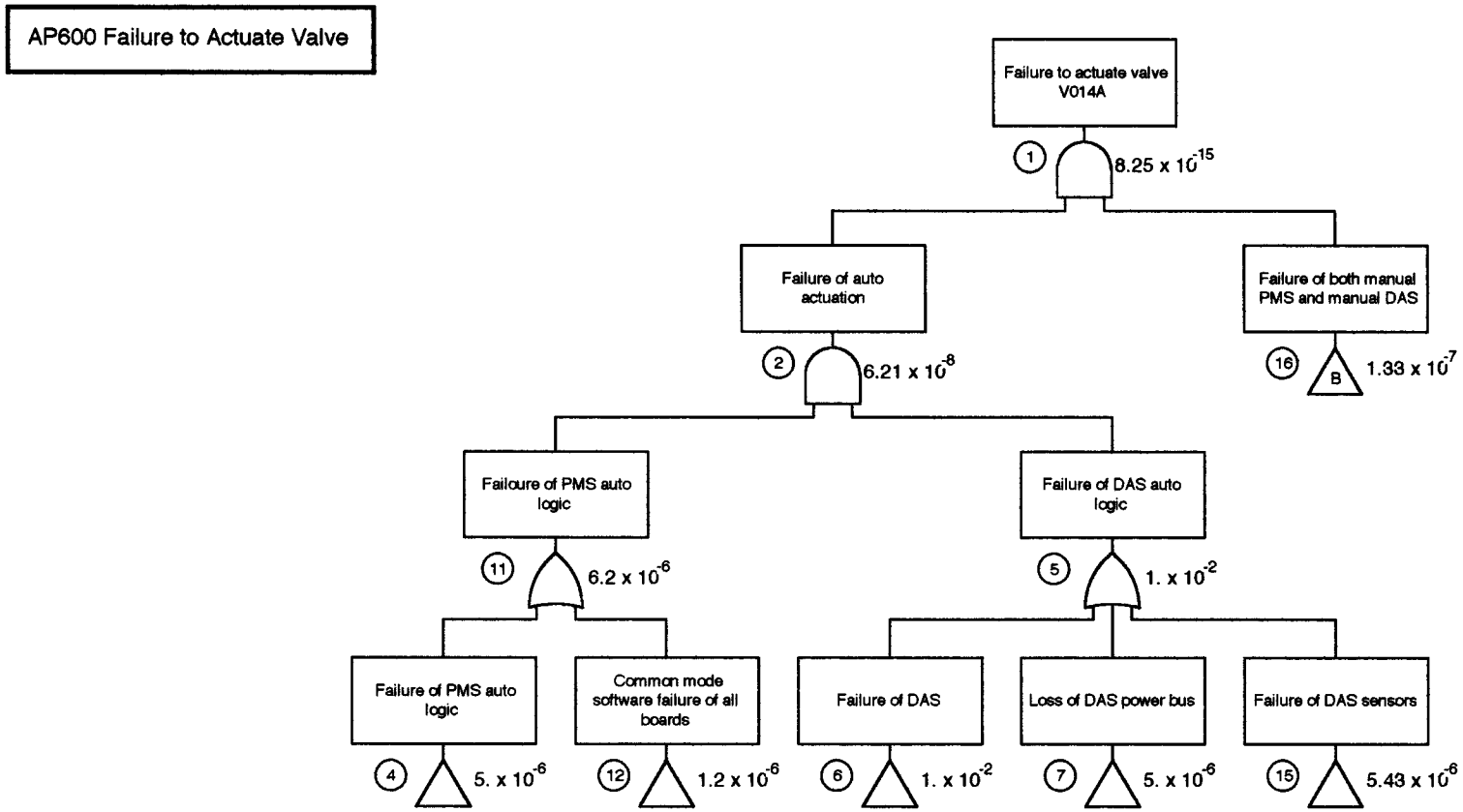
The primary product for modeling software reliability growth is SMERFS, a public domain package available from the Naval Surface Warfare Center. CASRE uses SMERFS, but has a window interface and several additional features.

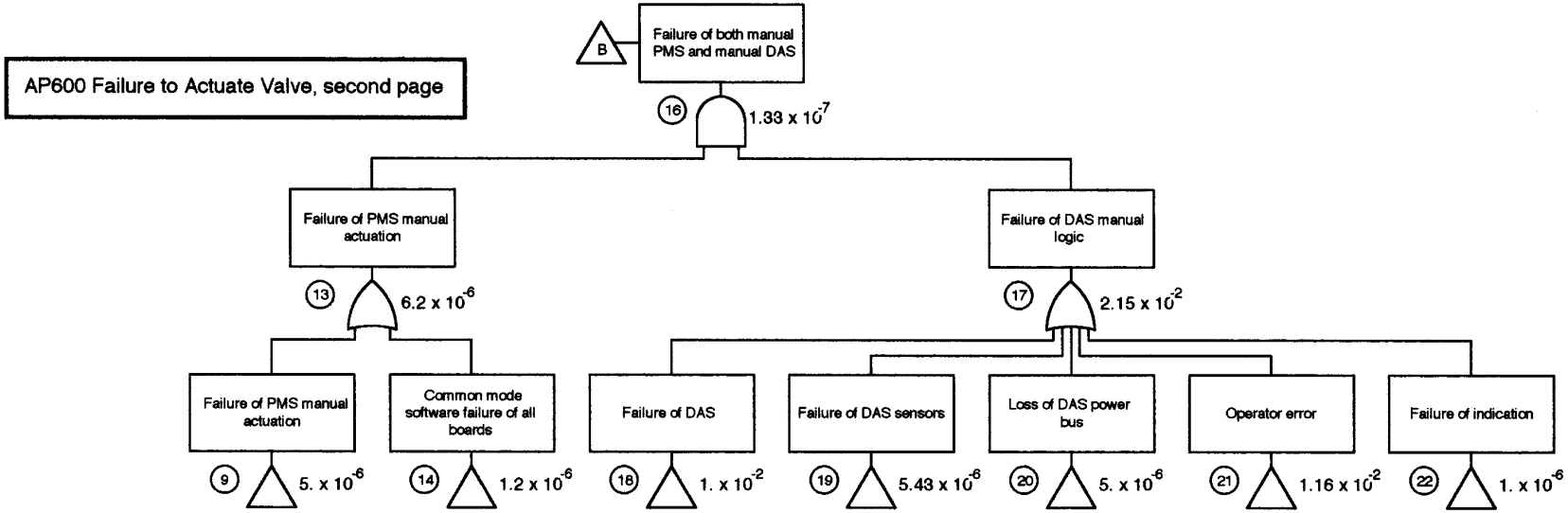
CASRE can be used starting at any point during the testing of a module, program or software system. Failure data is recorded as testing and repair takes place. Two formats are possible: recording time units between each failure or recording the number of failures during each time interval.

The program analyzes the failure data in various ways and plots the results on the PC screen or a printer. Different models can be used to fit the data and make predictions. Several methods of displaying the results can be used; all are illustrated in the example below. This example uses a sample data set supplied with the product. Curve fitting is done using a Generalized Poisson model, with a prediction of future reliability shown as a line on the charts.

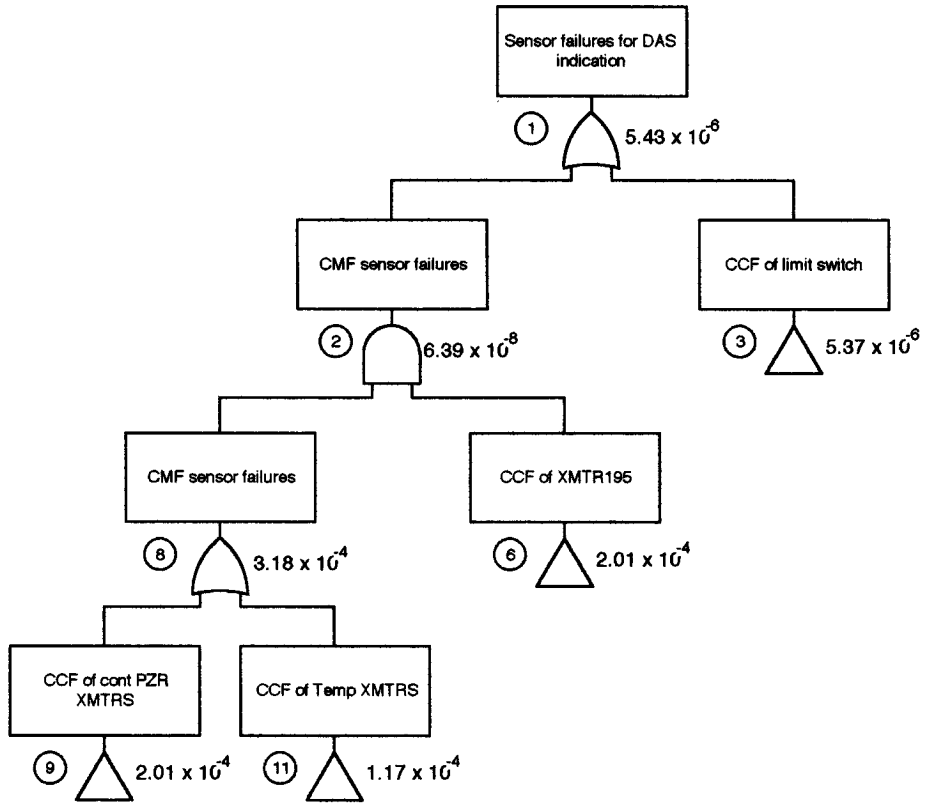
The final plot shows the same data fitted to two other models: the Schneidewind model and a Yamada S-shaped model.

