Chapter 10

System Software Safety

10.0 SYSTEM SOFTWARE SAFETY

10.1 Introduction

Much of the information in this chapter has been extracted from the JSSSC Software System Safety Handbook, December, 1999, and concepts from DO-178B, Software Considerations in Airborne Systems and Equipment Certification, December 1, 1992.

Since the introduction of the digital computer, system safety practitioners have been concerned with the implications of computers performing safety-critical or safety-significant functions. In earlier years, software engineers and programmers constrained software from performing in high risk or hazardous operations where human intervention was deemed both essential and prudent from a safety perspective. Today, however, computers often autonomously control safety critical functions and operations. This is due primarily to the capability of computers to perform at speeds unmatched by its human operator counterpart. The logic of the software also allows for decisions to be implemented unemotionally and precisely. In fact, some current operations no longer include a human operator.

Software that controls safety-critical functions introduce risks that must be thoroughly addressed (assessed and mitigated?) during the program by both management and design , software , and system safety engineering. In previous years, much has been written pertaining to "Software Safety" and the problems faced by the engineering community. However, little guidance was provided to the safety practitioner that was logical, practical, or economical. This chapter introduces an approach with engineering evidence that software can be analyzed within the context of both the systems and system safety engineering principles. The approach ensures that the safety risk associated with software performing safety-significant functions is identified, documented, and mitigated while supporting design-engineering objectives along the critical path of the system acquisition life cycle.

The concepts of risk associated with software performing safety-critical functions were introduced in the 1970's. At that time, the safety community believed that traditional safety engineering methods and techniques were no longer appropriate for software safety engineering analysis. This put most safety engineers in the position of “wait and see.” Useful tools, techniques, and methods for safety risk management were not available in the 1970's even though software was becoming more prevalent in system designs.

In the following two decades, it became clear that traditional safety engineering methods were indeed partially effective in performing software safety analysis by employing traditional approaches to the problem. This situation does not imply, however, that some modified techniques are not warranted. Several facts must be realized before a specific software safety approach is introduced. These basic facts must be considered by the design engineering community to successfully implement a system safety methodology that addresses the software implications.

- Software safety is a systems issue, not a software-specific issue. The hazards caused by software must be analyzed and solved within the context of good systems engineering principles.
- An isolated safety engineer may not be able to produce effective solutions to potential software-caused hazardous conditions without the assistance of supplemental expertise. The software safety "team" should consist of the safety engineer, software engineer, system engineer, software quality engineer, appropriate "ility" engineers (configuration

management, test & evaluation, verification & validation, reliability, and human factors), and the subsystem domain engineer.

- Today's system-level hazards, in most instances, contain multiple contributing factors from hardware, software, human error, and/or combinations of each, and,
- Finally, software safety engineering cannot be performed effectively outside the umbrella of the total system safety engineering effort. There must be an identified link between software faults, conditions, contributing factors, specific hazards and/or hazardous conditions of the system.

The safety engineer must also never lose sight of the basic, fundamental concepts of system safety engineering. The product of the system safety effort is not to produce a hazard analysis report, but to influence the design of the system to ensure that it is safe when it enters the production phase of the acquisition life cycle. This can be accomplished effectively if the following process tasks are performed:

- Identify the safety critical functions of the system.
- Identify the system and subsystem hazards/risks.
- Determine the effects of the risk occurrence.
- Analyze the risk to determine all contributing factors (i.e.. hardware, software, human error, and combinations of each.)
- Categorize the risk in terms of severity and likelihood of occurrence.
- Determine requirements for each contributing factor to eliminate, mitigate, and/or control the risk to acceptable levels. Employ the safety order of design precedence Chapter 3, Table 3-7, for hazard control.
- Determine testing requirements to prove the successful implementation of design requirements where the hazard risk index warrants.
- Determine and communicate residual safety risk after all other safety efforts are complete to the design team and program management.

