

A11104 556590

NIST
PUBLICATIONS

NISTIR 5589

A Study on Hazard Analysis in High Integrity Software Standards and Guidelines

**Laura M. Ippolito
Dolores R. Wallace**

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

QC
100
.U56
NO.5589
1995

NIST

A Study on Hazard Analysis in High Integrity Software Standards and Guidelines

**Laura M. Ippolito
Dolores R. Wallace**

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

January 1995



U.S. DEPARTMENT OF COMMERCE
Ronald H. Brown, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arati Prabhakar, Director

ABSTRACT

This report presents the results of a study on hazard analysis, especially software hazard analysis, in high integrity software standards and guidelines. It describes types of system hazard analysis (that influence software), types of software hazard analysis, techniques for conducting hazard analysis (along with some of their advantages and disadvantages), and other practices and processes that should be employed in order to ensure the safety of software.

KEYWORDS

High integrity software, software assurance, software development, software engineering, software hazard analysis, software quality, software reliability, software safety, system hazard analysis.

TABLE OF CONTENTS

| | <u>Page</u> |
|--|-------------|
| EXECUTIVE SUMMARY | vii |
| ABBREVIATIONS | ix |
| 1. INTRODUCTION | 1 |
| 1.1. Review Process | 2 |
| 2. CONTEXT FOR SOFTWARE HAZARD ANALYSIS | 5 |
| 2.1. Criticality Assessment | 5 |
| 2.2. Types of Software Related System Hazard Analyses | 7 |
| 2.3. Types of Software Hazard Analysis | 7 |
| 3. HAZARD ANALYSIS TECHNIQUES | 11 |
| 3.1. Code Walk-Throughs | 13 |
| 3.2. Event Tree Analysis | 14 |
| 3.3. Hazard and Operability Studies | 15 |
| 3.4. Nuclear Safety Cross Check Analysis | 16 |
| 3.5. Petri Nets | 17 |
| 3.6. Software Failure Mode, Effects, and Criticality Analysis | 18 |
| 3.7. Software Fault Tree Analysis | 20 |
| 3.8. Software Sneak Analysis | 23 |
| 3.9. Additional References for Software Hazard Analysis Techniques | 25 |
| 4. SOFTWARE QUALITY | 27 |
| 4.1. Software Development | 27 |
| 4.2. Software Assurance | 29 |
| 4.3. Software Engineering Practices | 30 |
| 5. CONCLUSIONS | 33 |
| 6. REFERENCES | 35 |
| APPENDIX A. BIBLIOGRAPHY OF HIGH INTEGRITY | 41 |
| A.1. Standards and Guidelines | 41 |
| A.2. Books | 52 |
| A.3. Papers | 53 |

List of Tables

| | <u>Page</u> |
|--|-------------|
| Table 2-1. Standards Specifying Software Hazard Analysis | 8 |
| Table 3-1a. Standards Specifying Software Hazard Analysis Techniques | 11 |
| Table 3-1b. Standards Specifying Software Hazard Analysis Techniques (cont.) | 12 |
| Table 3-2. Additional Techniques' References | 25 |

List of Figures

| | <u>Page</u> |
|---|-------------|
| Figure 2-1 System & Software Hazard Analyses | 6 |
| Figure 3-1 Event Tree | 15 |
| Figure 3-2 NSCCA Criticality Matrix | 16 |
| Figure 3-3 Petri Net | 18 |
| Figure 3-4 Headers for a Failure Mode Analyses Matrix | 19 |
| Figure 3-5 Fault Tree Symbols | 21 |
| Figure 3-6 System Fault Tree. | 21 |
| Figure 3-7 Software Fault Tree | 22 |
| Figure 4-1 Software Development & Assurance Processes. | 28 |

EXECUTIVE SUMMARY

The National Institute of Standards and Technology (NIST) has built a database¹ of standards, guidelines, technical papers, and books which was used in a study on hazard analysis, especially software hazard analysis, in high integrity software standards and guidelines. For this report these documents were examined in order to determine the following:

- What is software hazard analysis?
- Why is software hazard analysis important?
- When is software hazard analysis performed?
- What techniques are used for hazard analysis? Which ones can be used for software hazard analysis?
- What are the advantages and disadvantages of the different hazard analysis techniques?
- What standards and guidelines require or recommend that software hazard analysis be performed, and what methodologies are required or recommended?
- Is software hazard analysis sufficiently addressed by existing standards and guidelines?

This report describes types of system hazard analysis (preliminary hazard list; preliminary hazard analysis; system hazard analysis; and, operating and support hazard analysis) and types of software hazard analysis (software requirements hazard analysis; software design hazard analysis; code-level software hazard analysis; software safety testing; software/user interface analysis; and, software change hazard analysis).

This report includes techniques that are mentioned in some standards and/or guidelines for conducting hazard analysis (code walk-throughs; event tree analysis; hazard and operability studies; nuclear safety cross check analysis (NSCCA); Petri nets; software failure mode, effects, and criticality analysis; software fault tree analysis (SFTA); and, software sneak analysis (SSA)), along with some of their advantages and disadvantages.

Of the techniques investigated for this report, only NSCCA, SFTA, and SSA have been used specifically for software *hazard* analysis. And, while these techniques originated from similar techniques for hardware, they are relatively young and untried for software. More investigation and experimentation is needed to determine the usefulness, scope and cost effectiveness of these techniques.

¹This database includes current U.S. and international documents that address high integrity software systems.

This study also discusses the development of the software. It is not enough to conduct software hazard analysis or other safety analyses. Software needs to be developed using specific software development and software assurance processes to protect against or mitigate failure of the software. A complete software safety standard would at least reference other standards that address these mechanisms and would include a software safety policy identifying required functionality to protect against or mitigate failure.

As software is included in more and more critical systems (e.g., nuclear power plants, medical devices and transportation systems) the need for software safety programs becomes crucial. These software safety programs should consist of not only software safety analyses, but methodologies that assist in the assurance of developing quality software.

In summary, the following issues need to be addressed regarding the safety of software:

- Standards need to require software safety programs that include software hazard analysis and other safety analyses, and software development and software assurance processes.
- Techniques for safety analysis need to be updated, and the use of software engineering methodologies and computer-aided software engineering (CASE) tools needs to be considered for software safety analysis.

ABBREVIATIONS

The following acronyms and abbreviations are used in this report:

| | |
|--------|--|
| CASE | Computer-Aided Software Engineering |
| CSHA | Code-level Software Hazard Analysis |
| ETA | Event Tree Analysis |
| FMA | Failure Mode Analyses |
| FMEA | Failure Mode and Effects Analysis (system) |
| FMECA | Failure Mode, Effects, and Criticality Analysis (system) |
| FTA | Fault Tree Analysis (system) |
| HAZOP | HAZard and Operability studies |
| NIST | National Institute of Standards and Technology |
| NPP | Nuclear Power Plant |
| NRC | Nuclear Regulatory Commission (United States) |
| NSCCA | Nuclear Safety Cross Check Analysis |
| NSO | Nuclear Safety Objectives |
| O&SHA | Operating and Support Hazard Analysis |
| OOD | Object Oriented Design |
| PHA | Preliminary Hazard Analysis |
| PHL | Preliminary Hazard List |
| PRA | Probabilistic Risk Analysis |
| SDHA | Software Design Hazard Analysis |
| SFMEA | Software Failure Mode and Effects Analysis |
| SFMECA | Software Failure Mode, Effects, and Criticality Analysis |
| SFTA | Software Fault Tree Analysis |
| SHA | System Hazard Analysis |
| SCA | Sneak Circuit Analysis |
| SCHA | Software Change Hazard Analysis |
| SQA | Software Quality Assurance |
| SRHA | Software Requirements Hazard Analysis |
| SSA | Software Sneak Analysis |
| SV&V | Software Verification and Validation |
| USAF | United States Air Force |

1. INTRODUCTION

This report presents the results of a study on hazard analysis, especially software hazard analysis, in high integrity software standards and guidelines. A *hazard* is an (unsafe) "*condition* that may lead to an unintended event that causes an undesirable outcome" [WALLACE]. The *unintended event* that results in an undesirable outcome is a *mishap* [MIL882B]. For example, a driver of a car ignores warning lights at a railroad crossing and drives the car onto the tracks. The hazard is the presence of the car and train on the track at the same time. The unintended event (mishap) is the train colliding with the car. The undesirable outcome is the probable loss of life and damage to the car and train.

Hazard analysis is the process of identifying and evaluating the hazards of a system, and then making change recommendations that would either eliminate the hazard or reduce its risk to an "acceptable level" [MIL882B]. Traditionally this did not necessarily involve an analysis of software. However, as software is being included in more systems whose failure can cause physical harm, the need for identifying software hazards becomes increasingly important. *Software* hazard analysis makes recommendations to eliminate or control software hazards and hazards related to interfaces between the software and the system (includes hardware and human components). It includes analyzing the requirements, design, code, user interfaces, and changes. Software hazards may occur if the software is improperly developed (designed), the software dispatches incorrect information, or the software fails to transmit information when it should have.

NIST reviewed high integrity standards and guidelines to determine the extent to which software hazard analysis is addressed by the standards community. Hazard analysis techniques are also discussed in this report, along with some of their advantages and disadvantages. There are few techniques developed specifically for conducting software hazard analysis. Many are based on methods used to analyze hardware safety. However, software and hardware differ in many ways; hardware errors are often random failures due to age or poor workmanship whereas software is only subject to design faults [LEVINSON]. Therefore, it is very important to tailor techniques correctly to software and its specific characteristics.

Software hazard analysis is a part of software safety. Software safety also includes, for example, testing, identifying safety-critical, single- and multiple-failure sequences and determining software safety requirements [LEVESON86]. The following sample software safety program is suggested by [LEVESON89]:

- Software Development Management Responsibilities
- Software Hazard Analysis
- Establishing Software Safety Requirements
- Software Safety Requirements Review
- Software Safety Design Concepts

- Software Design and Recovery Analysis
- Software Safety Design Review
- Code Verification and Validation
- Assessment of Risk.

Software safety should not be confused with software reliability. Reliability is the ability of a system to perform its required functions under stated conditions for a specified period of time [IEEE610]. Safety is the probability that conditions (hazards) that can lead to a mishap do not occur, whether or not the intended function is performed [LEVESON86]. Reliability is interested in all possible software errors, while safety is concerned only with those errors that cause actual system hazards [LEVESON86]. Some of the techniques described in section 3 may actually demonstrate reliability more than safety. Software safety and software reliability are part of software quality. Quality is the degree to which a system meets specified requirements, and customer or user needs or expectations [IEEE610]. Assuring software quality is discussed in section 4.

This report does address computer security, however some hazard analysis techniques may also be used to reveal security threats. And, techniques used in security checks may be useful in hazard analysis. Further work is needed to explore these possibilities.

1.1. Review Process

The National Institute of Standards and Technology (NIST) has built a database² of standards, guidelines, technical papers, and books to assist in a study of current information regarding standards and practices for the assurance of high integrity software systems. ([NIST204] and [NIST209] document other parts of this study.) For this part of the study these documents were examined in order to determine the following:

- What is software hazard analysis?
- Why is software hazard analysis important?
- When is software hazard analysis performed?
- What techniques are used for hazard analysis? Which ones can be used for software hazard analysis?
- What are the advantages and disadvantages of the different hazard analysis techniques?

²This database includes current U.S. and international documents that address high integrity software systems.

- What standards and guidelines require or recommend that software hazard analysis be performed, and what methodologies are required or recommended?
- Is software hazard analysis sufficiently addressed by existing standards and guidelines?

Software engineering and software safety assessment are both relatively new fields and are constantly evolving. Software engineering standards and guidelines, including those for safety systems, are also changing as new terminology, methodologies, and techniques are developed. Taken as a group, software engineering standards and guidelines are in flux and inconsistent. This study includes different versions of some standards and guidelines since sometimes a newer version does not include all the information from a previous version. For example, guidance in standard N, not included in standard N+1, may appear again in standard N+2.

Section 2 of this report explains the types of system hazard analysis and the types of software hazard analysis, including how the two relate or interact. Section 3 describes hazard analysis techniques. Section 4 discusses assuring the quality of software. Section 5 presents the conclusions of this review. Appendix A lists the standards, guidelines, technical papers, and books examined for this review.

2. CONTEXT FOR SOFTWARE HAZARD ANALYSIS

Software hazard analysis is directly related to system hazard analysis since it depends upon system hazard analysis for its inputs. Figure 2-1 (based on [MIL882B]³) gives an approximate representation of when system and software hazard analyses are performed relative to the system and software life cycles. (The **software** life cycle processes and **software** hazard analyses are in **bold** face type.) Software hazard analysis should:

- respond to every hazard identified in system hazard analysis,
- ensure that the operation of the software does not interfere with the goals or operation of the system, and
- evaluate and make recommendations to mitigate how software could hinder the goals or operation of the system.