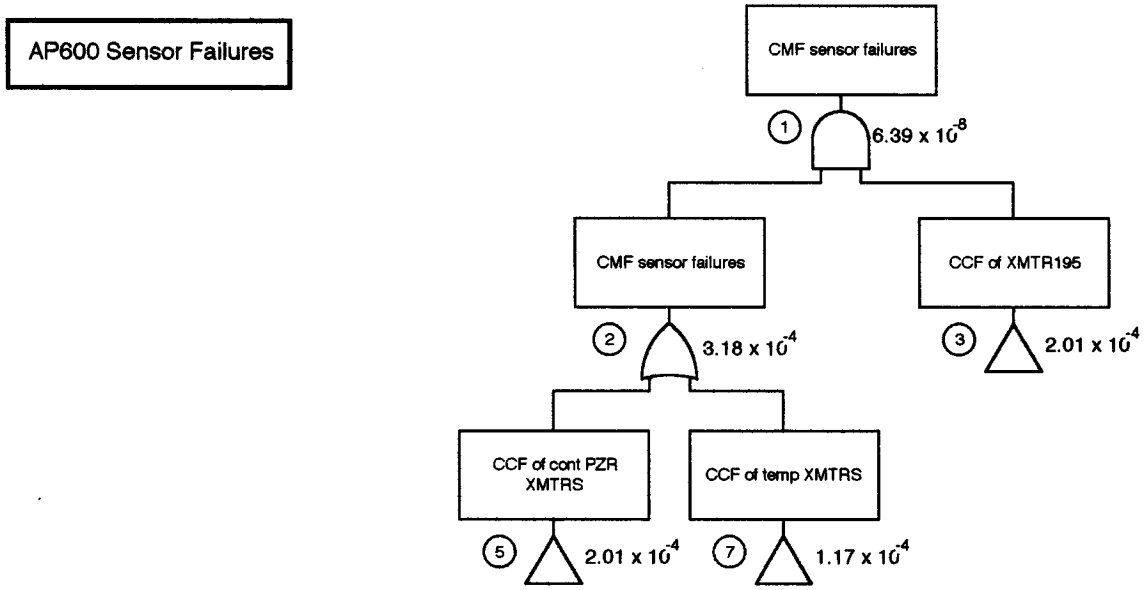

CM2SL Fault Tree for AP 600
Failure of 2 out of 2 CMTs for small LOCAs
1
1.13 x 10^-4
Component failures in CMT system
2
5.41 x 10^-7
Common cause failures in CMT system
31
1.12 x 10^-4
CMT train A fails
3
7.36 x 10^-4
CMT train B fails
17
7.36 x 10^-4
32
Common cause failure of 4 advs to open
6.1 x 10^-5
33
Common cause failure of tanks
1.2 x 10^-7
34
Common cause failure of 4 check valves to open
5.1 x 10^-5
Fault occurs after core make up tank
4
7.33 x 10^-4
16
CMT tank T002A ruptures
2.4 x 10^-6
Faults occur after core make up tank
18
7.33 x 10^-4
30
CMT tank T002B ruptures
2.4 x 10^-6
Component faults in ADV lines
5
3.62 x 10^-6
Faults occur with flow tuning orifice
12
7.28 x 10^-4
15
Check valves V016A / 017A fail to open
2. x 10^-6
Faults occur in ADV lines
19
3.62 x 10^-6
Faults occur with flow tuning orifice
26
7.28 x 10^-4
29
Check valves V016B / 017B fail to open
2. x 10^-6
NC fails open ADV014A fails to open
6
1.9 x 10^-3
NC fails open ADV015A fails to open
9
1.9 x 10^-3
13
Flow tuning orifice plugs
7.27 x 10^-4
14
Flow tuning orifice ruptures
7.2 x 10^-7
NC fails open ADV014B fails to open
20
1.9 x 10^-3
NC fails open ADV015B fails to open
23
1.9 x 10^-3
27
Flow tuning orifice plugs
7.27 x 10^-4
28
Flow tuning orifice ruptures
7.2 x 10^-7
7
Hardware failure of ADV 014A
1.9 x 10^-3
Failure of valve actuation for ADV014A
8
2. x 10^-6
10
Hardware failure of ADV 015A
1.9 x 10^-3
Failure of valve actuation for ADV015A
11
2. x 10^-6
21
Hardware failure of ADV 014B
1.9 x 10^-3
Failure of valve actuation for ADV014B
22
2. x 10^-6
24
Hardware failure of ADV 015B
1.9 x 10^-3
Failure of valve actuation for ADV015B
25
2. x 10^-6


AP600 Failure to Actuate Valve
Failure to actuate valve V014A
1
8.25 x 10^-15
Failure of auto actuation
Failure of both manual PMS and manual DAS
2
6.21 x 10^-8
16
B
1.33 x 10^-7
Failoure of PMS auto logic
Failure of DAS auto logic
11
6.2 x 10^-6
5
1. x 10^-2
Failure of PMS auto logic
Common mode software failure of all boards
Failure of DAS
Loss of DAS power bus
Failure of DAS sensors
4
5. x 10^-6
12
1.2 x 10^-6
6
1. x 10^-2
7
5. x 10^-6
15
5.43 x 10^-6

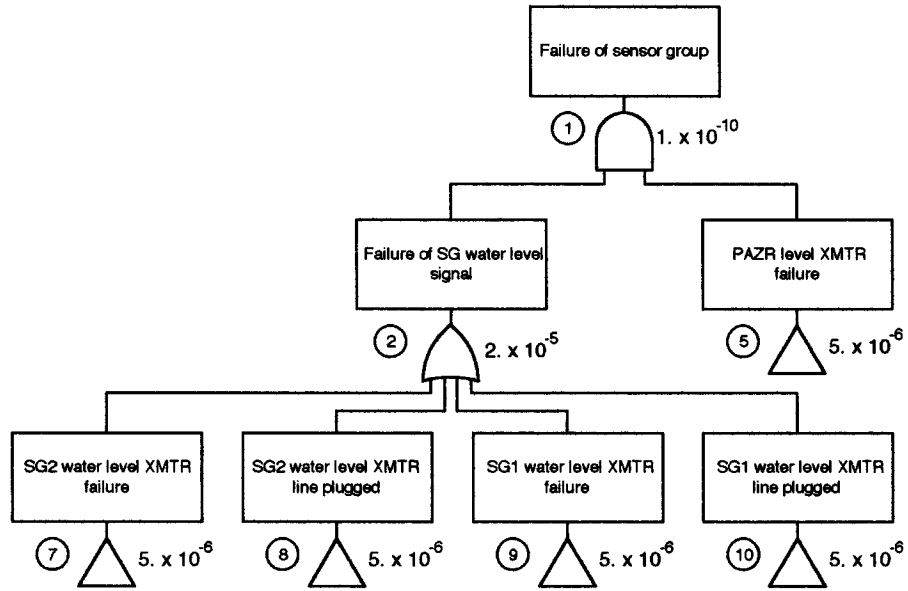

AP600 Failure to Actuate Valve, second page
B
Failure of both manual PMS and manual DAS
16
1.33 x 10^-7
Failure of PMS manual actuation
13
6.2 x 10^-6
Failure of DAS manual logic
17
2.15 x 10^-2
Failure of PMS manual actuation
9
5. x 10^-6
Common mode software failure of all boards
14
1.2 x 10^-6
Failure of DAS
18
1. x 10^-2
Failure of DAS sensors
19
5.43 x 10^-6
Loss of DAS power bus
20
5. x 10^-6
Operator error
21
1.16 x 10^-2
Failure of indication
22
1. x 10^-6

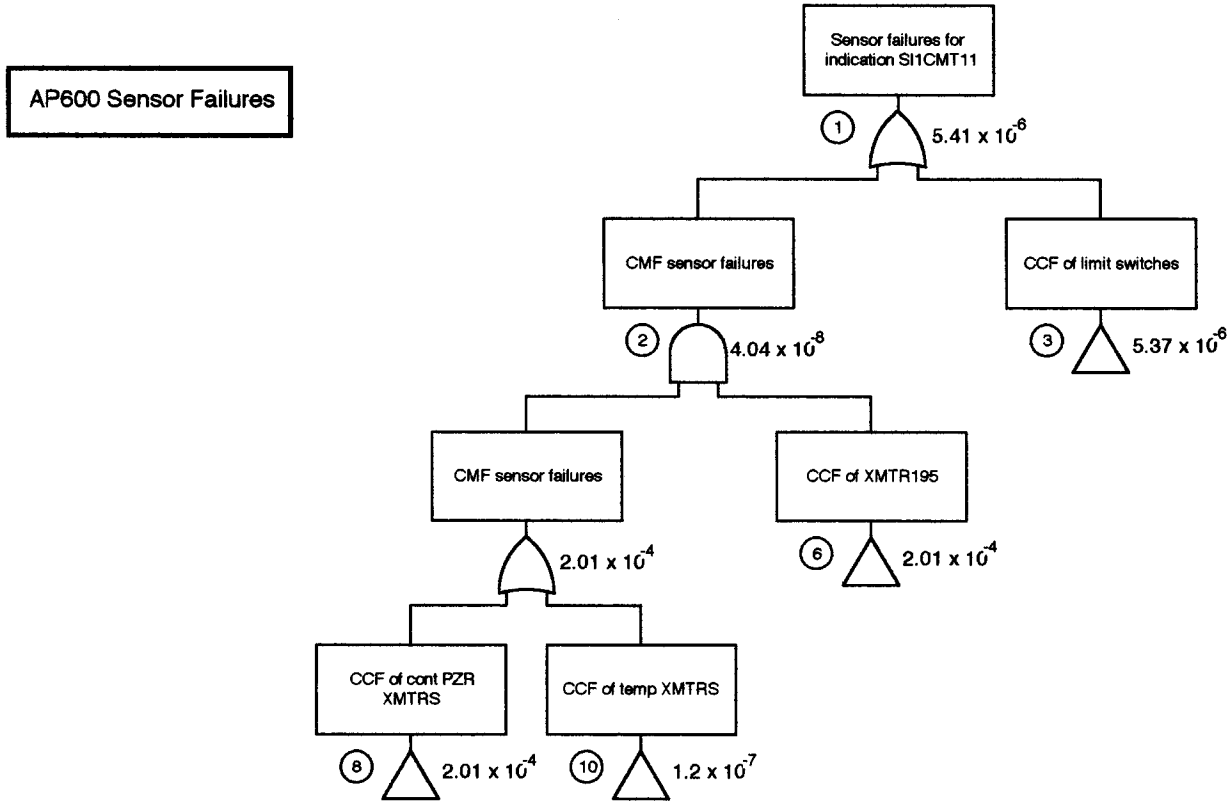
AP600 Sensor Failures for DAS Indication




AP600 Sensor Failures
CMF sensor failures
1
6.39 x 10^-8
CMF sensor failures
CCF of XMTR195
2
3.18 x 10^-4
3
2.01 x 10^-4
CCF of cont PZR XMTRS
CCF of temp XMTRS
5
2.01 x 10^-4
7
1.17 x 10^-4

AP600 Failure of Sensor Group




AP600 Sensor Failures
Sensor failures for indication SI1CMT11
1
5.41 x 10^-6
CMF sensor failures
CCF of limit switches
2
4.04 x 10^-8
3
5.37 x 10^-6
CMF sensor failures
CCF of XMTR195
2.01 x 10^-4
6
2.01 x 10^-4
CCF of cont PZR XMTRS
CCF of temp XMTRS
8
2.01 x 10^-4
10
1.2 x 10^-7

HAZOPtimizer by Arthur D. Little, Inc.

