Software Safety

SEI Curriculum Module SEI-CM-6-1.1 (Preliminary)

July 1987

Nancy G. Leveson
University of California, Irvine

Carnegie Mellon University
Software Engineering Institute

This work was sponsored by the U.S. Department of Defense.
Approved for public release. Distribution unlimited.

Nancy G. Leveson
Information and Computer Science Department
University of California, Irvine
Irvine, CA 92717

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position.
It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl H. Shingler
SEI Joint Program Office

ESD Report Number: ESD-TR-87-124

Software Safety

Contents

Software Safety

Module Revision History

Version 1.1 (July 1987)	format changes for title page and front matter
Version 1.0 (April 1987)	original version

Software Safety

Capsule Description

Software safety involves ensuring that software will execute within a system context without resulting in unacceptable risk. Building safety-critical software requires special procedures to be used in all phases of the software development process. This module introduces the problems involved in building such software along with the procedures that can be used to enhance the safety of the resulting software product.

Philosophy

Software safety has been an important topic for many years in defense and aerospace systems. Recently it has become even more important for software engineers to study this topic as computers are increasingly used to monitor and control safety-critical devices and processes in such disparate areas as medicine, transportation, energy, manufacturing, etc. Unfortunately, few software engineers are trained in this area. The purpose of this module is to introduce the problems involved in building safety-critical systems and to provide instruction on some of the procedures and techniques that can be used to solve these problems. There is a desperate need for software engineers with this training, yet few classes currently exist.

The module could be used as part of another course or as a complete course on its own. A survey paper is available to supplement a short presentation, while a book is planned to support a more extensive presentation of the subject matter.

The module includes both introductory material and in-depth presentation of detailed analysis techniques. Because software engineers building embedded systems will need to interact with system engineers, the module includes not only a discussion of software engineering techniques but also an introduction to some system safety engineering procedures with which they will need to be familiar.

Objectives

The goal of the module is to equip software engineers with the extra knowledge and skills necessary to participate in a safety-critical software development project. The positions for which they are qualified will depend upon the length of time spent on the module and depth of presentation. All software engineers should have some appreciation of the problems involved in such projects. Those who will participate in these projects but not be responsible for safety should, in addition, have some exposure to the analysis techniques and be able to implement the design techniques. Software engineers who are responsible for software safety must, in addition, be able to carry out required analysis and verification activities. The parts of the module taught and the depth with which each is presented will depend upon the objectives of the individual instructor.

Prerequisite Knowledge

There are no explicit prerequisites beyond a basic introduction to software engineering. In-depth coverage of such topics as software safety design techniques and risk assessment and measurement will require some prerequisite knowledge depending on how much background material the instructor wants to supply while teaching the course.

Module Content

Outline

Annotated Outline

Most material directly supporting this module can be found in [Leveson86]. Specific references are made to other sources where appropriate.

I. What is Software Safety

1. Introduction

Safety problems arise with the introduction of computers into safety-critical systems. An embedded computer system is one in which the computer and its software form part of some larger system, usually electromechanical, such as a ship, aircraft, missile, spacecraft, processing plant, or rapid transit system. The embedded system may be physically or only logically incorporated into the larger system. Embedded systems are usually real-time systems that must be capable of responding to input data by producing control signals within critical real time limits. Basic concepts of embedded systems should be presented, including open and closed loops, and the role of computers and humans within these loops. An introduction to these ideas can be found in [Allworth81, Kopetz79].

The instructor may want to present some examples of accidents with computer software as a contributing factor. The examples should be chosen to illustrate the various ways software can interact with other factors to lead to accidents. Examples can be found in [Leveson86] and [Neumann85].

The problem is always one of trading off benefits vs. risks. The reasons for using computers in safety-critical systems—versatility, power, performance, efficiency—must be compared against the added risks of inability to provide correct software. Even where computers can improve safety, it is not clear they will, because technological improvements often allow running greater risks.

2. What is Safety and Safety Engineering

Safety may be defined as "freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property." But safety is a relative concept. Nothing is absolutely safe under all conditions. Provide some examples and then get students to provide some. Discuss typical tradeoffs. Describe risk elimination vs. risk displacement. Safety therefore involves an attempt to provide acceptable risk.

Provide definitions of relevant terms including system, software safety, risk, hazard, accident, mishap.