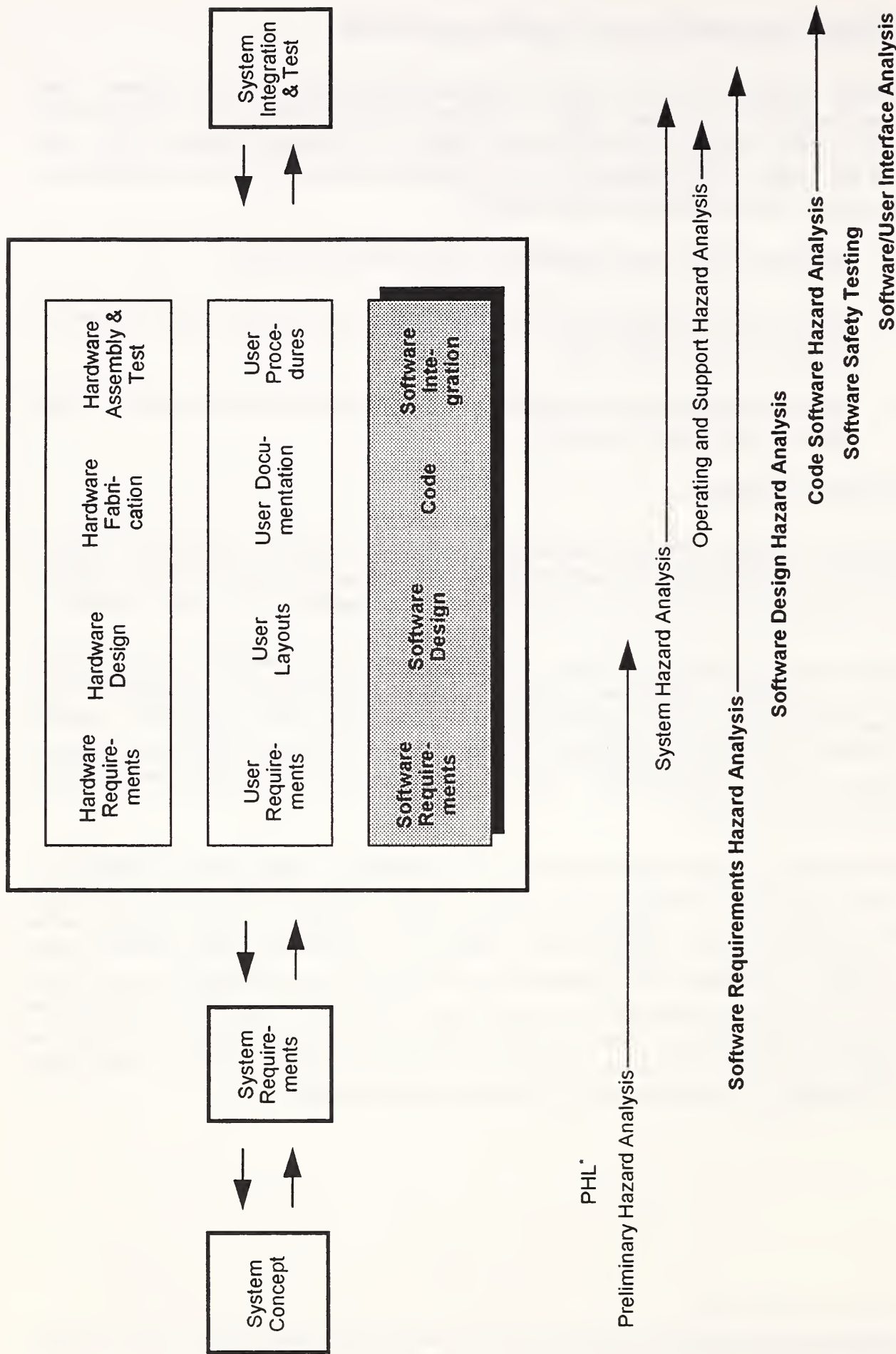
2.1. Criticality Assessment

This report does not explore criticality assessment in detail; it is only mentioned since it affects when and how any hazard analysis is performed. In software engineering "criticality" refers to the degree of impact that an item has on the development or operation of a system [IEEE610].

There are two types of criticality assessment. Criticality analysis of the software assigns degrees of analysis (e.g., superficial, detailed and thorough) to levels of software (e.g., criticality is high [failure of software could result in personal injury, loss of life, or loss of property], medium [failure of software could result in damage to equipment], or low [failure of software would not affect people or property]) [IEEEP1059]. The idea is that the more critical the software system, the more thorough the analysis.

Criticality of hazards can also be determined. It is unrealistic to expect that all hazards of a system can be removed or controlled. The idea is to assess the criticality of the hazards and concentrate on eliminating or mitigating the most critical. Often no action will be taken regarding a hazard with a low criticality level. However, a hazard with a high criticality level will be mitigated. For example, "the vulnerability of a reactor to an earthquake is high and the opportunity for occurrence, particularly on fault lines, is sufficiently high with consequences sufficiently severe that actions are taken, such as building away from fault lines, to mitigate the hazard" [NIST190]. In some cases, the probability of a hazard occurring and the results of the criticality assessment are combined into a probability risk assessment.

³[MIL882B] gives the most thorough explanation of system and software hazard analyses; most of the other documents reviewed for this study only provide a cursory explanation of these analyses.



PHL*

*Preliminary Hazard List
 **Software Change Hazard Analysis

Figure 2-1 System & Software Hazard Analyses.

2.2. Types of Software Related System Hazard Analyses

The following is a brief description based on [MIL882B]⁴ of each type of system hazard analysis. Only *hazard* analyses that include specific analysis of software or its interfaces to the system are included. For more information on system hazard analyses, see [MIL882B].

A preliminary hazard list (PHL) identifies hazardous areas of the system. It occurs during the system concept definition life cycle phase and consists of examining the system concept. The PHL may be used to establish the scope of future system and software hazard analyses.

The preliminary hazard analysis (PHA) uses the PHL to evaluate the hazards (i.e., criticality, probability) of the system. It begins in the earliest system life cycle phase while requirements are being considered. (More complete hazard analyses cannot be completed until after functions have been allocated to hardware, human, and software.) The PHA includes, but is not limited to, examining interfaces between hardware and software controls, software contribution to system mishaps, system design criteria to control software commands and responses, and software fail safe design considerations. This hazard analysis is the foundation for future system and software hazard analyses. The PHA ends when the component level of system hazard analysis is able to begin.

System hazard analysis (SHA) is twofold. On the one hand, SHA identifies those components of the system whose functioning (or lack of functioning) could result in a hazard. This aspect of SHA examines how component operation or failure affects the system. It begins when the components are designed sufficiently, and is updated as their design matures. If software affects a component, each product of the software life cycle is an input to this part of the SHA. Problems with the software are reported during the particular software life cycle process. This analysis is updated in the case of system OR software design changes.

SHA also identifies hazards by analyzing the system as an entity, concentrating on component interfaces. This aspect of SHA examines how the system operation and failure modes affect the safety of the system and its components. SHA at the system level starts as the system design matures, close to the design review, and is updated until the system design is complete.

The operating and support hazard analysis (O&SHA) identifies and evaluates hazards resulting from operations being performed by humans. It begins before the system test and integration life cycle phase (early enough to provide inputs to the design). O&SHA identifies safety requirements necessary to eliminate hazards or mitigate the risk of hazards. It includes identifying changes needed in software requirements, and warnings and emergency procedures necessitated by the failure of a software-controlled operation.

2.3. Types of Software Hazard Analysis

Table 2-1 is intended to provide a general idea of the extent to which hazard analysis is addressed by the standards community. It lists the standards and guidelines that specifically

⁴See footnote 3.

address software hazard analysis separately from system hazard analysis, software development or software verification and validation (i.e., standards and guidelines that contain the words *software hazard analysis* or *software safety analysis*). For example, [MIL882C] has incorporated software safety analyses into its system level analyses, instead of describing them separately as was done in [MIL882B]. Therefore, [MIL882B] appears in Table 2-1 and [MIL882C] does not. Table 2-1 does include standards and guidelines that reference software safety analysis, like the various drafts of [IEEEP1228], since software hazard analysis is a part of software safety analysis.

Table 2-1. Standards Specifying Software Hazard Analysis

| STANDARD | TYPE OF SOFTWARE HAZARD ANALYSIS |
|---|---|
| [AFISC] "Software System Safety" | <ul style="list-style-type: none"> ■ Preliminary Software Hazard Analysis ■ Follow-on Software Hazard Analysis |
| [IEEEP1228] ⁵ "(DRAFT) Standard for Software Safety Plans" | <ul style="list-style-type: none"> ■ Software Safety Requirements Analysis ■ Software Safety Design Analysis ■ Software Safety Code Analysis ■ Software Safety Test Analysis ■ Software Safety Change Analysis |
| [JPL93] "Software Systems Safety Handbook" | <ul style="list-style-type: none"> ■ Software Requirements Hazard Analysis ■ Software Top-Level and Detailed Design Hazard Analysis ■ Code-level Hazard Analysis ■ Interface Hazard Analysis ■ Software Change Hazard Analysis |
| [MIL882B] "System Safety Program Requirements" | <ul style="list-style-type: none"> ■ Software Requirements Hazard Analysis ■ Top-Level Design Hazard Analysis ■ Detailed Design Hazard Analysis ■ Code-Level Software Hazard Analysis ■ Software Safety Testing ■ Software/User Interface Analysis ■ Software Change Hazard Analysis |
| [MOD0056'89] "(DRAFT) Requirements for the Analysis of Safety Critical Hazards" | <ul style="list-style-type: none"> ■ Software Hazard Analysis ■ Software Classification ■ Software Functional Analysis |
| [NSS1740] "(Interim) NASA Software Safety Standard" | <ul style="list-style-type: none"> ■ Software Safety Requirements Analysis ■ Software Safety Architectural Design Analysis ■ Software Safety Detailed Design Analysis ■ Code Safety Analysis ■ Software Test Safety Analysis ■ Software Change Analysis |
| [SOFTENG] "Standard for Software Engineering of Safety Critical Software" | <ul style="list-style-type: none"> ■ Code Hazards Analysis |

⁵These types of analysis are included in all of the different versions of [IEEEP1228].

The documents reviewed for this study vary broadly in their scope and coverage of hazard analysis; Table 2-1 is not intended to indicate that a consensus exists among the standards. If two different versions of a standard or guideline include basically the same information, only the most recent is listed. Also note that terminology differs among the standards (e.g., code hazards analysis vs. code-level hazard analysis).

The following descriptions of software hazard analyses are based on [MIL882B]⁶.

The software requirements hazard analysis (SRHA) 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, operator, user, and diagnostic manuals. It begins in the system requirements phase of the system life cycle. The PHL and PHA are inputs to this analysis. SRHA 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 system specifications, software requirements specification, software design documents, software test plan, software configuration management plan, and the project management plan. The results of the SRHA are presented at the system requirements review (draft results), system design review (updated results), and software requirements review (final results).

Software design hazard analysis (SDHA) identifies safety-critical software components. It starts after the software requirements review and should be mostly complete before starting software coding. The PHA and SRHA are inputs to this analysis. SDHA 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 SDHA are presented at the software design review.

Code-Level software hazard analysis (CSHA) identifies how to eliminate hazards or mitigate the risk of hazards. It starts at the beginning of the code phase of the software life cycle and continues until after system testing has been completed. The SDHA is the input to this analysis. CSHA 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 CSHA are presented at the test readiness review⁷ (some CSHA results are given to programmers during coding).

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

⁶See footnote 3.

⁷The Test Readiness Review provides a final check to ensure that all test cases have been defined and are correct; all changes have been made to the code from previous analyses; the test facility is ready and test procedures have been checked; and, schedules, personnel and facility arrangements have been agreed to and checked.

components under normal conditions and 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. Software safety testing results in identifying corrections necessary to eliminate hazards or mitigate the risk of hazards, and then retests the corrected software under the same conditions. Testing of the software at the system level starts following a successful test readiness review.

The software/user interface analysis manages hazards that were not eliminated or controlled in the system design phase. For example, recommendations may be made to the design that provide for the detection of the hazard and a warning to the operator, a recovery from a situation caused by a hazard, the ability to terminate an event or process.

Software change hazard analysis (SCHA) analyzes all changes made to the software to determine their impact on safety. Software hazard analysis and testing is performed on any changes made to the requirements, design, code, systems, equipment, and test documentation to ensure that the change does not create a hazard or effect any existing hazards, and that the change is properly incorporated into the code. This software hazard analysis is performed after all other software hazard analyses have been completed; it checks the changes resulting from preceding software hazard analyses.

3. HAZARD ANALYSIS TECHNIQUES

There are several techniques that can be used to conduct hazard analysis. Some have been used to evaluate hardware (e.g., failure mode and effects analysis, fault tree analysis, and sneak circuit analysis) and are relatively new to the software arena. Each technique will need to be tailored to specific applications based on its size and cost [HANSEN]. A thorough software hazard analysis will require application of more than one software hazard analysis technique due to the various strengths and weaknesses of each technique [MIL882B].

Table 3-1 lists the standards and guidelines that refer to particular hazard analysis techniques⁸ and *imply* that these techniques can be used for *software* (see footnotes). Whether or not all of these techniques can be used to perform software hazard analysis is uncertain.⁹ If two different versions of a standard or guideline include basically the same information, only the most recent is included.