10.2 The Importance of System Safety

Before an engineer (safety, software, or systems) can logically address the safety requirements for software, a basic understanding of how software “fails” is necessary. Although the following list may not completely address every scenario, it provides the most common failure mechanisms that should be evaluated during the safety analysis process.

- Failure of the software to perform a required function, i.e., either the function is never executed or no answer is produced.
- The software performs a function that is not required, i.e., getting the wrong answer, issuing the wrong control instruction, or doing the right action but under inappropriate conditions.
- The software possesses timing and/or sequencing problems, i.e., failing to ensure that two things happen at the same time, at different times, or in a particular order.

- The software failed to recognize that a hazardous condition occurred requiring corrective action.

- The software failed to recognize a safety-critical function and failed to initiate the appropriate fault tolerant response.

- The software produced the intended but inappropriate response to a hazardous condition.

- The specific causes most commonly associated with the software failure mechanisms listed above are:

- Specification Errors: Specification errors include omitted, improperly stated, misunderstood, and/or incorrect specifications and requirements. Software may be developed "correctly" with regard to the specification, but wrong from a systems perspective. This is probably the single largest cause of software failures and/or errors.

- Design and Coding Errors: These errors are usually introduced by the programmer and can result from specification errors, usually the direct result of poor structured programming techniques. These errors can consist of incomplete interfaces, timing errors, incorrect interfaces, incorrect algorithms, logic errors, lack of self-tests, overload faults, endless loops, and syntax errors. This is especially true for fault tolerant algorithms and parameters.

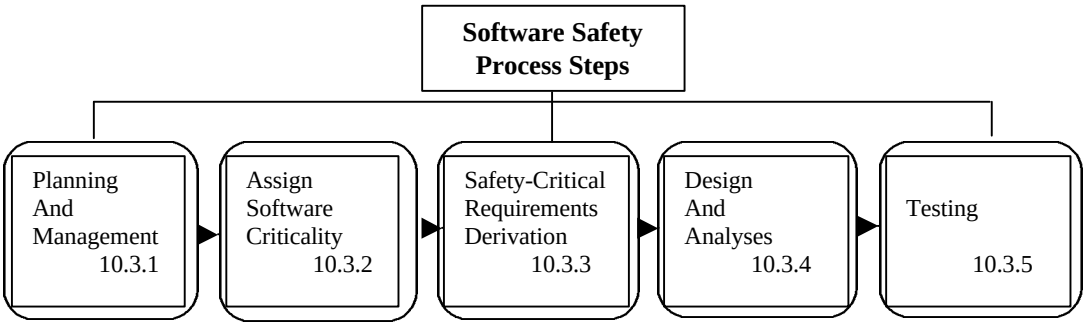
- Hardware/Computer Induced Errors: Although not as common as other errors, then can exist. Possibilities include random power supply transients, computer functions that transform one or more bits in a computer word that unintentionally change the meaning of the software instruction, and hardware failure modes that are not identified and/or corrected by the software to revert the system to a safe state.

- Documentation Errors: Poor documentation can be the cause of software errors through miscommunication. Miscommunication can introduce the software errors mentioned above. This includes inaccurate documentation pertaining to system specifications, design requirements, test requirements, source code and software architecture documents including data flow and functional flow diagrams.

- Debugging/Software Change Induced Hazards: These errors are basically self-explanatory. The cause of these errors can be traced back to programming and coding errors, poor structured programming techniques, poor documentation, and poor specification requirements. Software change induced errors help validate the necessity for software configuration.

10.3 Software Safety Development Process

The process outlined below is briefly explained in this Handbook. Further guidance and specific instructions can be obtained through a careful examination of the JSSSC Software System Safety Handbook, Dec. 1999 and DO-178B, Software Considerations in Airborne Systems and Equipment Certification, Dec. 1, 1992 at a minimum.