Provide a brief definition of system safety engineering and its history. Describe characteristics of accidents: multifactorial, occur in interfaces, related to complexity and coupling. This is a good place to provide a detailed description of a complex accident showing how these factors were involved. Examples can be found in [Perrow84].

3. Why is There a Safety Problem with Software

Safety is not just a matter of increasing reliability. We are currently unable to achieve ultra-high reliability in software (explain how reliability is specified and defined, hardware reliability vs. software reliability). Hardware reliability techniques are aimed toward random failures. Software systems often involve increased coupling and complexity. Extensive reuse of certified software is presently infeasible. Exhaustive testing and verification is impractical for most software. Operating conditions often differ from test conditions. There is no way to guarantee that simulations are accurate. Assumptions must be made about the controlled process and its environment. The problem is amplified when writing software for hardware that is new or does not yet exist. There are great difficulties in writing correct software requirements. Software fault tolerance techniques are limited in effectiveness.

4. Relationship of Safety with Other Software Qualities

Discuss the relationship of safety with reliability, availability, security. All can be put under a general label of dependability [Laprie82].

II. Introduction to System Safety

1. Motivation

It is important for software engineers to understand something about system safety for two reasons:

- Software is just one part of the system; in fact, software is only unsafe when operating within a hazardous system context. Software engineers must work closely with safety and system engineers to make the entire system safe. This requires that software engineers understand basic system safety techniques.
- Software often replaces standard hardware safety devices such as interlocks. Unless the software includes the software-equivalent devices, safety will be compromised.

2. Hazard Analysis

Describe the goals and techniques involved in the following analyses using examples to illustrate them: preliminary hazard analysis, subsystem hazard analysis, system hazard analysis, operating and support hazard analysis.

3. Hazard Control

The goal is to eliminate hazards or to minimize their occurrence and/or severity (effects). Present the order of precedence for applying safety design measures along with examples of each in hardware:

- design for intrinsic safety
- design to prevent or minimize the occurrence of hazards (e.g., monitoring, automatic control, lockouts, lockins, interlocks)
- design to control hazard if it occurs using automatic safety devices (e.g., hazard detection, fail-safe designs, damage control, containment, isolation)
- provide warning devices, procedures, and training to help personnel react to hazard.

III. Management of Safety-Critical Software Projects

1. Importance in Achieving Safety

Degree of safety achieved is directly related to the amount of management emphasis. Management always needs to prioritize conflicting or costly goals.

2. Responsibilities of Software Management

Discuss the implications of setting policy and defining goals, delegating and assigning responsibility for safety, granting authority, fixing accountability, and delineating lines of authority, coordination, and administration.

3. Duties of Software Safety Group

Discuss the duties of the software safety group, including the following:

- Participation in planning the software aspects of the system safety plan. Describe what such a plan is and provide an example. Requires that software group interact closely with system safety group.
- Overall responsibility for software safety analyses.
- Participation with delegated software safety responsibility in all design reviews and configuration board activities.
- Establishing and overseeing audit trails for all identified software hazards. This may involve merely responsibility for input to system hazard auditing for software hazards.
- All necessary interfacing with the system safety group.
- General participation or auditing of all aspects of software development activities to ensure that software hazards are minimized and controlled to an acceptable degree.
- Production of any documentation of the safety aspects of the software which are needed for certification or licensing of the system.

4. Management Structure

Discuss the position of the system safety management within the overall management structure of typical engineering projects and where the software safety group should be inserted.

IV. Software Safety Modeling and Analysis

1. System Hazard Analysis

The goal is to show the system is safe if it operates as intended and to show it is safe in the presence of faults. System safety techniques try to show that no single fault causes a hazard and that hazards from sequences of faults are sufficiently remote. The latter approaches the impossible if an attempt is made to combine all possible failures in all possible sequences and to analyze the output. Instead, system safety approaches often involve techniques that first define what is hazardous and then work backward to find all combinations of faults that could produce that event.

2. Software Hazard Analysis

SHA starts from the system hazard analysis and identifies software hazards including both operating and failure modes, single and multiple failure sequences. The purpose is to identify the safety-critical areas of the software for further attention.

3. Software Safety Requirements

Using examples, differentiate between mission and other requirements, including safety. Software safety requirements are derived from the software hazard analysis and system preliminary hazard analysis. They should include the requirements for detecting, eliminating, and controlling hazards and for limiting damage in case of an accident; the ways in which the software and system can fail safely; and the extent to which failure is tolerable. Stress the fact that safety requirements may conflict with other requirements, and these conflicts must be determined and resolved before software can be built.