Table 3-1a. Standards Specifying Software Hazard Analysis Techniques

| STANDARD | TECHNIQUE(S) |
|--|--|
| [AFISC] ¹⁰ "Software System Safety" | <ul style="list-style-type: none"> ■ Nuclear Safety Cross-Check Analysis ■ Petri Nets ■ Software Fault Tree (Soft Tree) - Uses of Fault Trees: Cutset, Quantitative, Common Cause Analysis ■ Software Sneak Circuit Analysis (Desk Checking, Code Walk-Through, Structural Analysis, Proof of Correctness) |
| [FDA89] ¹¹ "(DRAFT) Reviewer Guidance for Computer-Controlled Devices" | <ul style="list-style-type: none"> ■ Code Walk-Throughs ■ Failure Mode, Effects, and Criticality Analysis ■ Fault Tree Analysis |
| [FDA91] ¹² "Reviewer Guidance for Computer-Controlled Medical Devices Undergoing 510(k) Review" | <ul style="list-style-type: none"> ■ Failure Mode, Effects, and Criticality Analysis |

⁸This table only includes *standards* or *guidelines* that specifically *name* techniques that can be applied to software. Therefore, not all of the references cited in this report will appear in this table.

⁹The authors welcome further information regarding application of these techniques in software hazard analysis.

¹⁰[AFISC] states that these are techniques that can be used to help provide a thorough software hazard analysis.

¹¹[FDA89] states that these techniques are examples of practices appropriate to software design and development.

¹²[FDA91] refers to this as a software failure analysis technique.

Table 3-1b. Standards Specifying Software Hazard Analysis Techniques (cont.)

| STANDARD | TECHNIQUE(S) |
|--|---|
| [IECWG9'91] ¹³ "Software for Computers in the Application of Industrial Safety-Related Systems" | <ul style="list-style-type: none"> ■ Cause Consequence Diagrams ■ Event Tree Analysis ■ Failure Mode, Effects, and Criticality Analysis ■ Fault Tree Analysis ■ Hazard and Operability Study ■ Monte-Carlo Simulation |
| [IEEEP1228-C] ¹⁴ "Draft Standard for Software Safety Plans" [IEEEP1228-D] [IEEEP1228-E] | <ul style="list-style-type: none"> ■ Event Tree Analysis ■ Failure Modes and Effects Analysis ■ Fault Tree Analysis ■ Petri Nets ■ Sneak Circuit Analysis |
| [JPL93] ¹⁵ "Software Systems Safety Handbook" | <ul style="list-style-type: none"> ■ Petri Nets ■ Software Fault Tree Analysis |
| [MIL882B] ¹⁶ "System Safety Program Requirements" | <ul style="list-style-type: none"> ■ Code Walk-Throughs ■ Cross Reference Listing Analysis ■ Design Walk-Throughs ■ Nuclear Safety Cross-Check Analysis ■ Petri Net Analysis ■ Software Fault Tree Analysis ■ Software/Hardware Integrated Critical Path Analysis ■ Software Sneak Analysis |
| [MOD0055'89] ¹⁷ "(DRAFT) Requirements for the Procurement of Safety Critical Software in Defence Equipment" | <ul style="list-style-type: none"> ■ Common Cause Failure Analysis ■ Event Tree Analysis ■ Failure Modes and Effects Analysis ■ Fault Tree Analysis |

[MOD0056'91] includes a list of criteria that hazard analysis techniques should meet. While this study was unable to fully determine whether or not the techniques described in this report adhere to this criteria, it is mentioned here for those interested in investigating the techniques themselves. The following list identifies some of that criteria verbatim from [MOD0056'91]:

- The results from the technique shall enhance the understanding of the way risks arise, are prevented or reduced.

¹³[IECWG9'91] recommends these techniques for all safety integrity levels of software.

¹⁴These techniques were included in the appendix of the earlier versions of this standard and later deleted versions.

¹⁵[JPL93] refers to these as error detection and prevention techniques.

¹⁶[MIL882B] states that these are some of the current techniques available to conduct software system safety analysis, and that a thorough software hazard analysis will require application of more than one of these techniques.

¹⁷These techniques are listed in an appendix of [MOD0055'89] entitled "Hazard Analysis Techniques".

- The technique shall permit the modelling and evaluation of a wide range of failure modes.
- The technique shall enable systematic analysis to be carried out in a manner that is auditable, repeatable and verifiable.
- The technique shall be appropriately matched to the expertise of the staff.
- The technique shall be appropriate for a system of the given complexity operating in the given domain and containing the given hazards.
- The technique shall give valid results using data of the quality and quantity actually available.
- The technique shall be appropriate for the particular life cycle phase at which it is to be applied.
- Tools of adequate integrity or standard proformas shall be available commercially to support the technique, or shall be able to be supplied.
- Fully documented examples of its successful application shall exist or a written justification for its application shall be able to be supplied.
- The technique shall be defined by a national or international standard or a published reference book, or a definition and instructions for its application shall be able to be supplied.

This section describes code walk-throughs; event tree analysis; hazard and operability studies; nuclear safety cross check analysis; Petri nets; software failure mode, effects, and criticality analysis; software sneak analysis; and, software fault tree analysis. These are techniques that were the most commonly cited based on Table 3-1.

3.1. Code Walk-Throughs

This subsection describing code walk-throughs is taken from [NIST209]. No documentation was found that defined code walk-throughs specifically for software *hazard* analysis. A code walk-through is an evaluation technique in which a programmer leads one or more other members of the development team through a segment of code, while the other members ask questions and make comments about technique, style, and identify possible errors, violations of development standards, and other problems. Code walk-throughs are similar to inspections, but are less formal and have a different team composition. Other essential differences include the following:

- Participants are fellow programmers rather than representatives of other functions.
- Frequently no preparation.
- Standards usually ignored.

- Checklists are rarely used.
- Follow-up is often ignored.

Advantages of code walk-throughs include, but are not limited to:

- Less intimidating than formal inspections.
- Identifies the most error-prone sections of the program, so more attention can be paid to these sections during testing.
- Very effective in finding logic design and coding errors.

Disadvantages of code walk-throughs include, but are not limited to:

- Designers or programmers may feel walk-through is an attack on their character or work.

3.2. Event Tree Analysis

This subsection describing event tree analysis (ETA) is based primarily on [IECWG9'89], [IECWG10'89] and [RAHEJA91]. No documentation was found that defined ETA specifically for *software* use. ETA uses a bottom-up approach to model the effects of an event that may have serious repercussions. The initiating event is the root of the event tree. Two lines are drawn from the root, depicting the positive and negative consequences of the event. This is done for each subsequent consequence until all consequences are considered. Figure 3-1 gives an example of an event tree where the initiating event is railroad crossing lights failing to function. Event trees can be used to calculate the probabilities of each consequence, and can include timing information.

Advantages of ETA (from [IECWG9'89]) include, but are not limited to:

- Event trees are easy to draw once the sequence of events is established.
- Event trees are easy to understand.
- It is easy to compute probabilities.

Disadvantages of ETA (from [IECWG9'89]) include, but are not limited to:

- It may be difficult to identify all consequences.
- The event tree can become very large.

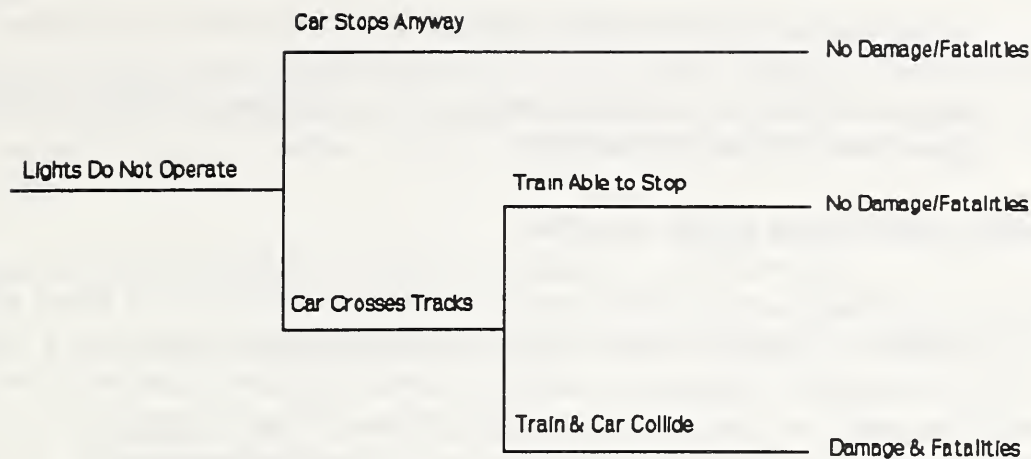


Figure 3-1 Event Tree.

3.3. Hazard and Operability Studies

This subsection describing hazard and operability studies (HAZOP) is based primarily on [IECWG9'89]. HAZOP examines the component of the system (can be adapted for use on software systems, however no documentation was found supporting HAZOP as a software *hazard* analysis technique) and its operation to identify failure modes that could lead to hazards. This analysis is conducted by engineers led by a hazard analysis techniques expert, and covers the entire project life cycle. First, the entire system installation is examined. Then, the components of the system are examined. Each component (and subcomponents, where necessary) has a checklist (drawn up prior to the start of the analysis) which includes questions like "What if it happens?" and "How can it happen?". If positive answers are given, more questions are asked, such as "What can be done about it?" and "Is there an alternative?". Formal review meetings are held periodically and comprehensive records are kept.

Advantages of HAZOP (from [IECWG9'89]) include, but are not limited to:

- Identifies potential hazards and their effects at every stage of the project life cycle.
- Eliminates failures or mitigates their risk levels at every stage of the project life cycle.
- May help the design team to decide on the need for an increase of redundancy, diversity.
- Focuses on the sensitive areas of the system which, on failure could lead to potentially hazardous consequences and this highlights where effort can be most effectively applied to improve the system safety, reliability and availability.

Disadvantages of HAZOP (from [IECWG9'89]) include, but are not limited to:

- Is time consuming and requires specific expertise which may increase costs.
- May be difficult to schedule meetings in a way that is both cost effective and agreeable to the project timetable.

3.4. Nuclear Safety Cross Check Analysis

This subsection describing nuclear safety cross check analysis (NSCCA) is based primarily on [AFISC], [HANSEN] and [LEVESON86]. NSCCA was created to conform to a United States Air Force (USAF) regulation dealing with nuclear systems; however, it can be tailored for other applications. Its purpose is to demonstrate that the software will not contribute to an unintended event. NSCCA consists of a technical component and a procedural component. The purpose of the technical process is to ensure that the system's safety requirements are met. The goal of the procedural component is to protect against sabotage, collusion, compromise or alteration of critical software components.

The technical component analyzes and tests the software. A criticality analysis is conducted to determine the extent to which each software function affects the nuclear safety objectives (NSO). The criticality analysis begins with deriving the NSOs based on Department of Defense directive 5030/15 and USAF regulation 122-10 pertaining to nuclear systems [AFISC]. Then, the software is decomposed to the lowest level of functions. Each function is examined to ascertain the extent to which it operates or controls a nuclear critical event. Functions that do not affect critical events are not further analyzed. A matrix is developed which plots the software functions against the NSOs, assigns influence ratings of high, medium or low, and gives recommendations for evaluation techniques. Figure 3-2 is an example of a criticality matrix (h = high, m = medium, l = low influence rating).

| SOFTWARE FUNCTION | NSO | | | EVALUATION TECHNIQUE | |
|-------------------|-----|---|---|----------------------|---|
| | 1 | 2 | 3 | A | B |
| 1 | m | h | h | √ | √ |
| 2 | | m | l | | √ |
| 3 | h | l | m | √ | |
| 4 | l | | h | √ | |

Figure 3-2 NSCCA Criticality Matrix.

The criticality analysis is used in deciding where to allocate resources in order to comply with the requirements. The NSCCA program plan is then developed which establishes the tools and

facilities requirements, analyses requirements, test requirements, test planning, and test procedures.

The procedural component implements security and control measures. This includes personnel security (background investigations, clearances), document security (configuration control), test facility security (secured media, access restrictions), and product control (configuration management).

Advantages of NSCCA include, but are not limited to, the following:

- Promotes independence and avoids "rubber-stamping" [AFISC].
- Conducted by an organization independent of the software developer [LEVESON86].
- Performed over the entire life cycle [LEVESON86].

Disadvantages of NSCCA include, but are not limited to, the following:

- The effectiveness of NSCCA is dependent upon the analyses and test procedures chosen [LEVESON86].