Page 1

Company: Guidelines for Hazard Evaluation ProceduresSheet Name: ccps_1
Facility: Center for Chemical Process Safety, 1992Reference:
HAZOP Date: 12/08/1994 Unit:
Leader/Secretary: Drawing Number:
Process: Print Date/Time: 9/8/1995 09:36:05
Team Members: Description: Extracted from page 207 of referenced book

Item number	Component Description	Failure Mode	Effect	Employee Safety Category	Production Cost Category	Equipment Cost Category	Frequency Category
1	Feed gas line	Leak	Release of feed gas. Fire likely. Plant 8 shut down to isolate rupture.	2	2	1	3
2	Feed gas line	Rupture	Release of feed gas. Explosion and/or fire likely. Plant 8 shut down to isolate rupture. Major damag	3	4	3	1
3	Feed gas knockout drum	Leak	Release of feed gas. Fire likely. Some damage to Plant 7 pipe rack.	1	1	1	3
4	Feed gas knockout drum	Rupture	Release of feed gas. Major explosion and/ or fire likely. Damage to pipe rack and nearby equipment.	3	4	2	1
5	Feed gas knockout drum	Level controller fails high	Potential carry-over of amine to downstream equipment. Minor upset.	1	1	1	4

HAZOPtimizer by Arthur D. Little, Inc.

Page 2

Company: Guidelines for Hazard Evaluation ProceduresSheet Name: ccps_1
Facility: Center for Chemical Process Safety, 1992Reference:
HAZOP Date: 12/08/1994 Unit:
Leader/Secretary: Drawing Number:
Process: Print Date/Time: 9/8/1995 09:36:06
Team Members: Description: Extracted from page 207 of referenced book

Item number	Component Description	Failure Mode	Effect	Employee Safety Category	Production Cost Category	Equipment Cost Category	Frequency Category
6	Feed gas knockout drum	Level controller fails low	Blowdown of gas to vessels in liquid service. Damage to vessel not designed for gas flow.	3	4	3	4
7	Outline	Leak	Release of feed gas. Fire likely. Some damage to Plant 7 pipe rack.	2	1	1	3

HAZOPtimizer by Arthur D. Little, Inc.

Company: LLNL
Facility: GE ABWR Reactor Design
HAZOP Date: 12/09/1994
Leader/Secretary:
Process:
Team Members:

Sheet Name: abwr
Reference: LLNL Report to NRC, Sept. 7, 1992
Unit:
Drawing Number:
Print Date/Time: 9/8/1995 09:40:42
Description:

Item number	Component	Function	Failure Mode	Impact on System	Detection	Remarks
1	1LDS	Uses various process variables to determine if a loss of coolant is occurring, and if so, it then isolates the leak from RPV.	The LDS consists of independent input devices, software and output devices contained in other safety systems. The input devices are all quadruply redundant. A single failure of the input devices has no effect on the successful operation of LDS.	No single failure can completely disable LDS.	By surveillance.	
2	NMS	Takes in LPRM neutron sensor values, averages the values to get a divisional average, sends the divisional average to other divisions, takes in other divisional averages, and averages divisional averages to produce an APRM trip signal.	The resultant average of the divisional average is incorrect and lower than the actual reutron flux.	The resultant average of the divisional averages in the NMS for all divisions is calculated to be significantly less than true value; reactor will not trip on desired APRM signal.	By surveillance.	It is assumed that range checking will be used.

HAZOPtimizer by Arthur D. Little, Inc.

Company: LLNL
Facility: GE ABWR Reactor Design
HAZOP Date: 12/09/1994
Leader/Secretary:
Process:
Team Members:

Sheet Name: abwr
Reference: LLNL Report to NRC, Sept. 7, 1992
Unit:
Drawing Number:
Print Date/Time: 9/8/1995 09:40:44
Description:

Item number	Component	Function	Failure Mode	Impact on System	Detection	Remarks
6	Division I ESF TLU 1 or 2	Receives trip inputs from the various DTMs. Performs voting and other logic to produce system trips.	The TLU failure cannot be detected by other modules. The TLU cannot initiate any protection system. The LDs do not switch to one-out-of-one voting.	The portion of the ESF systems solely controlled by this TLU cannot be automatically initiated.	By surveillance.	Placing one of the Divisions II, III or IV in Sensors Bypass mode reduces the operating TLUs to two-out-of-three voting with no other effects on the system.
7	RCIC Isolation	Isolate the PCIC piping and equipment from the RPV if a leak is detected in the RCIC piping and equipment.	The RCIC isolation valves fail to close.	The leak will continue until the operator takes manual initiation to control the isolation valves. RCIC isolation annunciation will show RCIC isolated, when (in actuality) RCIC has not been isolated.	By surveillance.	
8	Manual start / stop for Pumps	Allows manual control of the pumps from the control room.	Multiplexer in the same division as the pump fails.	No manual control of the pump.	By surveillance.	The logic shows manual start / stop switches for pumps interlocked with valve positions. No switches local to the pump are shown in the logic. Thus the loss of the multiplexer may prohibit all control of pumps needing interlock signals from the mplxr

HAZOPtimizer by Arthur D. Little, Inc.

Company: LLNL
Facility: GE ABWR Reactor Design
HAZOP Date: 12/09/1994
Leader/Secretary:
Process:
Team Members:

Sheet Name: abwr
Reference: LLNL Report to NRC, Sept. 7, 1992
Unit:
Drawing Number:
Print Date/Time: 9/8/1995 09:40:45
Description:

Item number	Component	Function	Failure Mode	Impact on System	Detection	Remarks
9	Diesel Generators	Provide electric power to the protection system equipment when normal power is lost.	Loss of one diesel generator or the ability to initiate one generator during LOCA.	No impact on system.	By surveillance.	GE documents show the diesel generators being initiated automatically from the protection system during a LOCA, but the logic on how this occurs is incomplete.

PRELIMINARY HAZARD ANALYSIS
ABC Chemicals
VCM Plant Conceptual Design

FILE NAME = ABCCHEM.DBF

SHEET# 1 **HA CODE#:** 1 **ANALYST:** D. Lawrence
ID DATE: 12/21/94 **REV DATE:** 12/22/94 **RESOLUTION DATE:** / /
REVIEWED BY: **CONCUR:**
LIFE CYCLE: Design
SUBSYSTEM: Chlorination
COMPONENT: Chlorine line
GENERIC HAZARD: Toxic release

HAZARD SCENARIO:
Chlorine line gasket / packing leak

EFFECT:
Small chlorine release on site

RISK ASSESSMENT:

SEVERITY:	INITIAL	FINAL
I. CATASTROPHIC		
II. CRITICAL		
III. MARGINAL		
IV. NEGLIGIBLE	X	

PROBABILITY:	INITIAL	FINAL
A. FREQUENT		
B. PROBABLE	X	
C. OCCASIONAL		
D. REMOTE		
E. IMPROBABLE		

RAC CODES:	
INITIAL RAC	4
FINAL RAC	

REMARKS OR COMMENTS:

PRELIMINARY HAZARD ANALYSIS
ABC Chemicals
VCM Plant Conceptual Design

FILE NAME = ABCCHEM.DBF

SHEET# 2 **HA CODE#:** 2 **ANALYST:** D. Lawrence
ID DATE: 12/21/94 **REV DATE:** 09/08/95 **RESOLUTION DATE:** / /
REVIEWED BY: **CONCUR:**
LIFE CYCLE: Design
SUBSYSTEM: Chlorination
COMPONENT: Chlorine line
GENERIC HAZARD: Toxic release

HAZARD SCENARIO:
Chlorine line rupture (ie, vehicle accident, blocked-in line)

EFFECT:
Large chlorine release, major on-site impact. Potential off-site impacts.

RISK ASSESSMENT:

SEVERITY:	INITIAL	FINAL	PROBABILITY:	INITIAL	FINAL	RAC CODES:	
I. CATASTROPHIC	X		A. FREQUENT			INITIAL RAC	1
II. CRITICAL			B. PROBABLE			FINAL RAC	
III. MARGINAL			C. OCCASIONAL	X			
IV. NEGLIGIBLE			D. REMOTE				
			E. IMPROBABLE				

REMARKS OR COMMENTS:

RECOMMENDATIONS TO MINIMIZE RISK:

1) **Verify chlorine line is evacuated whenever the VCM plant is down for extended time.**

HA CODE#: 2 ACTION DUE: NOT REQ'D
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: DROPPED
Status History:
12/21/94 Recommendation identification date
01/14/95 Recommendation proved not feasible; dropped.

2) **Provide valves and interlocks to positively isolate the line in the event of a rupture.**

HA CODE#: 2 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

3) **Train VCM plant personnel to respond to chlorine releases.**

HA CODE#: 2 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

4) **Equip VCM plant personnel with PPE for chlorine.**

HA CODE#: 2 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

5) **Do not bury chlorine pipeline.**

HA CODE#: 2 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

PRELIMINARY HAZARD ANALYSIS
ABC Chemicals
VCM Plant Conceptual Design

FILE NAME = ABCCHEM.DBF

SHEET# 3 **HA CODE#:** 3 **ANALYST:** D. Lawrence
ID DATE: 12/21/94 **REV DATE:** 12/22/94 **RESOLUTION DATE:** / /
REVIEWED BY: **CONCUR:**
LIFE CYCLE: Design
SUBSYSTEM: Chlorination
COMPONENT: Chlorination reactor
GENERIC HAZARD: Toxic release

HAZARD SCENARIO:
Direct clorination reactor exotherm

EFFECT:
Large chlorine / EDC / ethylene release. Depending on reactor size and operating conditions, potential off-site impacts.

RISK ASSESSMENT:

SEVERITY:	INITIAL	FINAL	PROBABILITY:	INITIAL	FINAL	RAC CODES:	
I. CATASTROPHIC	X		A. FREQUENT			INITIAL RAC	2
II. CRITICAL			B. PROBABLE			FINAL RAC	
III. MARGINAL			C. OCCASIONAL				
IV. NEGLIGIBLE			D. REMOTE	X			
			E. IMPROBABLE				

REMARKS OR COMMENTS:

RECOMMENDATIONS TO MINIMIZE RISK:

1) **Consider moving VCM plant west of Plant Road.**

HA CODE#: 3 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

2) **Perform dispersion studies to assess off-site impact of chlorine / EDC release due to exotherm.**

HA CODE#: 3 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:

12/21/94 Recommendation identification date

3) **Verify reactor pressure relief system can handle this release.**

HA CODE#: 3 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

PRELIMINARY HAZARD ANALYSIS
ABC Chemicals
VCM Plant Conceptual Design

FILE NAME = ABCCHEM.DBF

SHEET# 4 **HA CODE#:** 4 **ANALYST:** D. Lawrence
ID DATE: 12/21/94 **REV DATE:** 12/22/94 **RESOLUTION DATE:** / /
REVIEWED BY: **CONCUR:**
LIFE CYCLE: Design
SUBSYSTEM: Chlorination
COMPONENT: Chlorination reactor
GENERIC HAZARD: Toxic release

HAZARD SCENARIO:
Direct chlorination reactor rupture.