V. Software Design Concepts to Enhance Safety

1. Why the Need for Runtime Measures

Verification and analysis is not enough because the techniques are so complex as to be error-prone themselves, the cost may be prohibitive, and elimination of all hazards may require too severe a performance penalty.

2. Hazard Prevention vs. Hazard Control

Risk is reduced by reducing hazard likelihood or severity or both. Hazards can be prevented, or they can be detected and treated. Prevention of hazards tends to involve reducing functionality or inhibiting design freedom, while detection of hazards is difficult and unreliable.

3. Hazard Prevention

The goal is intrinsic safety through design to make software faults and failures non-hazardous. General techniques include:

- minimization of complexity
- separation of safety-critical functions and data
  - limit actions of software
  - minimize interfaces (not only for reliability enhancement, but to aid in verification and certification of safety)
- firewalls
- authority limitation, access limitations
- minimization of hazardous states or time in hazardous states
- control flow limitations, sequence control (e.g., concurrency and synchronization, batons, handshaking)
- protection against credible hardware failures
- hierarchical design.

4. Hazard Detection and Treatment

a. Detection of unsafe states

The first step is identification of safety-critical items. General techniques include: assertions, monitoring, exception-handling, watchdog timers, acceptance tests, algorithm redundancy, and voting. Types of checks include: replication checks, timing checks, reversal checks, coding checks, reasonableness checks, structural checks, diagnostic checks, and checks for hazardous conditions.

b. Recovery

Differentiate between fail-operational, fail-soft, and fail-safe behavior; backward vs. forward recovery. Discuss masking through redundancy and voting (n-version programming), backward recovery (recovery blocks), forward recovery (robust data structures, dynamic alteration of flow of control, reconfiguration, ignoring single cycle errors, reduced function multiple control modes, designing for a safe side).

VI. Verification and Certification of Safety

Differentiate between verification, validation, and certification in terms of product and process. Describe and give examples of SFTA, Software Common Mode Analysis, SNSA.

VII. Assessment of Safety

1. Introduction to Quantitative Risk Assessment

Present a general introduction to quantitative risk analysis [Morgan81a, Morgan81b]: single-valued best estimate, probabilistic, bounding. Discuss the pros and cons of assessment (caveats). As an example, it might be interesting to include a discussion of WASH 1400 and the pro and con discussion raised by it. Discuss the Titanic effect.

2. Probabilistic Risk Assessment

Present probabilistic use of fault trees and event trees, minimal cut sets, evaluation of boolean expressions [Vesely81].

3. Software Reliability and Safety

Describe the state of the art in software reliability and safety models and general principles of software reliability modeling along with weaknesses of metrics and reliability growth models [Littlewood80, Kopetz79]. Discuss the general approaches that have been proposed for adding cost into these models [Arlat85, Friedman86, Cheung80].

VIII. Man/Machine Interface Considerations

Software engineers need to interact with human factors experts and thus should understand some of the important issues to be resolved with respect to human factors and software requirements.

Discuss the issues in allocation of tasks between man and machine: complementary vs. incompatible tasks, static allocation vs. dynamic allocation, the human as monitor vs. controller, complacency and situational awareness; issues in selecting amount and type of information to give human under normal and emergency conditions; maintaining human confidence in the system.

IX. Miscellaneous Issues

Discuss social issues: e.g. what systems should be built using computers; regulation (how? does the government have the right to regulate?); ethical and moral considerations for those who must build potentially dangerous systems; liability issues.

Teaching Considerations

Suggested Schedules

The material in the module can be taught in different ways, depending on the time available. The following examples show the number of hours that might be spent on a short course (first number in the parentheses) or a full length course (second number in the parentheses). These numbers are based on a total course length of 10 to 30 hours. Shorter presentations could be derived by leaving out topics and longer ones by presenting more in-depth material.

I. What is Software Safety (1 - 3)

II. Introduction to System Safety (1 - 2)

III. Management of Safety-Critical Software Projects (.5 - 2)

IV. Software Safety Modeling and Analysis (2 - 4)

V. Software Design Concepts to Enhance Safety (2 - 4)

VI. Verification and Certification of Safety (2 - 4)

VII. Assessment of Safety (.5 - 5)

VIII. Man/Machine Interface (.5 - 2)

IX. Miscellaneous Issues (.5 - 4)

Support Materials

It is recognized that teaching this material will require ample support materials. There is a survey paper available now to be used with a short version of the course and a book in preparation for a more in-depth course. The support materials package for the module will eventually include sample safety plans, sample safety requirements, sample system and software hazard analyses, a fault tree analysis, etc.