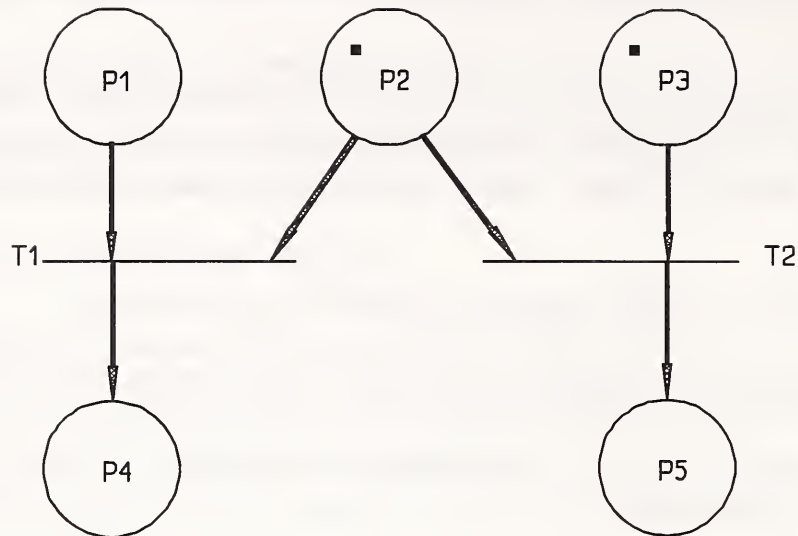
3.5. Petri Nets

This subsection on Petri nets is based primarily on [AFISC] and [LEVESON87]. No documentation was found that defined Petri nets specifically for software *hazard* analysis. Petri nets were invented in 1961 by Carl Petri to model systems mathematically. The system (including software systems) is modelled using conditions and events represented by state transition diagrams. Petri nets consist of places (conditions--represented by circles), transitions (events--represented by bars), inputs (pre-conditions--represented by arrows originating from places and terminating at transitions), outputs (post-conditions--represented by arrows originating from transitions and terminating at places), and tokens (indication of true condition--represented by dots). Figure 3-3 gives an example of a Petri net from [AFISC]. In this figure P1-5 are places and T1-2 are transitions. Since P2 and P3 contain tokens (their conditions are true) T2 can "fire". (Although P2 has a token, P1 does not, therefore T1 does not fire.) The tokens in P2 and P3 will then be removed and a new token placed in P5.

Petri nets can be "executed" to see how the design will actually work under certain conditions. Specifically, Petri nets can be used to determine all the states (including hazardous states) the system can reach, given an initial condition. Often Petri nets become too large to generate all states, however, the same backward analysis used in fault trees can be applied to Petri nets.

Advantages of Petri nets include, but are not limited to:

- Users describe systems with graphical notation; they are not concerned with the mathematical underpinnings of Petri nets [AFISC].



■ = token

Figure 3-3 Petri Net.

- Can be used early in the development life cycle which enables changes to be made relatively inexpensively [AFISC].
- Can be used at various levels of abstraction [AFISC].
- Can accommodate timing information [LEVESON87].
- Provide a language that allows the automatic generation of a specific simulation [TYSZER].

Disadvantages of Petri nets include, but are not limited to:

- Often Petri nets become too large to generate all states of the system [AFISC].
- Building Petri nets is non-trivial [LEVESON86].
- Can be difficult to analyze [LEVESON87].

3.6. Software Failure Mode, Effects, and Criticality Analysis

This subsection describing software failure mode, effects, and criticality analysis (SFMECA) and related analyses is based primarily on [HALL], [PES87] and [RAHEJA89]. No documentation was found defining SFMECA specifically for software *hazard* analysis. (However, the U.S. Patent and Trademark Office recently approved it for use as a software *reliability* technique

[MAZUR].) Software failure mode and effects analysis (SFMEA) and SFMECA are based on failure mode and effects analysis (FMEA) and failure mode, effects, and criticality analysis (FMECA) respectively, which analyze hardware. FMECA and SFMECA extend FMEA and SFMEA, respectively, by assigning a criticality level to each component failure. Since all four of these analyses are similar, they will be discussed in this subsection using the term failure mode analyses (FMA).

FMA reveal weak or missing requirements and help to identify latent software defects. The FMA use inductive reasoning to determine the effect on the system of a component (includes software instructions) failing in a particular failure mode. For example, if a function of a train crossing system is to turn on warning lights as a train approaches the crossing and leave the lights on until the train has past, some of its failure modes could be:

- the lights do not turn on when a train approaches
- the lights turn on though no train is coming
- the lights turn off too soon (before the train has fully crossed)
- the lights turn on too late (after the train has begun crossing).

The effect on the system of each component's failure in each failure mode would then be assessed by developing a matrix for each component, with headers similar to the one shown in Figure 3-4. The criticality factor, that is, the seriousness of the effect of the failure, can be used in determining where to apply the other analyses and testing resources.

| Component | Failure Mode & Causes | Effect on System | Criticality | Change/Action Required | Prevention & Control Safeguards |
|-----------|-----------------------|------------------|-------------|------------------------|---------------------------------|
|-----------|-----------------------|------------------|-------------|------------------------|---------------------------------|

Figure 3-4 Headers for a Failure Mode Analyses Matrix.

Advantages of FMA include, but are not limited to:

- Reveal unforeseen hazards since possible hazards do not need to be identified up front [PES87].
- Are systematic [PES87].

Disadvantages of FMA include, but are not limited to:

- Does not consider multiple failures [PES87].
- Can be time consuming and expensive [PES87].

FMA generally look at the hardware or software by identifying the failure effects that could occur at the highest system level. Then, these are tracked to successive lower functional levels until the lowest command and control function have been considered. The FMA can be considered a reliability technique also, because the FMA identify functional paths on which other verification techniques can be applied to assess the probability that the function will perform its intended function for a specified period of time.

3.7. Software Fault Tree Analysis

This subsection describing software fault tree analysis (SFTA) is based primarily on [LEVESON83] and [LEVESON91]. Additional information was obtained from [LEVESON86], [RAHEJA91], and others as referenced. SFTA is derived from Fault Tree Analysis (FTA) which has been used in the past for system hazard analysis. The most significant difference between the two is that FTA analyzes hardware and SFTA examines software. SFTA can be used in conjunction with FTA whereby hardware (system) and software fault trees are combined in order to analyze the entire system. This is significant since many hazards can be caused by a combination of a software error with a hardware or human failure. However, issues like the difference between a software fault and a hardware failure¹⁸ must be considered when linking trees [LEVINSON]. Because SFTA is so similar to FTA, both are addressed in this subsection.

FTA can begin once preliminary hazard analyses (PHL and/or PHA) have been performed. The analyst assumes that an already identified hazard has occurred and then works backward to discover the possible causes of the hazard. This is done by creating a fault tree, whose root is the hazard, using the symbols shown in Figure 3-5. The system fault tree is expanded until it contains at its lowest level basic events which cannot be further analyzed. An example of a simple, incomplete system fault tree is shown in Figure 3-6 (the hazard is that a car crosses a train track when a train is at the crossroads). A software fault tree, actually a subtree of the system fault tree, is created when software is identified as an event which contributes to a system hazard. Figure 3-7 gives a general idea of what types of information are included in a software fault tree at the code level.

The purpose of SFTA is to demonstrate that the software will not cause a system to reach an unsafe state, and to discover what environmental conditions would allow the system to reach an unsafe state. SFTA is often conducted on the code, but can also be applied at other stages of the life cycle process (e.g., requirements and design) [LEVESON91]. SFTA is not always applied to all of the code, only that portion which is safety critical.

Fault trees can be used to calculate the probability of a hazard (the top event) occurring, if the probabilities of lower events are known. This aides in determining which parts of the software (or system as a whole) are the most critical and therefore require more intensive safety analysis.

¹⁸From [IEEE610], a *hardware failure* is the inability of the system to perform its required functions within specified performance requirements; a *software fault* is an incorrect step, process, or data definition in a computer program.

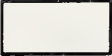



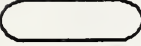


| | |
|---|--|
|        | <p>Event to be analyzed; always requires further analysis.</p> <p>Component level faults or independent basic events; no further analysis.</p> <p>Event normally expected to occur.</p> <p>Event not further analyzed due to lack of information or non-criticality.</p> <p>Condition; defines the state of the system.</p> <p>AND Gate; indicates when all inputs must occur to produce the output.</p> <p>OR Gate; indicates when at least one input must occur to produce the output.</p> |
|---|--|

Figure 3-5 Fault Tree Symbols.

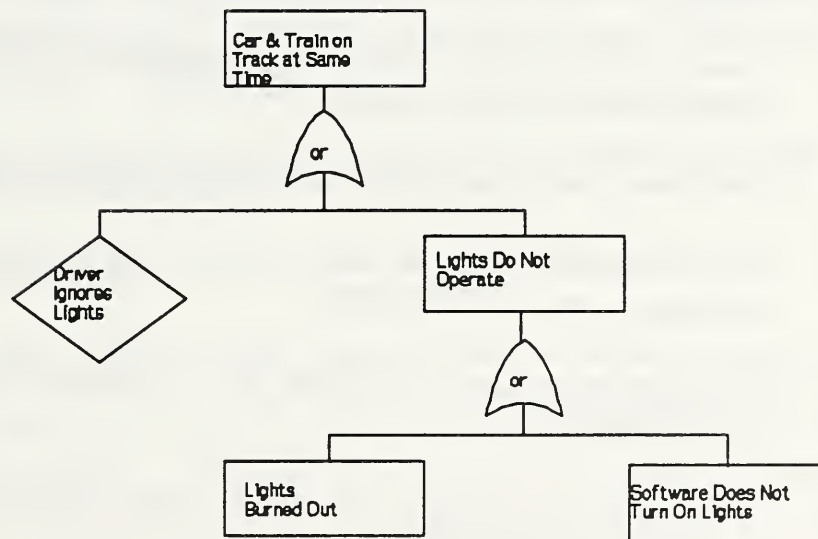


Figure 3-6 System Fault Tree.

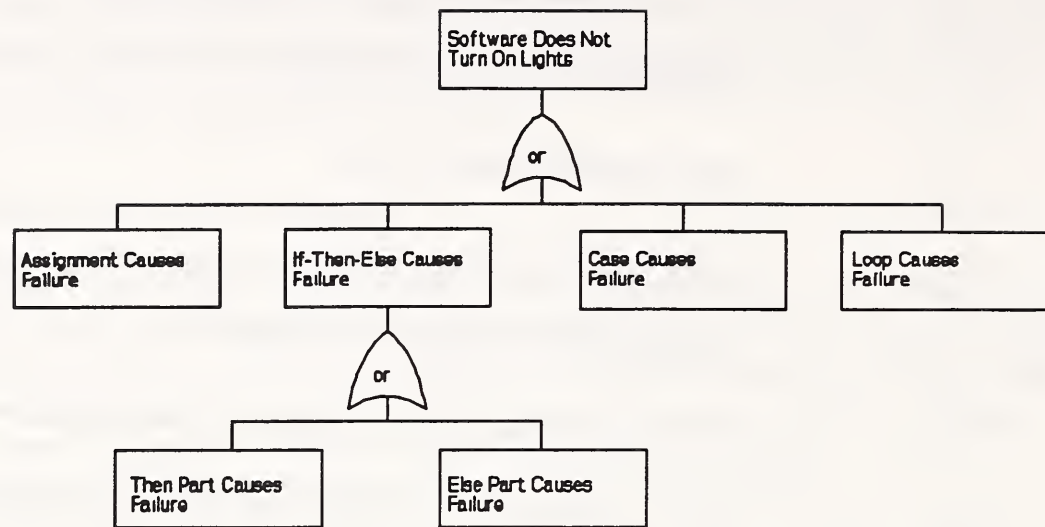


Figure 3-7 Software Fault Tree.

Advantages of SFTA include, but are not limited to, the following:

- Fault trees can reveal multiple failure sequences involving different parts of the system that can lead to hazards [EWICS3].
- Fault trees can reveal when a correct state becomes unsafe [LEVESON83].
- The information stored in fault trees can later be used for other processes, e.g, redesign, testing [LEVESON83].
- The development and use of fault trees is already known to hardware engineers [LEVESON83].
- Fault trees can examine the entire system (software, hardware and human interface) [LEVESON83].
- Investigates the consequences of machines failures or environmental changes as opposed to verification methods which assume machine functions operate correctly [LEVESON83].
- An entire fault tree does not need to be completed for the analysis to be useful [LEVESON91].
- Is systematic [IECWG9'89].

Disadvantages of SFTA include, but are not limited to, the following:

- It may be difficult to identify all hazards of a large system (suggests that SFTA should not be the only type of hazard analysis used) [LEVESON83].
- Is not practical on systems with large numbers of safety critical failures [LEVESON83].
- It is difficult to introduce timing information into fault trees [LEVESON86].
- May verify that the program will never reach an unsafe state, but does not address incorrect but safe states [LEVESON91].
- Depends on the ability of the analyst conducting the analysis [HANSEN].
- Fault trees may become very large and complex [IECWG9'89].

"(Software) fault tree analysis has already been applied to software-based systems with some promising results" [LEVINSON]. SFTA discovered a critical failure scenario in a program which controls the flight and telemetry for a University of California spacecraft that was not detected during testing [LEVESON83]. It has also been used on industrial projects including avionics systems for military aircraft, a nuclear power plant (NPP) shutdown system, medical systems, and various military and aerospace software systems [LEVESON91]. Engineers who performed SFTA on the NPP shutdown system (6,000 lines of Fortran and Pascal code) felt the technique was useful and easy to learn [LEVESON91].

Tools are being developed for SFTA, however, some SFTA users feel that human involvement was crucial to the process [LEVESON91].