10.3.1 Software Safety Planning and Management

Software system safety planning precedes all other phases of the software systems safety program. It is perhaps the single most important step and should impose provisions for accommodating safety well before each of the software life cycle phases: requirements, design, coding, and testing starts in the cycle. Detailed planning ensures that critical program interfaces and support are identified and formal lines of communication are established between disciplines and among engineering functions. The software aspects of systems safety tend to be more problematic in this area since the risks associated with the software are often ignored or not well understood until late in the system design.

Planning Provisions

The software system safety plan should contain provisions assuring that:

- Software safety organization is properly chartered and a safety team is commissioned at the beginning of the life cycle.
- Acceptable levels of software risk are defined consistently with risks defined for the entire system.
- Interfaces between software and the rest of the system's functions are clearly delineated and understood.
- Software application concepts are examined to identify hazards/risks within safety-critical software functions.
- Requirements and specifications are examined for hazards (e.g. identification of hazardous commands, processing limits, sequence of events, timing constraints, failure tolerance, etc.)
- Design and implementation is properly incorporated into the software safety requirements.

- Appropriate verification and validation requirements are established to assure proper implementation of software system safety requirements.
- Test plans and procedures can achieve the intent of the software safety verification requirements.
- Results of software safety verification efforts are satisfactory.

Software Safety Team

Software safety planning also calls for creating a software safety team. Team size and shape depends commensurately on mission size and importance (see Figure 10-1). To be effective, the team should consist of analytical individuals with sufficient system engineering background. Chapter 5 of this handbook provides a comprehensive matrix of minimum qualifications for key system safety personnel. It applies to software system safety provided professional backgrounds include sufficient experience with software development (software requirements, design, coding, testing, etc.)

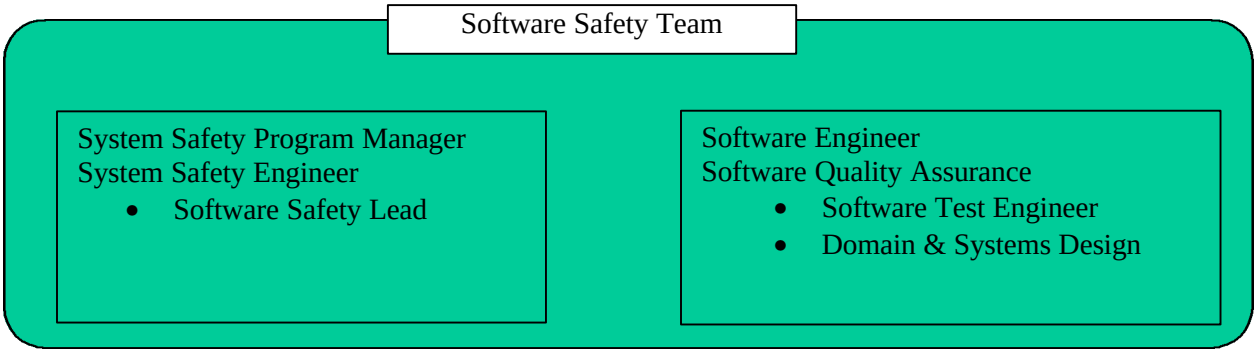


Figure 10-1: Example Membership of Software System Safety Team

Several typical activities expected of the team range from identifying software-based hazards to tracing safety requirements, from identifying limitations in the actual code to developing software safety test plans and ultimately reviewing test results for their compliance with safety requirements.

Management

Software System Safety program management begins as soon as the System Safety Program (SSP) is established and continues throughout the system development. Management of the effort requires a variety of tasks or processes from establishing the Software Safety Working Group (SwSWG) to preparing the System Safety Assessment Report (SSAR). Even after a system is placed into service, management of the software system safety effort continues to address modifications and enhancements to the software and the system. Often, changes in the use or application of a system necessitate a re-assessment of the safety of the software in the new application. Effective management of the safety program is essential to the effective reduction of the system risk. Initial efforts parallel portions of the planning process since many of the required efforts need to begin very early in the safety program. Safety management pertaining to software generally ends with the completion of the program and its associated testing; whether it is a single phase of the development process or continues throughout the development, production, deployment and maintenance phases. Management efforts end when the last safety deliverable is completed and is accepted by the FAA. Management efforts may then revert to a “caretaker” status in which the safety manager monitors the use of

the system in the field and identifies potential safety deficiencies based on user reports and accident/incidents reports. Even if the developer has no responsibility for the system after deployment, the safety program manager can develop a valuable database of lessons learned for future systems by identifying these safety deficiencies.