EFFECT:
Large chlorine / EDC / ethylene release. Depending on reactor size and operating conditions, potential off-site impacts.

RISK ASSESSMENT:

SEVERITY:	INITIAL	FINAL	PROBABILITY:	INITIAL	FINAL	RAC CODES:	
I. CATASTROPHIC	X		A. FREQUENT			INITIAL RAC	2
II. CRITICAL			B. PROBABLE			FINAL RAC	
III. MARGINAL			C. OCCASIONAL				
IV. NEGLIGIBLE			D. REMOTE	X			
			E. IMPROBABLE				

REMARKS OR COMMENTS:

RECOMMENDATIONS TO MINIMIZE RISK:

1) **Minimize inventory of chlorine / EDC in reactor.**

HA CODE#: 4 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

PRELIMINARY HAZARD ANALYSIS
ABC Chemicals
VCM Plant Conceptual Design

FILE NAME = ABCCHEM.DBF

SHEET# 5 **HA CODE#:** 5 **ANALYST:** D. Lawrence
ID DATE: 12/21/94 **REV DATE:** 12/22/94 **RESOLUTION DATE:** / /
REVIEWED BY: **CONCUR:**
LIFE CYCLE: Design
SUBSYSTEM: Chlorination
COMPONENT: Chlorination reactor
GENERIC HAZARD: Toxic release

HAZARD SCENARIO:
Direct chlorination reactor relief valve lift.

EFFECT:
Potential large EDC / chlorine / ethylene release.

RISK ASSESSMENT:

SEVERITY:	INITIAL	FINAL	PROBABILITY:	INITIAL	FINAL	RAC CODES:	
I. CATASTROPHIC	__	__	A. FREQUENT	__	__	INITIAL RAC	1
II. CRITICAL	X	__	B. PROBABLE	X	__	FINAL RAC	__
III. MARGINAL	__	__	C. OCCASIONAL	__	__		
IV. NEGLIGIBLE	__	__	D. REMOTE	__	__		
			E. IMPROBABLE	__	__		

REMARKS OR COMMENTS:

RECOMMENDATIONS TO MINIMIZE RISK:

1) **Verify the reactor pressure relief system incinerator and scrubber are sized to handle this release.**

HA CODE#: 5 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

PRELIMINARY HAZARD ANALYSIS
ABC Chemicals
VCM Plant Conceptual Design

FILE NAME = ABCCHEM.DBF

SHEET# 6 **HA CODE#:** 6 **ANALYST:** D. Lawrence
ID DATE: 12/21/94 **REV DATE:** 12/22/94 **RESOLUTION DATE:** / /
REVIEWED BY: **CONCUR:**
LIFE CYCLE: Design
SUBSYSTEM: Chlorination
COMPONENT: EDC Storage sphere
GENERIC HAZARD: Toxic release

HAZARD SCENARIO:
EDC storage sphere rupture.

EFFECT:
Large release of EDC, potential off-site impact. Potential river contamination.

RISK ASSESSMENT:

SEVERITY:	INITIAL	FINAL	PROBABILITY:	INITIAL	FINAL	RAC CODES:	
I. CATASTROPHIC	X		A. FREQUENT			INITIAL RAC	2
II. CRITICAL			B. PROBABLE			FINAL RAC	
III. MARGINAL			C. OCCASIONAL				
IV. NEGLIGIBLE			D. REMOTE	X			
			E. IMPROBABLE				

REMARKS OR COMMENTS:

RECOMMENDATIONS TO MINIMIZE RISK:

1) **Consider moving EDC sphere away from river.**

HA CODE#: 6 ACTION DUE: / /
POINT OF CONTACT: RESPONSIBLE ORG:
REC CODE:

STATUS: OPEN
Status History:
12/21/94 Recommendation identification date

PRELIMINARY HAZARD ANALYSIS
ABC Chemicals
VCM Plant Conceptual Design

FILE NAME = ABCCHEM.DBF

SHEET# 8 **HA CODE#:** **ANALYST:**
ID DATE: / / **REV DATE: / /** **RESOLUTION DATE: / /**
REVIEWED BY: **CONCUR:**
LIFE CYCLE:
SUBSYSTEM:
COMPONENT:
GENERIC HAZARD:

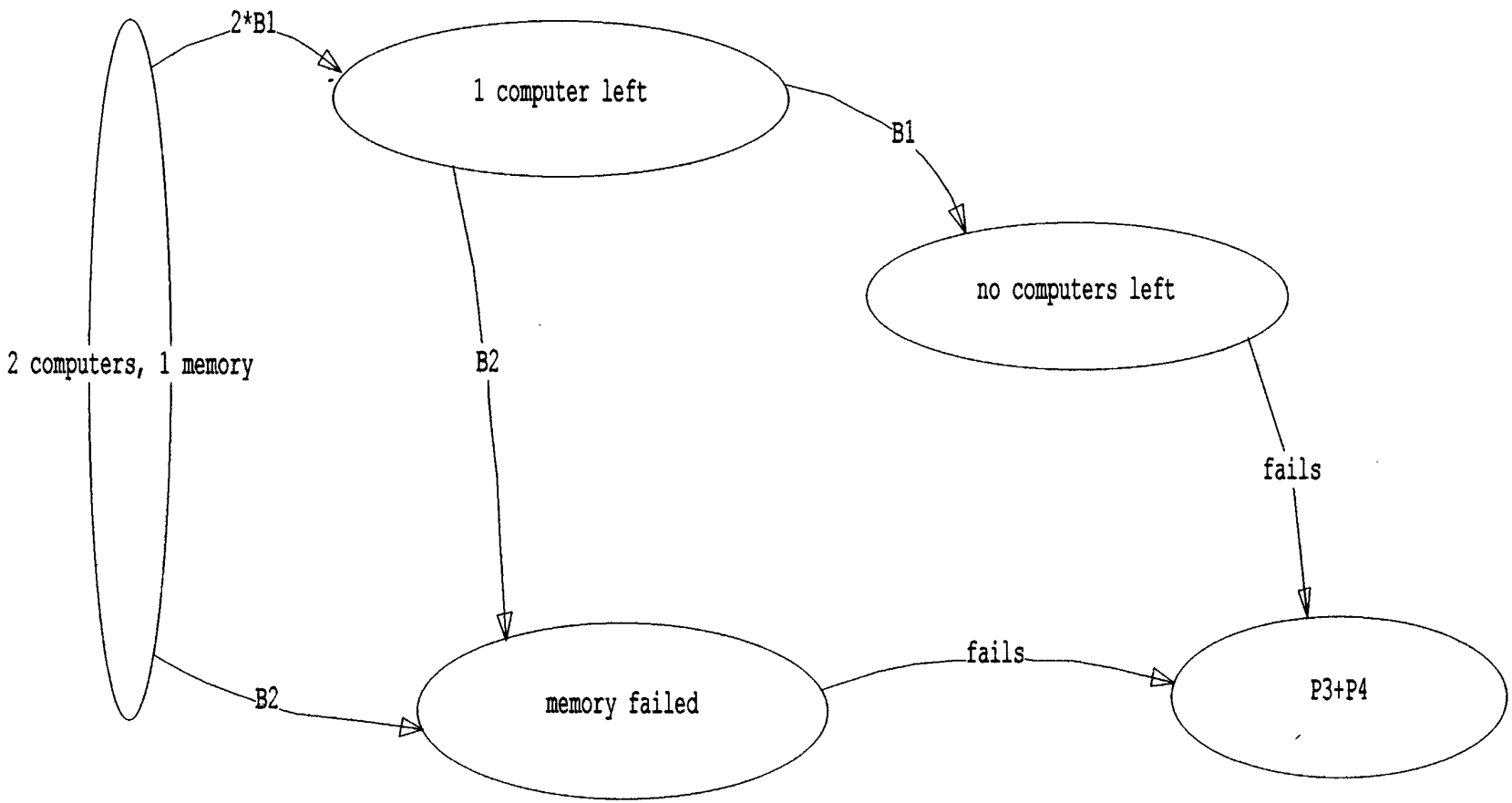
HAZARD SCENARIO:

EFFECT:

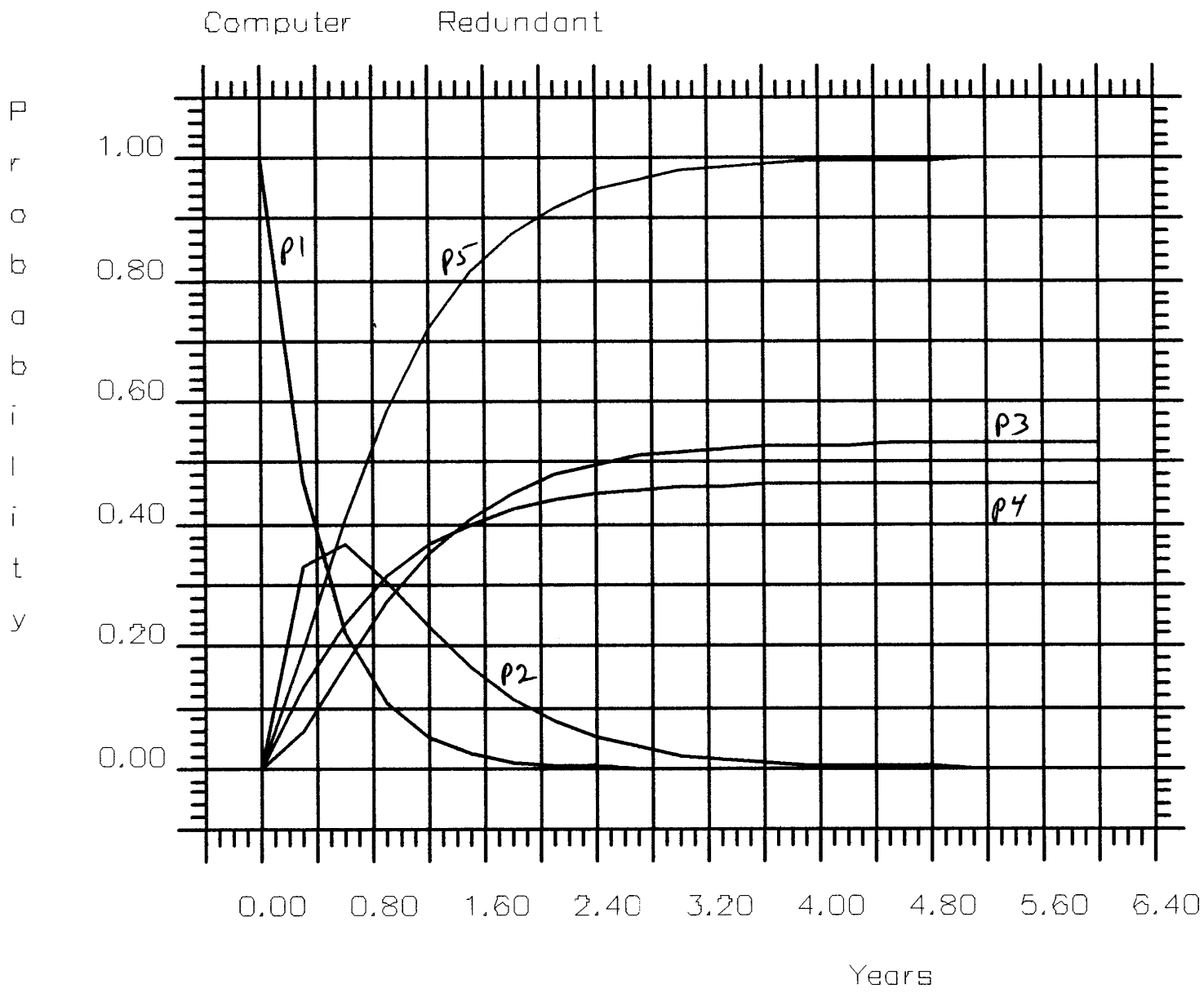
RISK ASSESSMENT:

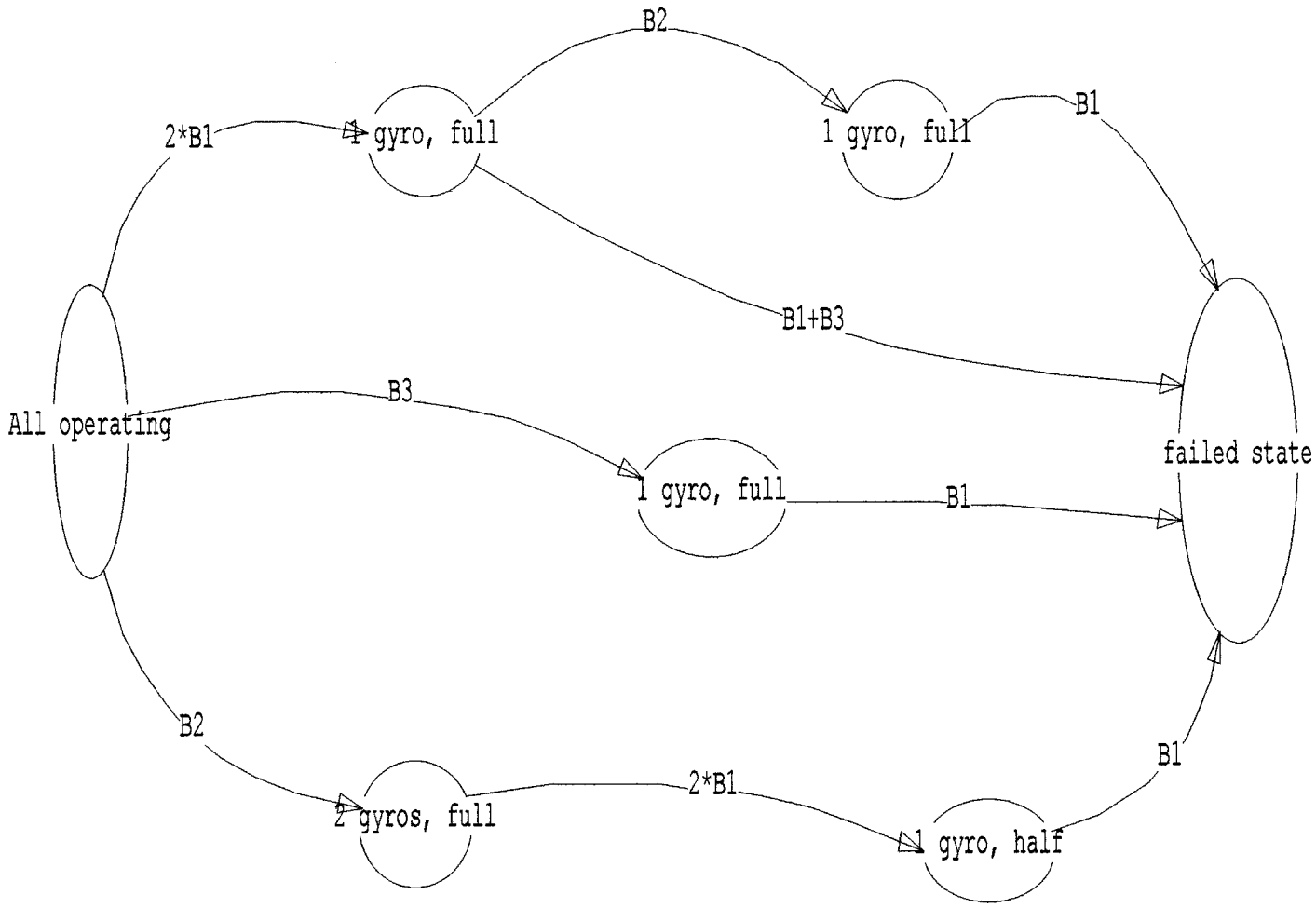
SEVERITY:	INITIAL	FINAL	PROBABILITY:	INITIAL	FINAL	RAC CODES:	
I. CATASTROPHIC	__	__	A. FREQUENT	__	__	INITIAL RAC	__
II. CRITICAL	__	__	B. PROBABLE	__	__	FINAL RAC	__
III. MARGINAL	__	__	C. OCCASIONAL	__	__		
IV. NEGLIGIBLE	__	__	D. REMOTE	__	__		
			E. IMPROBABLE	__	__		

REMARKS OR COMMENTS:

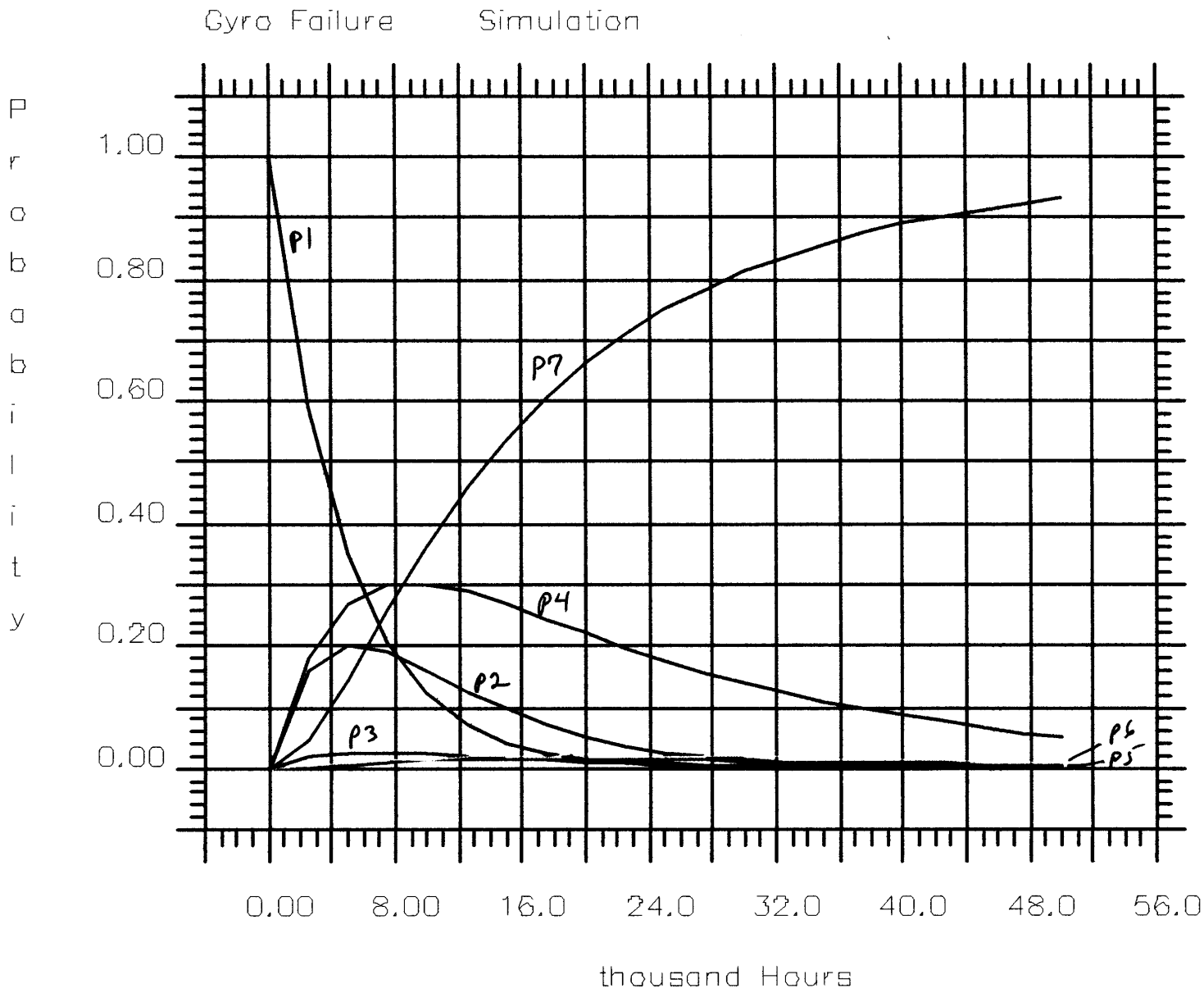


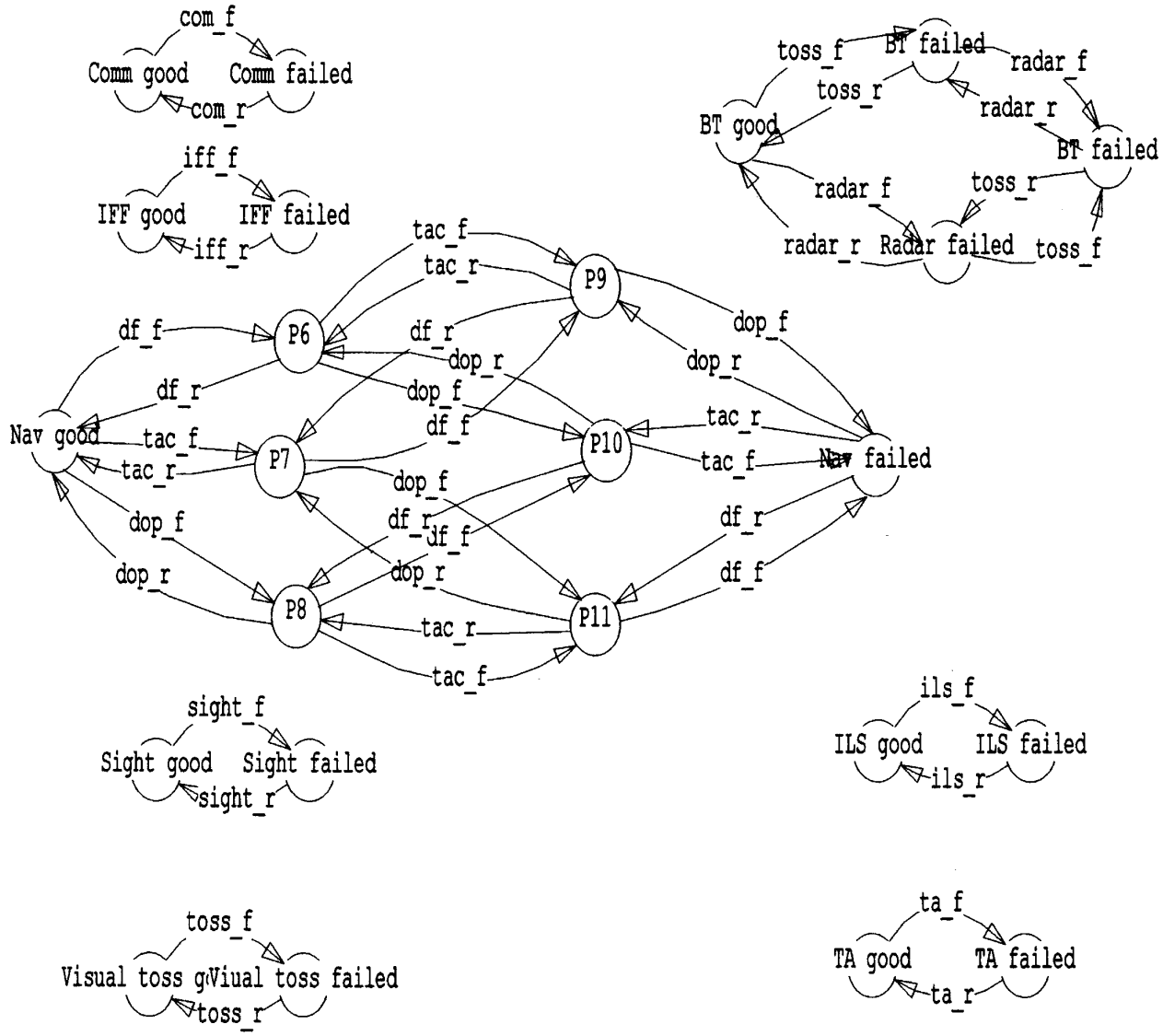
Computer Failure


Computer Redundant
Probability
1.00
0.80
0.60
0.40
0.20
0.00
P1
P5
P3
P4
P2
0.00
0.80
1.60
2.40
3.20
4.00
4.80
5.60
6.40
Years