3.8. Software Sneak Analysis

This subsection describing software sneak analysis (SSA) is based primarily on [AFISC], [CARLSEN], [GODOY], [PEYTON]. SSA is based on sneak circuit analysis (SCA), which was developed by Boeing Aerospace Company to evaluate electrical circuitry. Later Boeing adapted the technique to include software. SSA and SCA can be used together to analyze the entire system, which is called sneak analysis. As stated in section 3.6, analysis of the entire system allows hazards caused by a software and hardware, software and human, or hardware and human interaction to be detected.

Sneaks are latent design conditions or design flaws which have inadvertently been incorporated into electrical, software, and integrated systems designs [CARLSEN]. They are *not* caused by component failure. There are three main reasons (determined by Boeing [CARLSEN]) that sneaks occur:

- 1) system complexity -- the more complex a system is, the more sneaks it will likely contain

- 2) organizational complexity -- large, complex organizations (like those with many subcontractors) often develop products with many sneaks primarily because of the difficulty in accurately defining and assuring the consistency of the interfaces
- 3) rapidly changing technology -- when the time to analyze and test new systems is reduced, sneaks appear.

There are four types of software sneaks:

- 1) Sneak Output -- the occurrence of an undesired output
- 2) Sneak Inhibit -- the undesired inhibition of an output
- 3) Sneak Timing -- the occurrence of an undesired output by virtue of its timing or mismatched input timing
- 4) Sneak Message -- the program message does not adequately reflect the condition.

There are five steps to an SSA [PEYTON]:

- 1) gather all information pertaining to the system (e.g., source code, system requirements and specifications)
- 2) build a data base that shows every place a variable is used, every subroutine call, pertinent data characteristics, and subroutine name synonyms
- 3) generate network trees -- network trees are pictorial representations of program logic using electrical symbols
- 4) analyze the data using the network trees, clue lists (pointers to problems which may occur in certain shapes (flow of control), variable usages (data flows), or types of applications, and reference documents to find possible problems
- 5) formally document the results of the SSA including resolutions of problems.

Advantages of SSA include, but are not limited to:

- A feasibility study conducted in 1975 found that SSA is a viable means of identifying certain classes of software problems; works equally well on different software languages; and, does not require execution of the software to detect problems [GODOY].
- Is most effective in finding errors that are not usually detected by desk checking or standard verification and validation techniques [PEYTON].

Disadvantages of SSA include, but are not limited to:

- Depends on the experience and skill of the analyst [HECHT].
- Is actually more of a reliability than safety technique, and it is unlikely that many serious faults will be found using this analysis [LEVESON86].
- The results of the SSA depends on the availability and quality of documentation on the system [PEYTON].

3.9. Additional References for Software Hazard Analysis Techniques

The preceding subsections provide general descriptions of several software hazard analysis techniques. More specific details can be found in the references cited in each subsection, and in the additional references list in Table 3-2. Some of the references in Table 3-3 may actually describe the technique's application to hardware.

Table 3-2. Additional Techniques' References

| TECHNIQUE | REFERENCE |
|--|--|
| Code Walk-Throughs | ■ Dunn, Robert, <u>Software Defect Removal</u> , McGraw-Hill, Inc., 1984. |
| Event Tree Analysis | ■ Limnious, N. and J.P. Jeannette, "Event Trees and their Treatment on PC Computers," <i>Reliability Engineering</i> , Vol. 18, No. 3, 1987. |
| Hazard and Operability Studies | <ul style="list-style-type: none"> ■ "A Guide to Hazard and Operability Studies," Chemical Industry Safety and Health Council of the Chemical Industries Association, Alembic House, London, UK. ■ Kletz, T.A., "HAZOP and HAZAN", Institution of Chemical Engineers, UK, 1986. |
| Nuclear Safety Cross-Check Analysis | ■ Air Force Regulation 122-4, "Nuclear Surety Design Certification for Nuclear Weapon System Software and Firmware," Department of the Air Force, 24 August 1987. |
| Petri Nets | ■ Peterson, J.L., <u>Petri Net Theory and Modelling of Systems</u> , Prentice Hall, 1981. |
| Software Failure Mode, Effects, and Criticality Analysis | ■ MIL-STD-1629A, "Procedures for Performing A Failure Mode and Effect Analysis", Department of Defense, 24 Nov 1980. |
| Software Fault Tree Analysis | <ul style="list-style-type: none"> ■ Fussel, J., "Fault Tree Analysis - Concepts and Techniques," <u>Generic Techniques in Reliability Assessment</u>, Noordhoff Publishing Co., Leyden, Holland, 1976. ■ NUREG-0942, "Fault Tree Handbook," U.S. Nuclear Regulatory Commission, 1981. |
| Software Sneak Analysis | <ul style="list-style-type: none"> ■ Rankin, J.P., "Sneak Circuit Analysis," <i>Nuclear Safety</i>, Vol. 14, No. 5, 1973. ■ Hecht, Herbert and Myron Hecht, "Computer Resources Handbook for Flight Critical Systems" [HECHT] |

4. SOFTWARE QUALITY

System hazard analyses identify operational hazards which traditionally have been associated with natural disasters (e.g., earthquakes and fire) and equipment problems (e.g., electrical failure and hardware component failure). System hazard analyses also identify hazards which may result from how the system is developed. And, while these analyses examine software's impact on the system and its development process, neither system nor software hazard analyses address how the software was developed. The software must be developed correctly and must have provisions to mitigate consequences of its failure. If the software is not developed correctly, then the development process itself is a potential hazard. The techniques in section 3 principally focus on whether or not the software related to potential hazards has been examined, but few focus on verifying or proving the QUALITY of the software.

NIST has prepared a framework for the development and assurance of high integrity software [NIST223] which recommends software development and software assurance processes for producing quality software. Figure 4-1 depicts the software development and software assurance processes described in [NIST223]. The remainder of this section summarizes the software processes, includes software engineering practices to be used when developing and assuring high integrity software, and is based on [NIST223].

4.1. Software Development

Software development processes are those processes that are used to construct the software, that is, define the software, design it, implement the design into software code, and integrate the software into the system. Their purpose is to build the software, and make corrections as needed. Each software development process produces outputs which ultimately lead to the final software product which is integrated with other system components and is executed in the installed system.

The development of high integrity software includes the software requirements process, software design process, code process, software integration process, software installation process, and software operation and maintenance process. These processes are briefly outlined below; for more detailed information see [NIST223].

The objectives of the software requirements process are to fulfill the system and software objectives, develop software requirements based on, and traceable back to, the system requirements, and to provide complete, consistent, correct, testable, and understandable information from which the software may be designed. This process uses the system requirements, system design, the initial project management plan (PMP), and software requirements standards identified in the software quality assurance plan (SQAP), to develop the requirements for software. The software requirements encompass functional, performance, interface, safety, security, and quality requirements [NIST180]. The software requirements process produces a software requirements specification (SRS). A user's manual is also started, but not completed, during this process.

The objectives of the software design process are to develop the software design based on, and traceable back to, the software requirements, and to provide complete, consistent, correct,

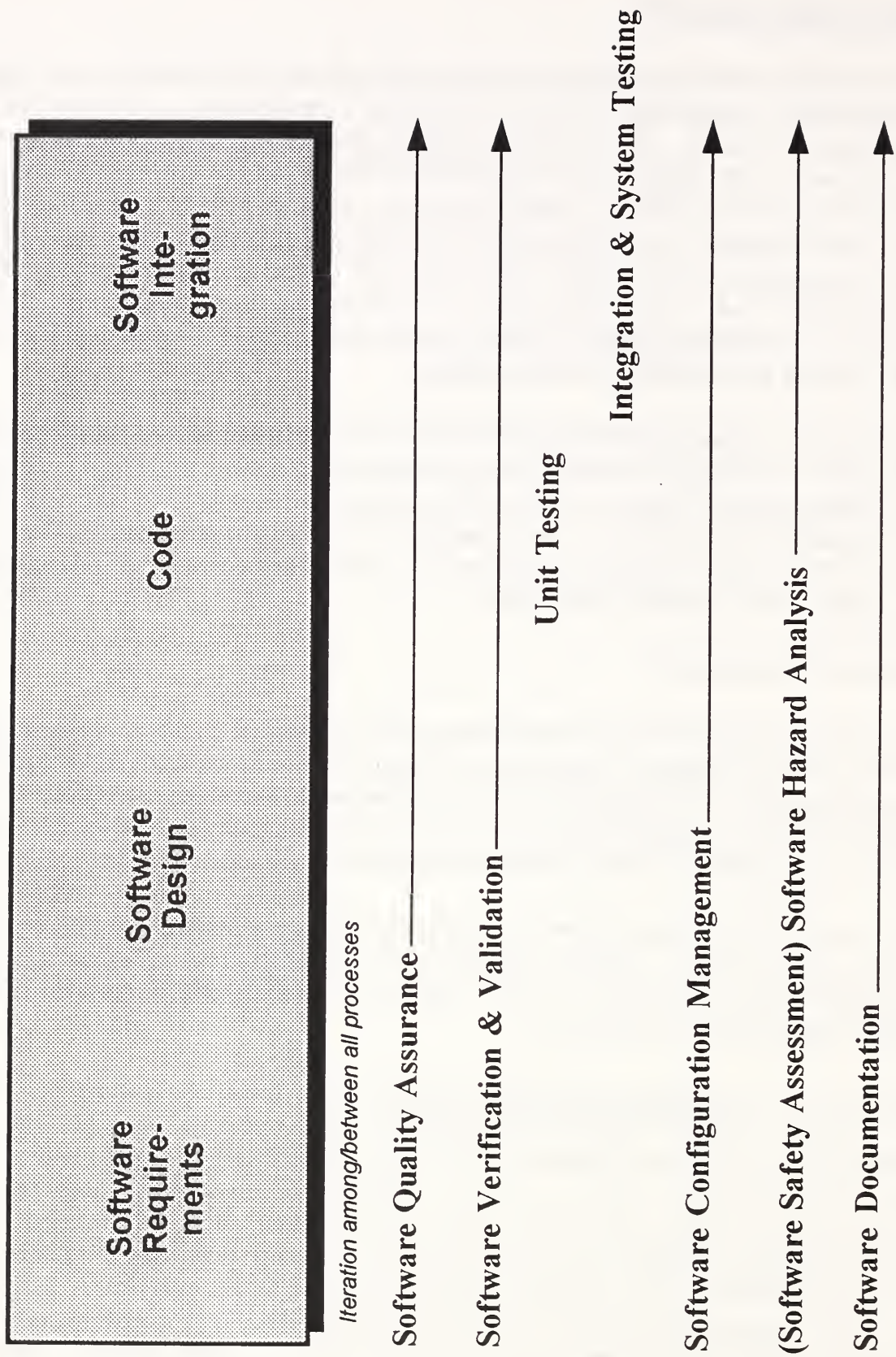


Figure 4-1 Software Development & Assurance Processes.

testable, and understandable information from which the software code may be generated. This process uses the SRS, the initial PMP, and software design standards identified in the SQAP to develop the software design. System documentation is available for reference. The software design process produces a software design description (SDD), a database design description (DBDD), and possibly a revised SRS and/or updated PMP.

The objective of the code process is to develop the source code based on, and traceable back to, the software design and software requirements. This process uses the SRS, SDD, DBDD, the PMP, and code standards identified in the SQAP, to develop the code. The code process produces the source code, a source code manual, and supporting documentation for source code. While the code process and unit test process are often associated with each other, the unit test process is a software assurance process and is described in section 4.2.

The objectives of the software integration process are to produce executable code, and to integrate the executable code into other software or hardware components. This process uses the source code to integrate software components with other software components and with the hardware in preparation for installation into the system. The software integration process produces the executable code and a software installation plan. A software maintenance manual is started, and a user's manual is completed during this process. The software integration process occurs also in accordance with the overall system integration and test plan which may mean several iterations of this software integration process until all software components have been integrated with other system components.

The objective of the software installation process is to install the software at each site, and to determine whether the software will perform as required at all the sites in which it will operate. The software installation process produces a software installation report, and a software maintenance manual is completed.

The objective of the software operation and maintenance process is to ensure that the software meets its requirements throughout its operation and when modifications are made to it. This process uses the integrated software, software documentation, and software operation and maintenance standards to monitor the software throughout its operation, and modify the software as necessary (e.g., for error correction, enhancements, changes to operating environment) for every site at which the software is installed. Essentially, this process will repeat groups of the preceding processes. The software operation and maintenance process produces a software operational procedures manual (if additional information is needed beyond the user's manual), and supporting documentation for modifications of the software (e.g., anomaly reports, modification feasibility reports).

4.2. Software Assurance

Software assurance processes plan and manage the software development processes, and some, like the project management and software quality assurance processes, also oversee other software assurance processes. Their purpose is to provide assurance that the software will meet its requirements and consequently support the system requirements. Software assurance processes check and analyze the decisions regarding the software and its relationship to the system, the plans and their implementations, and they analyze and test the software outputs.

In addition to the software hazard analysis process, the assurance of high integrity software includes the project management process, software quality assurance process, software verification and validation (includes test) process, and software configuration management process. These processes are briefly outlined below; for more detailed information see [NIST223].