Establishing a software safety program includes establishing a SwSWG. This is normally a sub-group of the SSWG and chaired by the safety manager. The SwSWG has overall responsibility for the following:

- Monitoring and control of the software safety program
- Identifying and resolving risks with software contributory factors
- Interfacing with the other IPTs, and
- Performing final safety assessment of the system (software) design.

10.3.2 Assign Software Criticality

The ability to prioritize and categorize hazards is essential for the allocation of resources to the functional area possessing the highest risk potential. System safety programs have historically used the Hazard Risk Index (HRI) to categorize hazards. However, the methodology to accurately categorize hazards using this traditional HRI matrix for hazards possessing software causal factors is insufficient. The ability to use the original (hardware oriented) HRI matrix was predicated on the probability of hazard occurrence and the ability to obtain component reliability information from engineering sources. The current technologies associated with the ability to accurately predict software error occurrence, and quantify its probability, is still in its development infancy. This is due to the nature of software as opposed to hardware. Statistical data may be used for hardware to predict failure probabilities. However, software does not fail in the same manner as hardware (it does not wear out, break, or have increasing tolerances). Software errors are generally requirements errors (failure to anticipate a set of conditions that lead to a hazard, or influence of an external component failure on the software) or implementation errors (coding errors, incorrect interpretation of design requirements). Therefore, assessing the risk associated with software is somewhat more complex. Without the ability to accurately predict a software error occurrence, supplemental methods of hazard categorization must be available when the hazard possesses software causal factors. This section of the handbook presents a method of categorizing hazards that possess software influence or causal factors.

Risk Severity

Regardless of the contributory factors (hardware, software, human error, and software influenced human error) the severity of the risk could remain constant. This is to say that the consequence of risk remains the same regardless of what actually caused the hazard to propagate within the context of the system. As the severity is the same, the severity tables presented in Chapter 3 remain applicable criteria for the determination of risk severity for those hazards possessing software causal factors.

Risk Probability

With the difficulty of assigning accurate probabilities to faults or errors within software modules of code, a supplemental method of determining risk probability is required when software causal factors exist. Figure 10-2 demonstrates that in order to determine a risk probability, software contributory factors must be assessed in conjunction with the contributors from hardware and human error. The determination of hardware and human error contributor probabilities remain constant in terms of historical “best” practices. However, the likelihood of the software aspect of the risk's cumulative causes must be addressed.