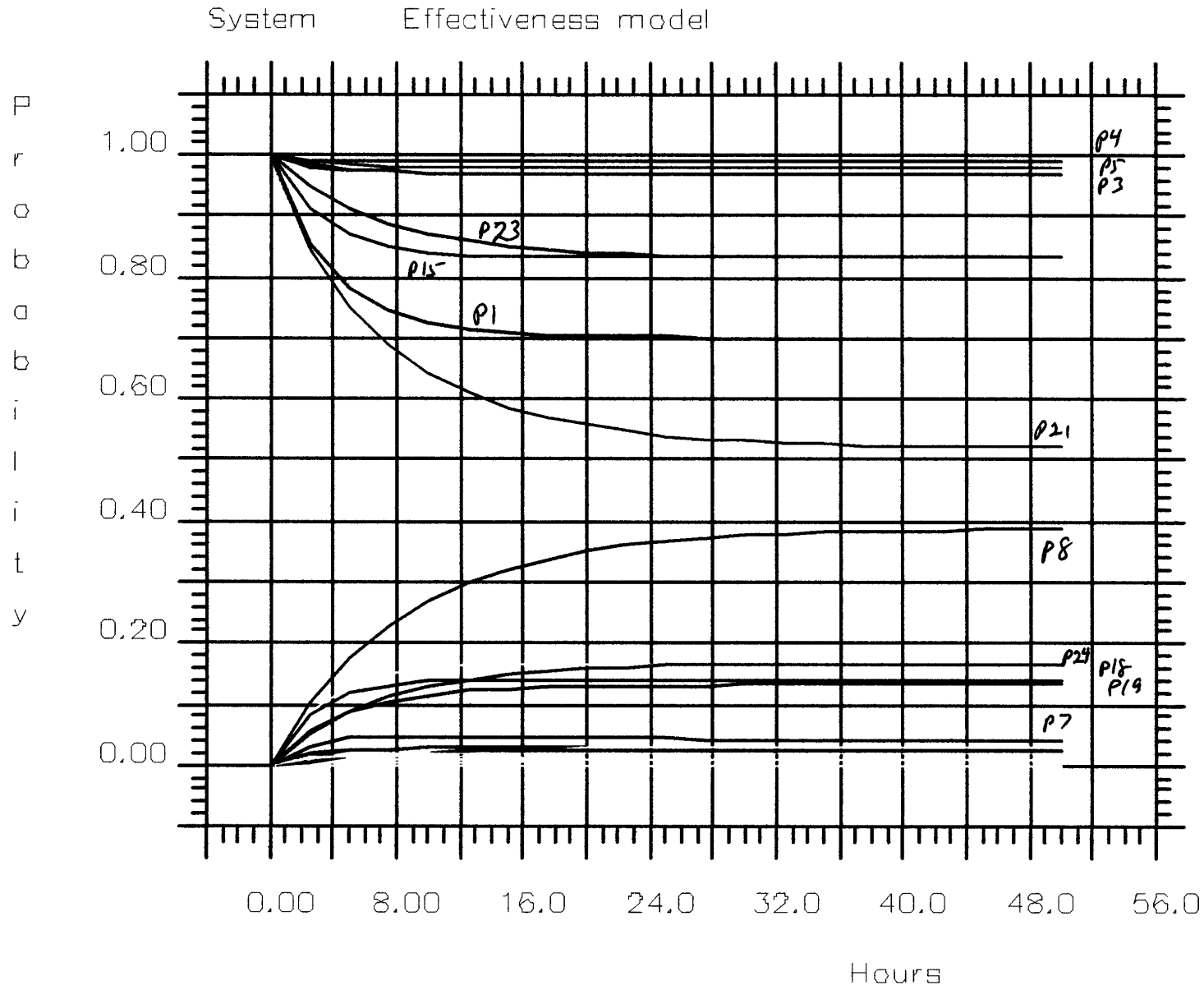


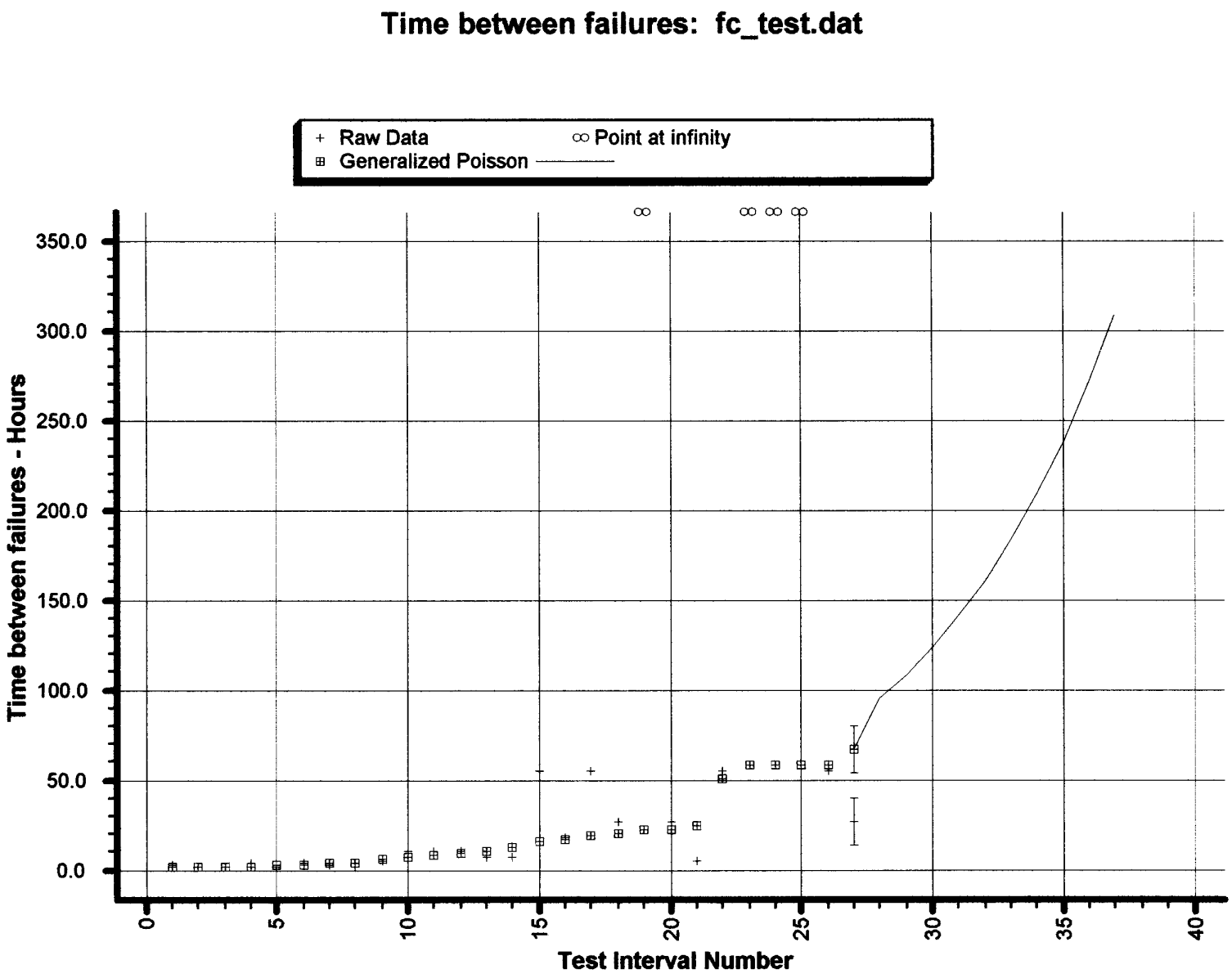
Gyro Failure

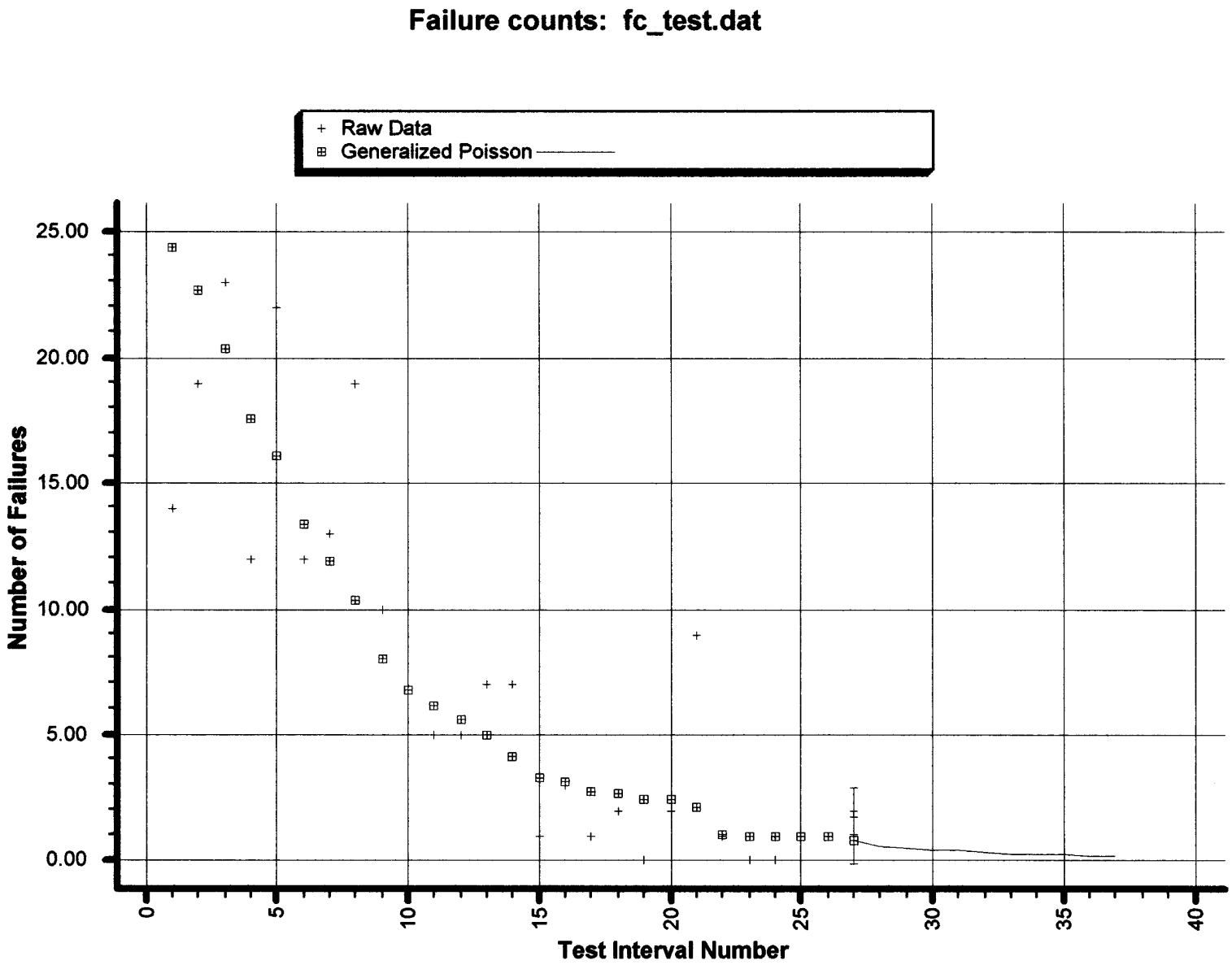

Gyro Failure Simulation
Probability
1.00
0.80
0.60
0.40
0.20
0.00
P1
P7
P4
P2
P3
P6
P5
0.00
8.00
16.0
24.0
32.0
40.0
48.0
56.0
thousand Hours