The objective of the project management (PM) process is to establish the organizational structure of the project and assign responsibilities. This process uses the system requirements documentation and information about the purpose of the software; criticality of the software; required deliverables; and, available time and resources, to plan and manage the software development and software assurance processes. The PM process overlaps and often reiterates other software assurance processes. It establishes or approves standards, monitoring and reporting practices, high-level policy for quality (process improvement and output quality), and cites regulations. The PM process produces a project management plan (PMP) which includes references to all other software assurance documentation.

The objectives of the software quality assurance (SQA) process are to ensure that the software development and software assurance processes comply with software assurance plans and standards, and to recommend process improvement. This process uses the system requirements, and information about the purpose and criticality of the software to evaluate the outputs of the software development and software assurance processes. A software quality assurance plan (SQAP) and review and audit reports are produced during the SQA process.

The objective of the software verification and validation (SV&V) process is to comprehensively analyze and test the software concurrently with processes of software development and software maintenance to determine that the software performs its intended functions correctly, ensure that it performs no unintended functions, and measure its quality and reliability [NIST165]. SV&V is a detailed engineering assessment for evaluating how well the software is meeting its technical requirements, and in particular its safety, security and reliability objectives and for ensuring that software requirements are not in conflict with any standards or requirements applicable to other system components. There are SV&V activities to analyze, review, demonstrate or test the outputs of every software development and maintenance process; these SV&V activities may directly impact software development processes.

The objectives of the software configuration management (SCM) process are to track the different versions of the software, and ensure that each version of the software contains the exact software outputs generated and approved for that version. SCM is responsible for ensuring that any changes to any software outputs during the development processes are made in a controlled and complete manner. The SCM process produces a software configuration management plan (SCMP).

4.3 Software Engineering Practices

Software engineering practices are those techniques recommended either to prevent errors from being entered into the software during development, or are properties to be built into high integrity software [NIST204]. Some software engineering practices that may enhance the quality of the software are described below.

Formal methods may be used to specify/model the requirements mathematically. A recent study supports the concept that formal methods may eliminate ambiguity in the requirements but cannot ensure completeness. The report suggests that better methods of technology transfer and better automated support are needed before formal methods can be widely used [NIST626]. And, [FUJII] contains a methodology for describing software specifications in English. The use of either formal methods or the [FUJII] approach requires analyzing the completeness and meaning of each requirement. However, one example in [FUJII] demonstrates that neither method can eliminate all ambiguity. And, neither method can prove the completeness of the total set of requirements. Formal methods can also be used for verifying the requirements and for design proof of correctness.

Prototyping, simulation, and modeling can be used in developing software requirements, and in the software design process. Rapid prototyping and simulation analysis are also useful in the verification and validation of the software requirements, software design, and code.

The way in which the software is designed contributes greatly to its quality. Component isolation separates safety critical components from other components, making analysis of, and changes to, these components easier to accomplish. Modularity ensures that changes to one component minimally affect other components. Information hiding prevents components' actions from interfering with other components. Redundancy is used to prevent or recover from failures. And, interaction with the operator or user of the software system during the design of the software/human interface can also be helpful.

Using a software design methodology that is well suited to the software application is important. Today, new technology is forcing a second look at design methods, specifically object-oriented design (OOD). NIST conducted a study of the attributes of OOD relative to safety-critical software for the United States Nuclear Regulatory Commission (NRC). The purpose was to describe attributes of OOD (e.g., classes, encapsulation, inheritance) relative to their capability for supporting features desired in software for safety systems (e.g., modularity, functional diversity, traceability, and non-ambiguity). The results were presented at the NRC/NIST workshop in September 1993 and published in the workshop proceedings [NIST216].

The use of high-level languages has also been recommended for quality software [NIST204]. Using high level, standard languages and their standards lessens programming errors. Eliminating programming practices that have been demonstrated to be problematic (e.g., floating point arithmetic, use of interrupts) simplifies analyzing system behavior. It is also important to use a language with a thoroughly tested compiler.

There are also software engineering practices that apply to the software assurance processes. Use of cost-modeling and risk assessment techniques can aid the project management process. Inspections, reviews, and audits can be applied to all software processes under the software quality assurance process. Software error, measurement, statistical, algorithm, database, technical, control and data flow, and timing and sizing analysis techniques are useful in the software verification and validation process. Test strategies such as equivalence partitioning, cause-effect, boundary value, stress, event directed, data flow, logic flow, performance, timing, sizing and random, top-down, bottom-up, sandwich, statistical testing, functional testing, performance testing, when applied appropriately contribute to the quality of the software. And, the use of

selected software hazard analysis techniques (e.g., software fault tree analysis, petri nets) can aid this process.

5. CONCLUSIONS

Although there has been increased attention in recent years to the contribution software makes to total system safety, software safety is still not addressed to the extent necessary. Few standards specify software safety procedures, much less software hazard analysis. And those standards that do address such analyses are often incomplete. There is also a lack of consensus among the standards' makers regarding terminology, what is included in a software safety analysis, and when and how it is performed. In order to ensure complete system safety, software safety has to be seriously considered.

The phrase "software *hazard* analysis" seems to be disappearing from standards. Standards either describe software *safety* analysis as a process which may or may not include software hazard analysis, or incorporate software safety or hazard analysis into the system level analyses. While software safety must be considered as part of overall system safety [LEVESON87], it is important to ensure that software safety is adequately addressed, and it is often not. For example, [MIL882B] contained a separate section for software safety. In the following version, [MIL882C], the software safety analysis is included within the sections addressing system safety, and some analyses, like the code level hazard analysis, were omitted.

Some of the techniques recommended by standards to conduct software hazard analysis are dated and/or are based on traditional hardware hazard analysis techniques. These techniques often were developed prior to the appearance of current software engineering methods and tools which perform the same functions. Standards' makers need to explore the analysis capabilities of current software engineering methodologies and CASE tools.

Of the techniques investigated for this report, only nuclear safety cross check analysis, software fault tree analysis, and software sneak analysis have been used specifically for software hazard analysis. And, while these techniques originated from similar techniques for hardware, these techniques are relatively young and untried for software. More investigation and experimentation is needed to determine the usefulness, scope and cost effectiveness of these techniques.

While this study cannot wholly evaluate the techniques against the criteria listed in section 3 from [MOD0056'91], the following comments are provided. Most of the techniques examined for this study do "enhance the understanding of the way risks arise, are prevented, or reduced; permit the modelling and evaluation of failure modes; and, enable systematic analysis to be carried out in a manner that is auditable, repeatable, and verifiable." However, most of these techniques are not currently automated by tools for analyzing software. In fact only Petri nets are supported by commercially available tools. And, again for examining software, there are few "documented examples of [a technique's] successful application." And when documentation does exist, it does not provide resource information (e.g., cost, schedule).

As software is included in more and more critical systems (e.g., nuclear power plants, medical devices and transportation systems) the need for software safety programs becomes crucial. These software safety programs should consist of not only software safety analyses, but methodologies that assist in the assurance of developing quality software.

In summary, the following issues need to be addressed regarding the safety of software:

- Standards need to require software safety programs that include software hazard analysis and other safety analyses, and software development and software assurance processes.
- Techniques for safety analysis need to be updated, and the use of software engineering methodologies and computer-aided software engineering (CASE) tools needs to be considered for software safety analysis.

6. REFERENCES

[AFISC]

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

[ANS501]

ANSI/ANS-50.1, "(DRAFT #6) Nuclear Safety Design Criteria for Light Water Reactors," American Nuclear Society, January 1993.

[CARLSEN]

Carlsen, Dr. Kjell, Brian C. Nielsen, and C. Andy Hailey, "SNEAK ANALYSIS - Boeing's Electrical Systems Engineering Quality Program Applied to the Automotive Industry," Presented at the University of Michigan Quality Assurance Seminar, Traverse City, Michigan, August 1989.

[FDA89]

"(DRAFT) Reviewer Guidance for Computer-Controlled Devices," Medical Device Industry Computer Software Committee, Food and Drug Administration, January 1989.

[FDA91]

"Reviewer Guidance for Computer-Controlled Medical Devices Undergoing 510(k) Review," Office of Device Evaluation, Center for Devices and Radiological Health, Food and Drug Administration, 1991.

[FUJII]

Fujii, R., "Software Engineering for Instrumentation and Control Systems," Nuclear Plant Instrumentation, Control, and Man-Machine Technologies, Oak Ridge, TN, April 19-21, 1993.

[GODOY]

Godoy, S.G. and G.J. Engels, "Sneak Circuit and Software Sneak Analysis," *Journal of Aircraft*, Vol. 15, No. 8, August 1978.

[HALL]

Hall, Fred M., Raymond A. Paul, Wendy E. Snow, "Hardware/Software FMECA," 1983 Proceedings Annual Reliability and Maintainability Symposium, 1983.

[HANSEN]

Hansen, Mark D., "Survey of Available Software-Safety Analysis Techniques," 1989 Proceedings--Annual Reliability and Maintainability Symposium, The Institute of Electrical and Electronics Engineers, 1989.

[HECHT]

Hecht, Herbert and Myron Hecht, ASD-TR-85-5020, "Computer Resources Handbook for Flight Critical Systems," USAF Aeronautical Systems Division (Wright-Patterson AFB), January 1985.

[IECWG9'89]

IEC/TC65A WG9, IEC 65A(Secretariat)94, "89/33006 DC - (DRAFT) Software for Computers in the Application of Industrial Safety-Related Systems," British Standards Institution, November 1989.

[IECWG9'91]

IEC/TC65A WG9, IEC 65A(Secretariat)122, "Software for Computers in the Application of Industrial Safety-Related Systems," Version 1.0, British Standards Institution, 26th September 1991.

[IECWG10'89]

IEC/TC65A WG10, "89/33005 DC - (DRAFT) Functional Safety of Programmable Electronic Systems," British Standards Institution, November 1989.

[IEEE610]

IEEE Std 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology," The Institute of Electrical and Electronics Engineers, Inc., February 1991.

[IEEE1074]

ANSI/IEEE Std 1074-1991, "IEEE Standard for Developing Software Lifecycle Processes," The Institute of Electrical and Electronics Engineers, Inc., 1991.

[IEEEP1059]

IEEE Guide P1059 (DRAFT 7), "Guide to Software Verification and Validation Plans," IEEE Standards Department, October 18, 1992.

[IEEEP1228-C]

P1228, "(DRAFT C) Draft Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, November 13, 1990.

[IEEEP1228-D]

P1228, "(DRAFT D) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., March 6, 1991.

[IEEEP1228-E]

P1228, "(DRAFT E) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., July 19, 1991.

[JPL93]

JPL D-10058, "Software Systems Safety Handbook," Prepared by Jet Propulsion Laboratory for National Aeronautics and Space Administration, May 10, 1993,

[LEVESON83]

Leveson, Nancy G. and Peter R. Harvey, "Analyzing Software Safety," *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 5, The Institute of Electrical and Electronics Engineers, September 1983.

[LEVESON86]

Leveson, N.G., "Software Safety: Why, What, and How," *Computing Surveys*, Vol. 18, No. 2, Association for Computing Machinery, June 1986.

[LEVESON87]

Leveson, Nancy G., Janice L. Stolzy, "Safety Analysis Using Petri Nets," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 3, The Institute of Electrical and Electronics Engineers, March 1987.

[LEVESON89]

Leveson, Nancy, "Software Safety," Presentation to IEEE Software Safety Working Group, October 1989.

[LEVESON91]

Leveson, Nancy G., Stephen S. Cha and Timothy J. Shimeall, "Safety Verification of Ada Programs Using Software Fault Trees," *IEEE Software*, The Institute of Electrical and Electronics Engineers, July 1991.

[LEVESON92]

Leveson, Nancy G. and Clark S. Turner, "An Investigation of the Therac-25 Accidents," University of California, Irvine, CA, November 1992.

[LEVINSON]

Levinson, Stanley H. and H. Tazewell Daughtrey, "Risk Analysis of Software-Dependent Systems," Presented at Probabilistic Safety Assessment International Topical Meeting, Clearwater Beach, FL, January 1993.

[MAZUR]

Mazur, Mojmir F., "SOFTWARE FMECA; Failure Mode, Effect and Criticality Analysis; U.S. Patent and Trademark Office Pilot Project," 1994.

[MIL882B]

MIL-STD-882B, "System Safety Program Requirements," Department of Defense, 30 March 1984.

[MIL882C]

MIL-STD-882C, "Systems Safety Program Requirements," Department of Defense.

[MOD0055'89]

Interim Defence Standard 00-55, "(DRAFT) Requirements for the Procurement of Safety Critical Software in Defence Equipment," Ministry of Defence, UK, May 1989.

[MOD0056'89]

Interim Defence Standard 00-56, "(DRAFT) Requirements for the Analysis of Safety Critical Hazards," Ministry of Defence, UK, May 1989.

[MOD0056'91]

Interim Defence Standard 00-56, "Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment," Ministry of Defence, UK, 5 April 1991.

[NIST190]

NIST Special Publication 500-190, Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991, U.S. Department of Commerce, National Institute of Standards and Technology, August 1991.

[NIST204]

NIST Special Publication 500-204, "High Integrity Software Standards and Guidelines," U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, September 1992.

[NIST209]

NIST Special Publication 500-209, "Software Error Analysis," U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, April 1993.