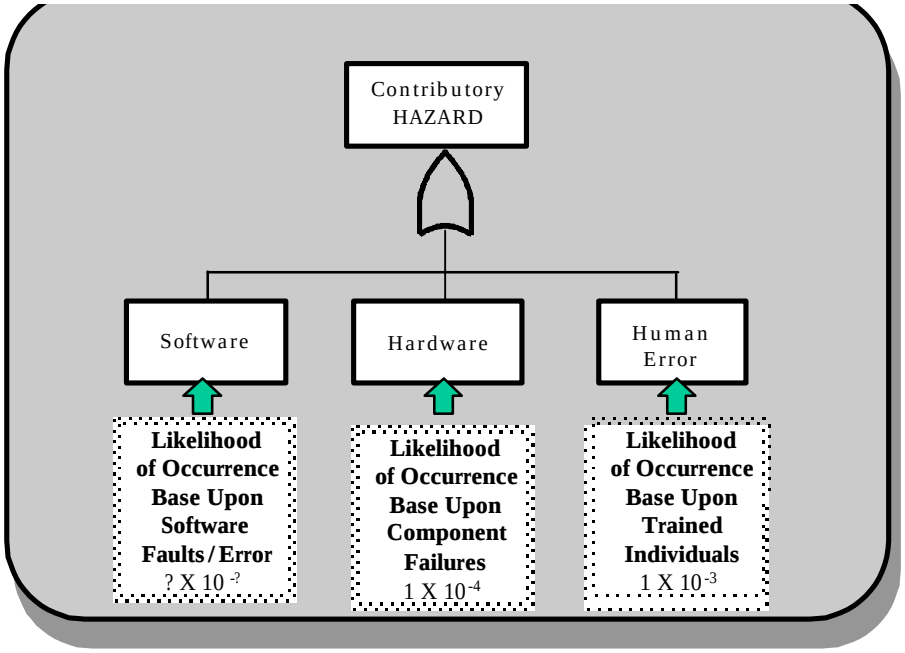


Figure 10-2: Likelihood of Occurrence Example

There have been numerous methods of determining the software's influence on system-level risks. Two of the most popular software listings are presented in MIL-STD 882C and RTCA DO-178B (see Figure 10-3). These do not specifically determine software-caused risk probabilities, but instead assesses the software's "control capability" within the context of the software contributors . In doing so, each software contributors can be labeled with a software control category for the purpose of helping to determine the degree of autonomy that the software has on the hazardous event. The software safety team must review these lists and tailor them to meet the objectives of the system safety and software development programs.

MIL-STD 882C	*RTCA-DO-178B*
(I) Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazards occurrence.	**(A)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
(IIa) Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.	**(B)** Software whose anomalous behavior, as shown by the System Safety assessment process, would cause or contribure to a failure of system function resulting in a hazardous/severe-major failure condition of the aircraft.
(IIb) Software item displays information requiring immediate operator action to mitigate a hazard. Software failure will allow or fail to prevent the hazard's occurrence.	**(C)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the the aircraft.
(IIIa) Software items issues commands over potentially hazardous hardware systems, subsystem, or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.	**(D)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
(IIIb) Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.	**(E)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.
(IV) Software does not control safety critical hardware systems, subsystems, or components and does not provide safety critical information.	

Figure 10-3: Examples of Software Control Capabilities

Once again, the concept of labeling software contributors with control capabilities is foreign to most software developers and programmers. They must be convinced that this activity has utility in the identification and prioritization of software entities that possesses safety implication. In most instances, the software development community desires the list to be as simplistic and short as possible. The most important aspect of the activity must not be lost, that is, the ability to categorize software causal factors for the determining of both risk likelihood, and the design, code, and test activities required to mitigate the potential software cause. Autonomous software with functional links to catastrophic risks demand more coverage than software that influences low-severity risks.

Software Hazard Criticality Matrix

The Software Hazard Criticality Matrix (SHCM) (see Figure 10-4 for an example matrix) assists the software safety engineering team and the subsystem and system designers in allocating the software safety requirements between software modules and resources, and across temporal boundaries (or into separate architectures). The software control measure of the SHCM also assists in the prioritization of software design and programming tasks.