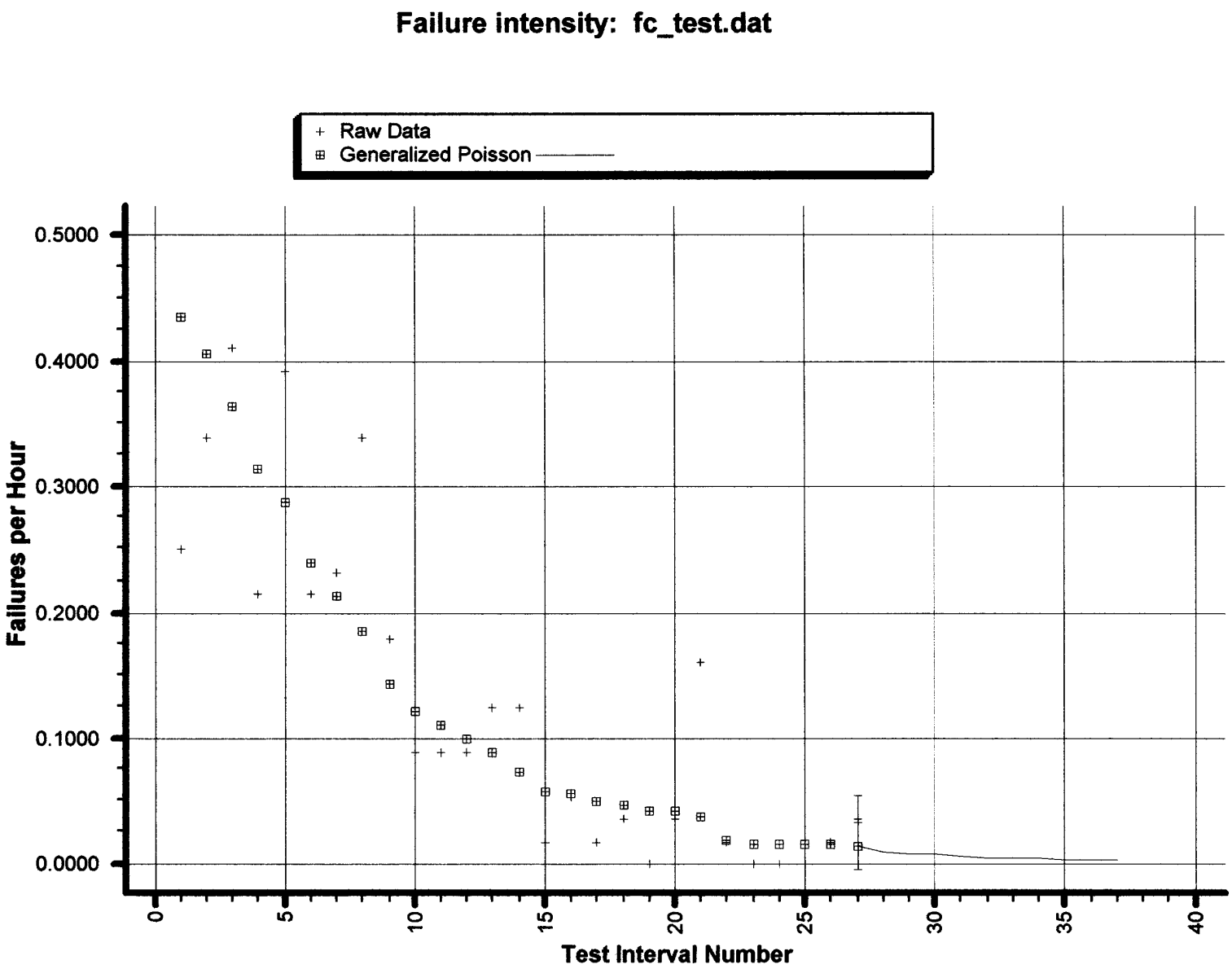


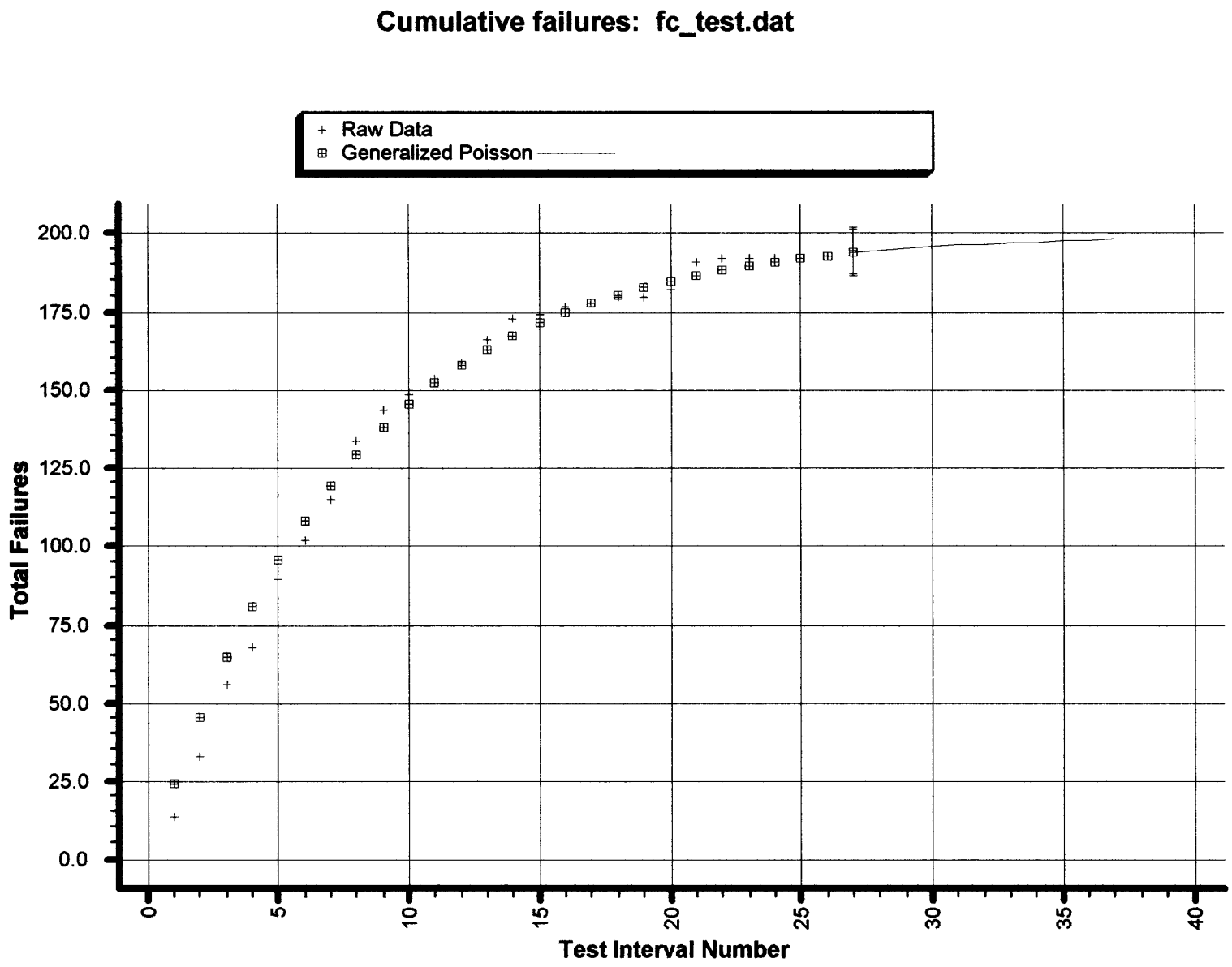
System Effectiveness

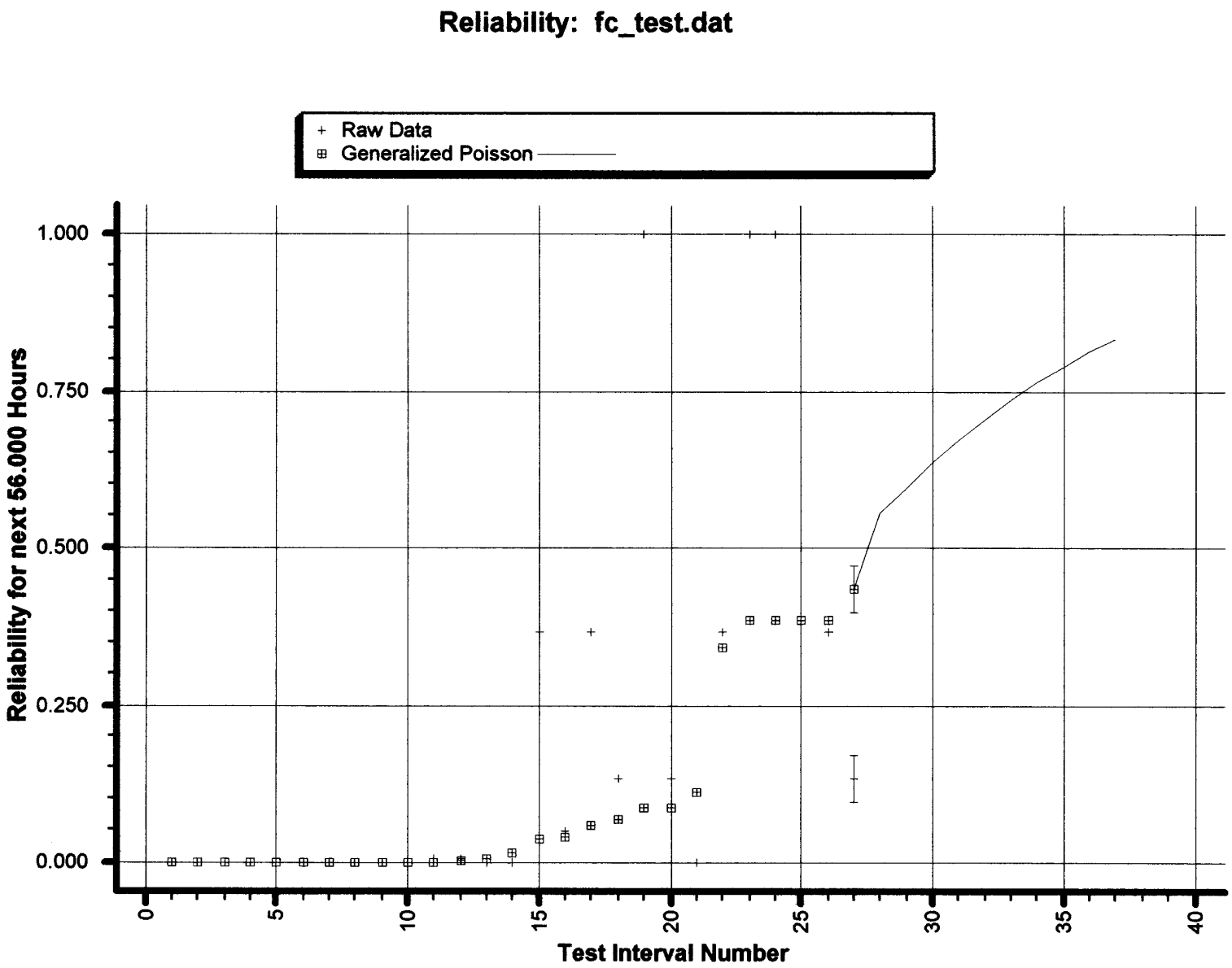

System Effectiveness model
Probability
1.00
0.80
0.60
0.40
0.20
0.00
0.00
8.00
16.0
24.0
32.0
40.0
48.0
56.0
Hours
P4
P5
P3
P23
P15
P1
P21
P8
P24
P18
P19
P7

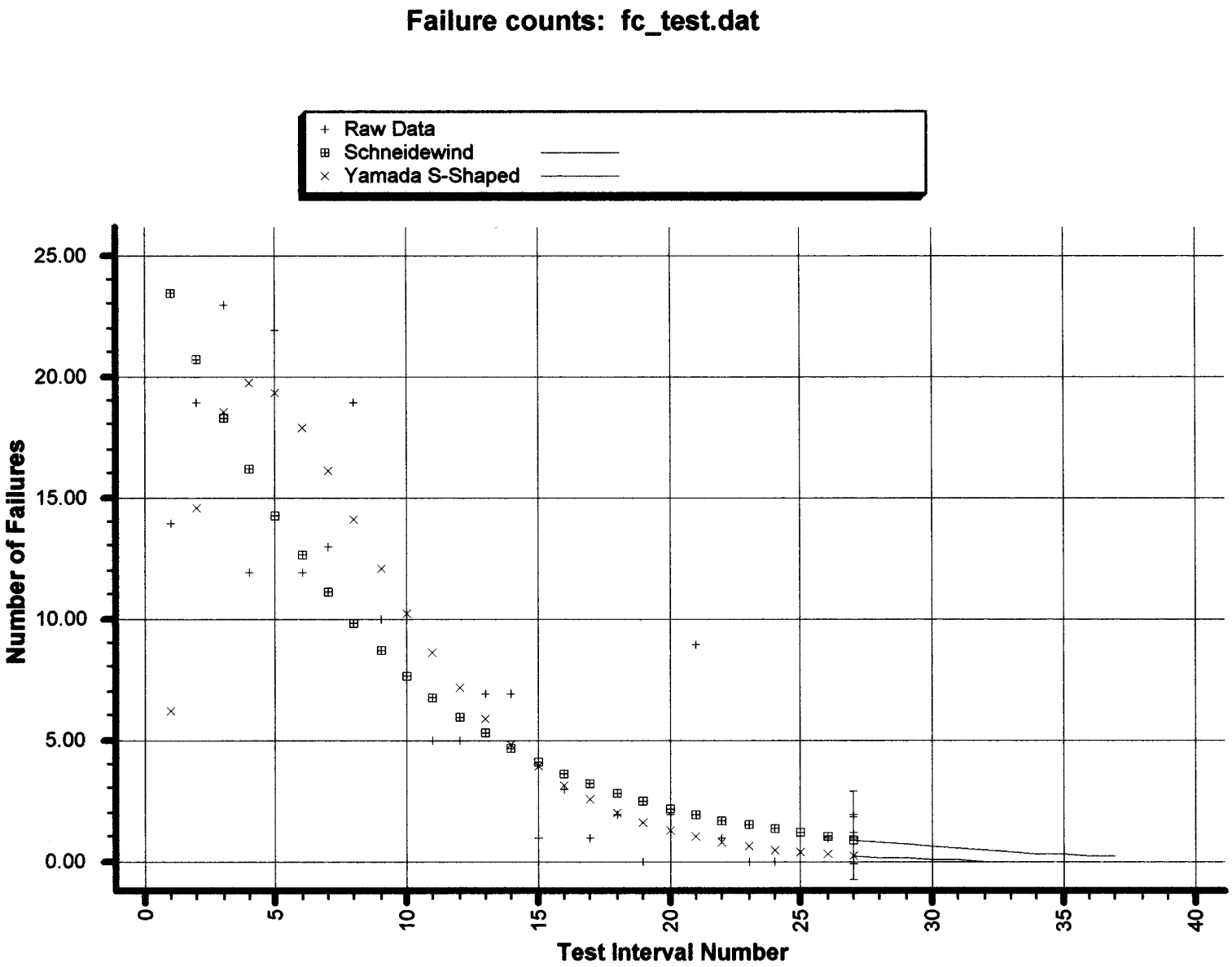

Time between failures: fc_test.dat
Raw Data
Generalized Poisson
Point at infinity
Time between failures - Hours
350.0
300.0
250.0
200.0
150.0
100.0
50.0
0.0
0
5
10
15
20
25
30
35
40
Test Interval Number


Failure counts: fc_test.dat
Raw Data
Generalized Poisson
Number of Failures
25.00
20.00
15.00
10.00
5.00
0.00
0
5
10
15
20
25
30
35
40
Test Interval Number


Failure intensity: fc_test.dat
Raw Data
Generalized Poisson
Failures per Hour
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
0
5
10
15
20
25
30
35
40
Test Interval Number


Cumulative failures: fc_test.dat
Raw Data
Generalized Poisson
Total Failures
200.0
175.0
150.0
125.0
100.0
75.0
50.0
25.0
0.0
0
5
10
15
20
25
30
35
40
Test Interval Number


Reliability: fc_test.dat
Raw Data
Generalized Poisson
Reliability for next 56.000 Hours
1.000
0.750
0.500
0.250
0.000
0
5
10
15
20
25
30
35
40
Test Interval Number


Failure counts: fc_test.dat
Raw Data
Schneidewind
Yamada S-Shaped
Number of Failures
25.00
20.00
15.00
10.00
5.00
0.00
0
5
10
15
20
25
30
35
40
Test Interval Number