[NIST216]

NIST Special Publication 500-216, *Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop, September 13-14, 1993, Rockville Crowne Plaza Hotel, Rockville, Maryland*, U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, January 1994.

[NIST626]

NIST GCR 93/626, "An International Survey of Industrial Applications of Formal Methods Volume 1 Purpose, Approach, Analysis, and Conclusions," U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, March 1993.

[NIST4909]

NISTIR 4909, "Software Quality Assurance: Documentation and Reviews," U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, September 1992.

[NIST223]

NIST SP 500-223, "A Framework for the Development and Assurance of High Integrity Software," U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, January 1995.

[NSS1740]

NSS 1740.13, "(Interim) NASA Software Safety Standard," National Aeronautics and Space Administration, June 1994.

[PES87]

"Programmable Electronic Systems in Safety Related Applications," Parts 1 and 2, Health and Safety Executive, United Kingdom, 1987.

[PEYTON]

Peyton, B.H. and D.C. Hess, "Software Sneak Analysis," IEEE Seventh Annual Conference of the Engineering in Medicine and Biology Society, The Institute of Electrical and Electronics Engineers, 1985.

[POTOCKI]

Potocki de Montalk, J.P., "Computer Software in Civil Aircraft," *Microprocessors and Microsystems*, Volume 17, Number 1, 1993.

[RAHEJA89]

Raheja, D. and G. Raheja, "Maintainability Analysis for Intelligent Controls," Proceedings IEEE International Symposium on Intelligent Control 1988, IEEE Computing Society Press, 1989.

[RAHEJA91]

Raheja, Dev G., "Assurance Technologies - Principles and Practices," McGraw-Hill, Inc., 1991.

[SOFTENG]

"Standard for Software Engineering of Safety Critical Software," Rev. 0, Ontario Hydro, December 1990.

[TYSZER]

Tyszer, J., P. Parent, J. Rajski and V. K. Agarwal, "The Hierarchical Description of Stochastic Petri Nets," Department of Electrical Engineering, McGill University.

[WALLACE]

Wallace, D.R., D.R. Kuhn, J.C. Cherniavsky, "Report on a Workshop on the Assurance of High Integrity Software," *Proceedings of the Sixth Annual Conference on Computer Assurance (COMPASS '91)*, NIST, Gaithersburg, MD, June 24-27, 1991, The Institute of Electrical and Electronics Engineers, 1991.

APPENDIX A. BIBLIOGRAPHY OF HIGH INTEGRITY SOFTWARE DOCUMENTS

A.1. Standards and Guidelines

AF800-5

AFSC/AFLCP 800-5, "(DRAFT) Software Independent Verification and Validation (IV&V)," Air Force Systems Command and Air Force Logistics Command, 1988.

AF800-45

AF PAMPHLET 800-45, "Software Independent Verification and Validation (IV&V)," Department of the Air Force, 1 May 1991.

AFISC

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

ANS103

ANSI/ANS-10.3-199x, (DRAFT 5), "Documentation of Computer Software," American Nuclear Society, 3/7/92.

ANS104

ANSI/ANS-10.4-1987, "Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry," American Nuclear Society, May 13, 1987.

ANS501

ANSI/ANS-50.1, "(DRAFT #6) Nuclear Safety Design Criteria for Light Water Reactors," American Nuclear Society, January 1993.

ANS7432

ANSI/IEEE-ANS-7-4.3.2-1982, "Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations," American Nuclear Society, 1982. AND ANSI/IEEE-ANS-7-4.3.2-19XX, Draft 2, as of November, 1991.

ANSP7432

P-7-4.3.2, draft 7, "American National Standard - Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," Sponsor: Nuclear Power Engineering Committee of the IEEE Power Engineering Society.

ANSIX99

ANSI X9.9-1986, "Financial Institution Message Authentication (Wholesale)," X9 Secretariat, American Bankers Association, August 15, 1986.

ANSIX917

ANSI X9.17-1985, "Financial Institution Key Management (Wholesale)," X9 Secretariat, American Bankers Association, April 4, 1985.

AQAP13

AQAP-13, "NATO Software Quality Control System Requirements," NATO, August 1991.

ASMENQA1

ASME NQA-1-1989, "Quality Assurance Program Requirements for Nuclear Facilities," The American Society of Mechanical Engineers, September 15, 1989.

ASMENQA2

ASME NQA-2a-1990, "Quality Assurance Requirements for Nuclear Facility Applications," The American Society of Mechanical Engineers, November 1990.

ASMENQA3

ASME NQA-3-1989, "Quality Assurance Program Requirements for the Collection of Scientific and Technical Information for Site Characterization of High-Level Nuclear Waste Repositories," The American Society of Mechanical Engineers, March 23, 1990.

ASMESUPP

Supplement 17S-1, ASME NQA-1-1989, "Supplementary Requirements for Quality Assurance Records," The American Society of Mechanical Engineers.

ASQCA3

ANSI/ASQC A3-1987, "Quality Systems Terminology," American Society of Quality Control, 1987.

BOEING

"(DRAFT) BA&E (Boeing Aerospace and Electronics) System Safety Instruction - System Safety Engineering in Software Development," The Boeing Company, 11/11/89.

BSI89

"89/97714-Guide to the Assessment of Reliability of Systems Containing Software," British Standards Institution, 12 September 1989.

CATEGORY

"Guideline for the Categorization of Software in Ontario Hydro's Nuclear Facilities with respect to Nuclear Safety," Revision 0, Nuclear Safety Department, June 1991.

CENSUS

"Programming Standards and Guidelines Manual," Bureau of the Census, March 27, 1991.

CSA89

CAN/CSA-Q396.1.2-89, "Quality Assurance Program for Previously Developed Software Used in Critical Applications," Canadian Standards Association, January 1989.

CSC003

CSC-STD-003-85, "Computer Security Requirements--Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments," Department of Defense, 25 June 1985.

DLP880

DLP880, "(DRAFT) Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations (based on IEC Standard 880)," David L. Parnas, Queen's University, Kingston, Ontario, March, 1991.

DOD2167A

DOD-STD-2167A, "Defense System Software Development," Department of Defense, 29 February 1988.

DOT86

"Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Devices for Federal E.F.T. Use," United States Department of the Treasury, September 1, 1986.

ESA

ESA PSS-05-10, Issue 1, "Guide to Software Verification and Validation," European Space Agency, February 1994. with **ESA Guide to the Software Engineering Standards**

FAA026

FAA-STD-026, "National Airspace System (NAS) Software Development," U.S. Department of Transportation, Federal Aviation Administration, March 31, 1989.

FDA89

"(DRAFT) Reviewer Guidance for Computer-Controlled Devices," Medical Device Industry Computer Software Committee, January 1989.

FDA91

"Reviewer Guidance for Computer-Controlled Medical Devices Undergoing 510(k) Review," Office of Device Evaluation, Center for Devices and Radiological Health, Food and Drug Administration.

FIPS74

FIPS PUB 74, "Guidelines for Implementing and Using the NBS Data Encryption Standard," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1981 April 1.

FIPS81

FIPS PUB 81, "DES Modes of Operation," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1980 December 2.

FIPS101

FIPS PUB 101, "Guideline for Lifecycle Validation, Verification, and Testing of Computer Software," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1983 June 6.

FIPS106

FIPS 106, "Guideline on Software Maintenance," U. S. Department of Commerce/National Bureau of Standards (U.S.), 1984 June 15.

FIPS132

FIPS PUB 132, "Guideline for Software Verification and Validation Plans," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1987 November 19.¹⁹

FIPS140

FIPS PUB 140 FS 1027, "General Security Requirements for Equipment Using the Data Encryption Standard," General Services Administration, April 14, 1982.

FIPS461

FIPS 46-1, "Data Encryption Standard," U.S. Department of Commerce/National Bureau of Standards (U.S.), 1988 January 22.

FIPS1401

FIPS 140-1, "Security Requirements for Cryptographic Modules," U.S. Department of Commerce/National Institute of Standards and Technology, 1990 May 2.

IEC880

IEC 880, "Software for Computers in the Safety Systems of Nuclear Power Stations," International Electrotechnical Commission, 1986.

IEC9126

ISO/IEC 9126, "Information Technology--Software Product Evaluation--Quality Characteristics and Guidelines for their Use," International Electrotechnical Commission, 1991-12-15.

IECSUPP

45A/WG-A3(Secretary)42, "(DRAFT) Software for Computers Important to Safety for Nuclear Power Plants as a Supplement to IEC Publication 880," International Electrotechnical Commission Technical Committee: Nuclear Instrumentation, Subcommittee 45A: Reactor Instrumentation, Working Group A3: Data Transmission and Processing Systems, May 1991.

¹⁹See IEEE1012.

IECSUPP-94

45A/WG-A3(Secretary)48, "(DRAFT) Nuclear Power Plants - Instrumentation and Control Systems Important to Safety - First Supplement to IEC Publication IEC 880," IEC SC45A, May 1994.

IECTC56

IEC/TC56, "89/97714 - (DRAFT) Guide to the Assessment of Reliability of Systems Containing Software," British Standards Institution, 12 September 1989.

IECWG9'89

IEC/TC65A WG9, IEC 65A(Secretariat)94, "89/33006 DC - (DRAFT) Software for Computers in the Application of Industrial Safety-Related Systems," British Standards Institution, November 1989.

IECWG9'91

IEC/TC65A WG9, IEC 65A(Secretariat)122, "Software for Computers in the Application of Industrial Safety-Related Systems," Version 1.0, 26th September 1991.

IECWG10'89

IEC/TC65A WG10, "89/33005 DC - (DRAFT) Functional Safety of Programmable Electronic Systems," British Standards Institution, November 1989.

IECWG10'92

IEC/TC65A WG10, "(DRAFT) Functional Safety of Electrical/Electronic/Programmable Electronic Systems," 1992.

IECWG10'93

IEC/TC65A WG10, "(DRAFT) Functional Safety: Safety Related Systems

IEEE603

IEEE Std 603-1980, "IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations," The Institute of Electrical and Electronics Engineers, Inc., November 24, 1980.

IEEE610

ANSI/IEEE Std 610.12-1990, "Glossary of Software Engineering Terminology," The Institute of Electrical and Electronics Engineers, Inc., February, 1991.

IEEE7301

ANSI/IEEE Std 730.1-1989, "IEEE Standard for Software Quality Assurance Plans," Institute of Electrical and Electronics Engineers, Inc., October 10, 1989.

IEEE730

ANSI/IEEE Std 730-1989, "IEEE Standard for Software Quality Assurance Plans," Institute of Electrical and Electronics Engineers, Inc., January 22, 1990.

IEEE828

ANSI/IEEE Std 828-1990, "IEEE Standard for Software Configuration Management Plans," Institute of Electrical and Electronics Engineers, Inc., February 15, 1991.

IEEE829

ANSI/IEEE Std 829-1983, "IEEE Standard for Software Test Documentation," Institute of Electrical and Electronics Engineers, Inc., August 19, 1983.

IEEE830-84

ANSI/IEEE Std 830-1984, "IEEE Guide to Software Requirements Specifications," Institute of Electrical and Electronics Engineers, Inc., July 29, 1984.

IEEE830-93

ANSI/IEEE Std 830, "(DRAFT) IEEE Recommended Practice for Software Requirements Specifications," Institute of Electrical and Electronics Engineers, Inc., 8/10/93.

IEEE982-1

ANSI/IEEE Std 982.1-1988, "IEEE Standard Dictionary of Measures to Produce Reliable Software," Institute of Electrical and Electronics Engineers, Inc., August 10, 1989.

IEEE982-2

ANSI/IEEE Std 982.2-1988, "IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software," Institute of Electrical and Electronics Engineers, Inc., August 10, 1989.

IEEE983

ANSI/IEEE Std 983-1986, "IEEE Guide for Software Quality Assurance Planning," Institute of Electrical and Electronics Engineers, Inc., February 20, 1986.

IEEE990

ANSI/IEEE Std 990-1987, "IEEE Recommended Practice for Ada As a Program Design Language," Institute of Electrical and Electronics Engineers, Inc., October 1, 1987.

IEEE1002

ANSI/IEEE Std 1002-1987, "IEEE Standard Taxonomy for Software Engineering Standards," Institute of Electrical and Electronics Engineers, Inc., June 4, 1987.

IEEE1008

ANSI/IEEE Std 1008-1987, "IEEE Standard for Software Unit Testing," Institute of Electrical and Electronics Engineers, Inc., July 28, 1986.

IEEE1012

ANSI/IEEE Std 1012-1986, "IEEE Standard for Software Verification and Validation Plans," The Institute of Electrical and Electronics Engineers, Inc., February 10, 1987.²⁰

²⁰Adopted by the Federal government as FIPS PUB 132.

IEEE1016

ANSI/IEEE Std 1016-1987, "IEEE Recommended Practice for Software Design Descriptions," Institute of Electrical and Electronics Engineers, Inc., October 6, 1987.

IEEE1028

ANSI/IEEE Std 1028-1988, "IEEE Standard for Software Reviews and Audits," Institute of Electrical and Electronics Engineers, Inc., June 29, 1989.

IEEE1042

ANSI/IEEE Std 1042-1987, "IEEE Guide to Software Configuration Management," Institute of Electrical and Electronics Engineers, Inc., March 10, 1988.