Exercises

Exercises and discussion questions are provided in the annotated outline for the course. In addition, a semester project might involve the students preparing a preliminary hazard analysis and fault tree for some common activity with which the student is familiar. The second step would be to hypothesize the introduction of a robot into such a setting. The overall safety requirements would be Asimov's Three Laws of Robotics [Asimov84]:

1. A robot may not injure a human being, or through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Using these high-level requirements, the student must write system and software safety requirements for the robot. The results might be used by the class in a mock safety review.

Another type of exercise might involve the students actually performing a software fault tree analysis on a piece of code.

Bibliography

Because the subject is new, it will be necessary for anybody teaching this module to do some reading and learning. The main reference available right now is [Leveson86], which has pointers to other papers and textbooks. A textbook on software safety by Leveson to accompany this module is in preparation and should be ready by early 1988.

Allworth81
Allworth, S. T. *Introduction to Real-Time Software Design.* Springer-Verlag, New York, 1981.

A nice introduction to real-time systems and how to build them. This provides a very good introduction to the basic techniques involved in designing software for these systems.

Anderson81
Anderson, T., and P. A. Lee. *Fault Tolerance: Principles and Practice.* Prentice-Hall, Englewood Cliffs, N.J., 1981.

A comprehensive introduction to fault tolerance. This book may be out of print. If so, see [Anderson86].

Anderson86
Anderson, T. *Resilient Computing Systems.* John Wiley, New York, 1986.

This book is not as complete as [Anderson81].

Arlat85
Arlat, J., and J. C. Laprie. On the Dependability Evaluation of High Safety Systems. *Proc. 15th Intl. Symp. on Fault Tolerant Computing.* IEEE Computer Society Press, June, 1985, 318-323.

Abstract: The definition and application of dependability measures (including both safety and availability aspects) suited to the evaluation of high-safety computer systems is described. Based on the derivation of exact and approximate values for the considered measures using homogeneous Markov processes, a comprehensive evaluation methodology is presented and applied to the design and validation of a specific computerized interlocking system for railway applications.

Asimov84
Asimov, Isaac. *I Robot.* Ballantine, 1984.

Cheung80
Cheung, R. C. A User-Oriented Software Reliability Model. *IEEE Trans. Software Eng. SE-6*, 2 (March 1980), 118-125.

Abstract: A user-oriented reliability model has been developed to measure the reliability of service that a system provides to a user community. It has been observed that in many systems, especially software systems, reliable service can be provided to a user when it is known that errors exist, provided that the service requested does not utilize the defective parts. The reliability of service, therefore, depends both on the reliability of the components and the probabilistic distribution of the utilization of the components to provide the service. In this paper, a user-oriented software reliability figure of merit is defined to measure the reliability of a software system with respect to a user environment. The effects of the user profile, which summarizes the characteristics of the users of a system, on system reliability are discussed. A simple Markov model is formulated to determine the reliability of a software system based on the reliability of each individual module and the measured intermodular transition probabilities as the user profile. Sensitivity analysis techniques are developed to determine modules most critical to system reliability. The applications of this model to develop cost-effective testing strategies and to determine the expected penalty cost of failures are also discussed. Some future refinements and extensions of the model are presented.

Friedman86
Friedman, M. *Modeling the Penalty Costs of Software Failure*. Ph.D. Th., Univ. of California, Irvine, March 1986.

Kopetz79
Kopetz, H. *Software Reliability.* Springer-Verlag, New York, 1979.

This small paperback book serves as an excellent introduction to software reliability concepts and techniques.

Laprie82
Laprie, J. C., and A. Costes. Dependability: A Unifying Concept for Reliable Computing. *Proc. 12th Intl. Symp. on Fault Tolerant Computing.* IEEE, June, 1982, 18-21.

Abstract: Presents an attempt to provide a conceptual framework for expressing the attributes of what *constitutes reliable computing. The topics addressed are dependability achievement, error characterization, error processing, and dependability measures.*

Leveson83

Leveson, N. G., and P. R. Harvey. Analyzing Software Safety. *IEEE Trans. Software Eng. SE-9*, 5 (Sept. 1983), 569-579.