IEEE1058

ANSI/IEEE Std 1058-1987, "IEEE Standard for Software Project Management Plans," Institute of Electrical and Electronics Engineers, Inc., October 6, 1988.

IEEE1074

ANSI/IEEE Std 1074-1991, "IEEE Standard for Developing Software Lifecycle Processes," The Institute of Electrical and Electronics Engineers, Inc., 1991.

IEEE7432

ANSI/IEEE Std 7432-1993, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," The Institute of Electrical and Electronics Engineers, Inc., 1993.

IEEE1063

ANSI/IEEE Std 1063-1987, "IEEE Standard for Software User Documentation," Institute of Electrical and Electronics Engineers, Inc., February 2, 1989.

IEEE1228

IEEE Std 1228-1994, "IEEE Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, August 9, 1994.

IEEEP1059

IEEE Std P1059-199X, "(DRAFT 7.1) IEEE Guide for Software Verification and Validation Plans," Institute of Electrical and Electronics Engineers, May 24, 1993.

IEEEP1228-C

P1228, "(DRAFT C) Draft Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, November 13, 1990.

IEEEP1228-D

P1228, "(DRAFT D) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., March 6, 1991.

IEEEP1228-E

P1228, "(DRAFT E) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., July 19, 1991.

IEEEP1228-G

P1228, "(DRAFT G) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., 1/14/92.

IEEEP1228-H

P1228, "(DRAFT H) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., 4/27/92.

IEEEP1228-J

P1228, "(DRAFT J) Standard for Software Safety Plans," Institute of Electrical and Electronics Engineers, Inc., 2/11/93.

IEEEGUIDE

"Guide to Software Design Descriptions," Institute of Electrical and Electronics Engineers, 1993.

IEEETEST

"(DRAFT) Guidelines for Assuring Testability," The Institution of Electrical Engineers, May 1987.

IFIP104

IFIP WG 10.4, "Dependability: Basic Concepts and Terminology," IFIP Working Group on Dependable Computing and Fault Tolerance, October 1990.

ISASP84

ISA-SP84, Draft 10, "(DRAFT) Programmable Electronic Systems (PES) for use in Safety Applications," Instrument Society of America, August 1992.

ISO9000

ISO 9000, "International Standards for Quality Management," May 1990.

ISO12207

ISO/IEC DIS 12207-1, "(DRAFT) Information Technology--Software--Part 1: Software Life Cycle Process," International Electrotechnical Commission, 1994.

ITSEC89

ITSEC 1.1989, "Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems," GISA - German Information Security Agency, 1989.

ITSEC90

ITSEC 1.1990, "(DRAFT) Information Technology Security Evaluation Criteria (ITSEC)," Harmonised Criteria of France-Germany-the Netherlands-the United Kingdom, 02 May 1990.

JPL93

JPL D-10058, "Software Systems Safety Handbook," PREPARED BY Jet Propulsion Laboratory FOR National Aeronautics and Space Administration, May 10, 1993,

MIL347

MIL-HDBK-347, "Mission-Critical Computer Resources Software Support," Department of Defense, 22 May 90.

MIL498

MIL-STD-498 (DRAFT), "Software Development and Documentation," Department of Defense, 30 November 1994.

MIL882B

MIL-STD-882B, "System Safety Program Requirements," Department of Defense, 30 March 1984.

MIL882C

MIL-STD-882C, "Systems Safety Program Requirements," Department of Defense, DISTRIBUTION STATEMENT A.

MIL1521B

[Proposed Updates to] MIL-STD-1521B, "Technical Reviews and Audits for Systems, Equipments, and Computer Software," Logicon Input to the JLC/CSM, June 16, 1989.

MILSDD

MIL-STD-SDD, "(DRAFT) Software Development and Documentation," Department of Defense, 22 December 1992.

MILSWM

MIL-HDBK-SWM (DRAFT), "Software Measurement Selection and Use," Department of Defense, 14 January 1994.

MOD0055'89

Interim Defence Standard 00-55, "(DRAFT) Requirements for the Procurement of Safety Critical Software in Defence Equipment," Ministry of Defence, UK, May 1989.

MOD0055'91

Interim Defence Standard 00-55, "The Procurement of Safety Critical Software in Defence Equipment," Parts 1 and 2, Ministry of Defence, UK, 5 April 1991.

MOD0056'89

Interim Defence Standard 00-56, "(DRAFT) Requirements for the Analysis of Safety Critical Hazards," Ministry of Defence, UK, May 1989.

MOD0056'91

Interim Defence Standard 00-56, "Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment," Ministry of Defence, UK, 5 April 1991.

NASAMGMT

"Management Plan Documentation Standard and Data Item Descriptions (DID)," NASA, 2/28/89.

NASAPROD

"Product Specification Documentation Standard and Data Item Descriptions (DID)," NASA, 2/28/89.

NCSC005

NCSC-TG-005, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria," National Computer Security Center, 31 July 1987.

NCSC021

NCSC-TG-021, "Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria," National Computer Security Center, April 1991.

NIST165

NIST Special Publication 500-165, "Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards," U.S. Department of Commerce/National Institute of Standards and Technology, September 1989.

NIST190

NIST Special Publication 500-190, "Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991," U.S. Department of Commerce/National Institute of Standards and Technology, August 1991.

NIST204

NIST Special Publication 500-204, "High Integrity Software Standards and Guidelines," U.S. Department of Commerce/National Institute of Standards and Technology, September 1992.

NIST209

NIST Special Publication 500-209, "Software Error Analysis," U.S. Department of Commerce/National Institute of Standards and Technology, April 1993.

NIST213

NIST Special Publication 500-213, "Next Generation Computer Resources: Reference Model for Project Support Environments (Version 2.0)," U.S. Department of Commerce/National Institute of Standards and Technology, November 1993.

NIST216

NIST Special Publication 500-216, "Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop (NUREG/CP 0136)," U.S. Department of Commerce/National Institute of Standards and Technology, March 1994.

NIST626

NIST GCR 93/626, "An International Survey of Industrial Applications of Formal Methods Volume 1 Purpose, Approach, Analysis, and Conclusions," U.S. Department of Commerce/National Institute of Standards and Technology, March 1993.

NIST4909

NISTIR 4909, "Software Quality Assurance: Documentation and Reviews," U.S. Department of Commerce/National Institute of Standards and Technology, September 1992.

NPR6300

NPR-STD-6300, "Management of Scientific, Engineering and Plant Software," Office of New Production Reactors, U.S. Department of Energy, March 1991.

NSA8616

NSA Spec. 86-16, "Security Guidelines for COMSEC Software Development," National Security Agency, 10 July 1986.

NSS1740

NSS 1740.13, "(Interim) NASA Software Safety Standard," National Aeronautics and Space Administration, June 1994.

NSWC8933

NSWC TR 89-33, "Software Systems Safety Design Guidelines and Recommendations," Naval Surface Warfare Center, March 1989.

NUREG6018

NUREG/CR-6018, "Survey and Assessment of Conventional Software Verification and Validation Methods," U.S. Nuclear Regulatory Commission, April 1993.

NUREG6101

NUREG/CR-6101 & UCRL-ID-114839, "Software Reliability and Safety in Nuclear Reactor Protection Systems," U.S. Nuclear Regulatory Commission, June 11, 1993.

PES87

"Programmable Electronic Systems in Safety Related Applications," Parts 1 and 2, Health and Safety Executive, 1987.

RTCA178A

RTCA/DO-178A, "Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, March, 1985.

RTCA178B

RTCA/DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," RTCA, Inc., June 29, 1993.

SAFEIT

"SafeIT," Volumes 1 and 2, Interdepartmental Committee on Software Engineering, June 1990.

SOFTENG

"Standard for Software Engineering of Safety Critical Software," Rev. 0, Ontario Hydro, December 1990.

SOFTENG2

"Software Engineering of Category II Software," Rev. 00, Ontario Hydro, 1993 05.

TCSEC

DOD 5200.28-STD, "Department of Defense Trusted Computer System Evaluation Criteria," Department of Defense, December 1985.

UL1998

UL 1998, "The Proposed First Edition of the Standard for Safety-Related Software," Underwater Laboratories, August 17, 1990.

USEREXP

"User Expectations and Requirements for Software Engineering Standards (Discussion Draft), Software Engineering Standards Long-Range Planning Study Group, November 22, 1991.

WL-1037

WL-TR-1037, "Evaluation and Validation Reference Manual," Version 3.0, Wright Laboratory, Wright-Patterson AFB, Ohio, May 1991.

WL-1038

WL-TR-1038, "Evaluation and Validation Guidebook," Version 3.0, Wright Laboratory, Wright-Patterson AFB, Ohio, May 1991.

A.2 Books

BEIZER

Beizer, Boris, Software Testing Techniques, Van Nostrand Reinhold, New York, 1990.

EWICS1

Redmill, F. J. (ed.), "Dependability of Critical Computer Systems 1," Elsevier Science Publishers LTD, 1988.

EWICS2

Redmill, F. J. (ed.), "Dependability of Critical Computer Systems 2," Elsevier Science Publishers LTD, 1989.

EWICS3

Bishop, P. G. (ed.), "Dependability of Critical Computer Systems 3 - Techniques Directory," Elsevier Science Publishers LTD, 1990.

MUSA1

Musa, J.D., A. Iannino, and K. Okumoto, Software Reliability, Measurement, Prediction, Application, McGraw-Hill, New York, 1987.

RAHEJA91

Raheja, Dev G., "Assurance Technologies - Principles and Practices," McGraw-Hill, Inc., 1991.

WILEY

Encyclopedia of Software Engineering, John Wiley & Sons, Inc., 1994.

A.3 Papers

BELL

Bell, R. and S. Smith, "An Overview of IEC Draft Standard: 'Functional Safety of Programmable Electronic Systems.'"

BUTLER

Butler, R. and G. Finelli, "The Infeasibility of Experimental Quantified Life-Critical Software Reliability," *Proceedings of SIGSOFT'91: Software for Critical Systems*, Association for Computing Machinery, December 1991.

FUJII1

Fujii, Roger U., "Software Engineering For Instrumentation and Control," American Nuclear Society, *Nuclear Plan Instrumentation, Control, and Man-Machine Interface Technologies*, Oak Ridge, TN, April 1993.

HANSEN

Hansen, Mark D., "Survey of Available Software-Safety Analysis Techniques," Annual Reliability and Maintainability Symposium - 1989 Proceedings, 1989.

JOANNOU

Joannou, P.K., J. Harauz, D.R. Tremaine, N. Ichiyen, A.B. Clark, "The Canadian Nuclear Industry's Initiative in Real-Time Software Engineering," Ontario Hydro and AECL CANDU, Ontario, Canada.

JUNK

Junk, William S., "Annotated Bibliography - Software Safety," April 24, 1990.

LEVESON83

Leveson, Nancy G. and Peter R. Harvey, "Analyzing Software Safety," *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 5, September 1983.

LEVESON86

Leveson, N.G., "Software Safety: Why, What, and How," *Computing Surveys*, Vol. 18, No. 2, June 1986.

LEVESON87

Leveson, Nancy G., Janice L. Stolzy, "Safety Analysis Using Petri Nets," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 3, March 1987.

LEVESON89

Leveson, Nancy, "Software Safety," Presentation to IEEE Software Safety Working Group, October 1989.

LEVESON91

Leveson, Nancy G., Stephen S. Cha and Timothy J. Shimeall, "Safety Verification of Ada Programs Using Software Fault Trees," *IEEE Software*, July 1991.

LEVESON92

Leveson, Nancy G. and Clark S. Turner, "An Investigation of the Therac-25 Accidents," University of California, Irvine, CA, November 1992.

LEVINSON

Levinson, Stanley H. and H. Tazewell Daughtrey, "Risk Analysis of Software-Dependent Systems," Probabilistic Safety Assessment International Topical Meeting, Clearwater Beach, FL, January 1993.

MAZUR

Mazur, Mojmir F., "SOFTWARE FMECA; Failure Mode, Effect and Criticality Analysis; U.S. Patent and Trademark Office Pilot Project," 1994.

MUSA2

Musa, J.D., and A.F. Ackerman, "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, May 1989.

PETERSON

Peterson, James L., "Petri Nets," *Computing Surveys*, Vol. 9, No. 3, September 1977.

SESAW91-1

DeWalt, Michael P., "Comparison of FAA DO-178A and DOD-STD-2167A Approaches to Software Certification," Software Engineering Standards Application Workshop sponsored by IEEE Computer Society, San Francisco, May 1991.

SESAW91-2

Sanz, Julio Gonzalez, "Standardization for Safety Software: Current Status and Perspectives," Software Engineering Standards Application Workshop sponsored by IEEE Computer Society, San Francisco, May 1991.

SESAW91-3

Wright, Cynthia L. and Anthony J. Zawilski, "Existing and Emerging Standards for Software Safety," Software Engineering Standards Application Workshop sponsored by IEEE Computer Society, San Francisco, May 1991.

TYSZER

Tyszer, J., P. Parent, J. Rajski and V. K. Agarwal, "The Hierarchical Description of Stochastic Petri Nets," Department of Electrical Engineering, McGill University.