Abstract: With the increased use of software controls in critical real-time applications, a new dimension has been introduced into software reliability—the "cost" of errors. The problems of safety have become critical as these applications have increasingly included areas where the consequences of failure are serious and may involve grave dangers to human life and property. This paper defines software safety and describes a technique called software fault tree analysis which can be used to analyze a design as to its safety. The technique has been applied to a program which controls the flight and telemetry for a University of California spacecraft. A critical failure scenario was detected by the technique which had not been revealed during substantial testing of the program. Parts of this analysis are presented as an example of the use of the technique and the results are discussed.

An introduction to software fault tree analysis along with an example of the technique applied to a real satellite control software program.

Leveson86

Leveson, N. G. Software Safety: Why, What, and How. *Computing Surveys 18*, 2 (June 1986), 125-163.

Abstract: Software safety issues become important when computers are used to control real-time, safety-critical processes. This survey attempts to explain why there is a problem, what the problem is, and what is known about how to solve it. Since this is a relatively new software research area, emphasis is placed on delineating the outstanding issues and research topics.

There is a very long bibliography at the end to aid in finding further information.

Littlewood80

Littlewood, B. Theories of software reliability: How good are they and how can they be improved. *IEEE Trans. Software Eng. SE-6*, 5 (Sept. 1980), 489-500.

Abstract: An examination of the assumptions used in early bug-counting models of software reliability shows them to be deficient. Suggestions are made to improve modeling assumptions and examples are given of mathematical implementations. Model verification via real-life data is discussed and minimum requirements are presented. An example shows how these requirements may be satisfied in practice. It is suggested that current theories are only the first step along what threatens to be a long road.

An explanation and evaluation of software reliability measurement.

Morgan81a

Morgan, M. G. Probing the Question of Technology-Induced Risk. *IEEE Spectrum 18*, 11 (Nov. 1981), 58-64.

Abstract: In the first of two articles on risk assessment and management, the author explores what constitutes risk. A second article will examine questions of regulation and management of risk.

An introduction to the problems and techniques involved in risk assessment.

Morgan81b

Morgan, M. G. Choosing and Managing Technology-Induced Risk. *IEEE Spectrum 18*, 12 (Dec. 1981), 53-60.

Abstract: This is the second of two articles on risk assessment. In the first, the author developed a framework for thinking about risk. In this framework, two processes—exposures and effects—act upon natural processes and human activities to produce some effect or change. The other two processes—perception and evaluation—act upon this change and develop some notion of risk through costs and benefits, and this is developed upon.

Neumann85

Neumann, P. G. Some Computer-Related Disasters and Other Egregious Horrors. *ACM Software Engineering Notes 10*, 1 (Jan. 1985), 6-7.

Perrow84

Perrow, C. *Normal Accidents: Living with High Risk Technologies.* Basic Books, New York, 1984.

This book does not really deal with computers, but it is an excellent study of the factors involved in accidents in complex systems. It includes details of many accidents in various application areas including nuclear power, transportation, manufacturing, energy, and defense. It also discusses the causes of accidents from an organizational standpoint. It is extremely interesting and easy to read.

Petersen71

Petersen, D. *Techniques of Safety Management.* McGraw-Hill, New York, 1971.

This is just one of many basic textbooks on system safety engineering. Most libraries have a selection of several, all of which are basically similar.

Roland83

Roland, H. E., and B. Moriarty. *System Safety Engineering and Management.* John Wiley, New York, 1983.

Another basic textbook on system safety engineering, somewhat newer than [Petersen71].

Rouse81

Rouse, W. B. Human-Computer Interaction in the Control of Dynamic Systems. *Computing Surveys 13*, 1 (March 1981), 71-100.

Abstract: Modes of human-computer interaction in the control of dynamic systems are discussed, and the problem of allocating tasks between human and computer considered. Models of human performance in a variety of tasks associated with the control of dynamic systems are reviewed. These models are evaluated in the context of a design example involving chemical plants, and ships.

A nice introduction to human factors issues. It also includes an extensive bibliography.

Vesely81

Vesely, W. E., F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook.* NUREG-0492, U.S. Nuclear Regulatory Commission, Jan., 1981.

A very easily read textbook on fault tree analysis and probabilistic hazard analysis.