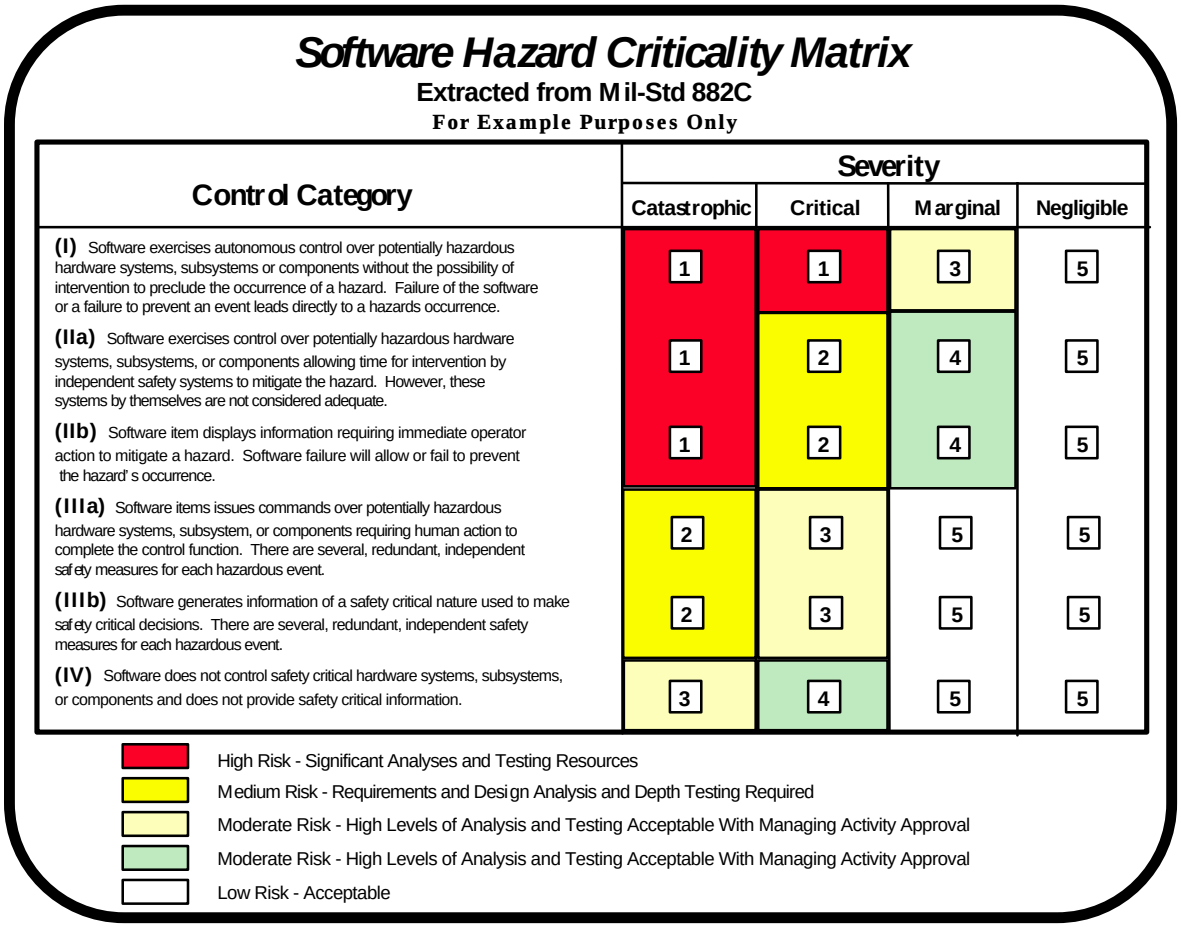


Figure 10-4: Software Hazard Criticality Matrix

10.3.3 Derivation of System Safety-Critical Software Requirements

Safety-critical software requirements are derived from known safety-critical functions, tailored generic software safety requirements and inverted contributory factors determined from previous activities. Safety requirement specifications identify the specifics and the decisions made, based upon the level of risk, desired level of safety assurance, and the visibility of software safety within the developer organization. Methods for doing so are dependent upon the quality, breadth and depth of initial hazard and failure mode analyses and on lessons-learned derived from similar systems. The generic list of requirements and guidelines establish the beginning point that initiates the system-specific requirements identification process. System-specific software safety requirements require a flow-down of hazard controls into requirements for the subsystems which provide a trace (audit trail) between the requirement, its associated risk and to the module(s) of code that are affected. Once this is achieved as a core set of requirements, design decisions are identified, assessed, implemented, and included in the hazard record database. Relationships to other risks or requirements are also determined. The identification of system-specific requirements (see Figure 10-5) is the direct result of a complete hazard analysis methodology.

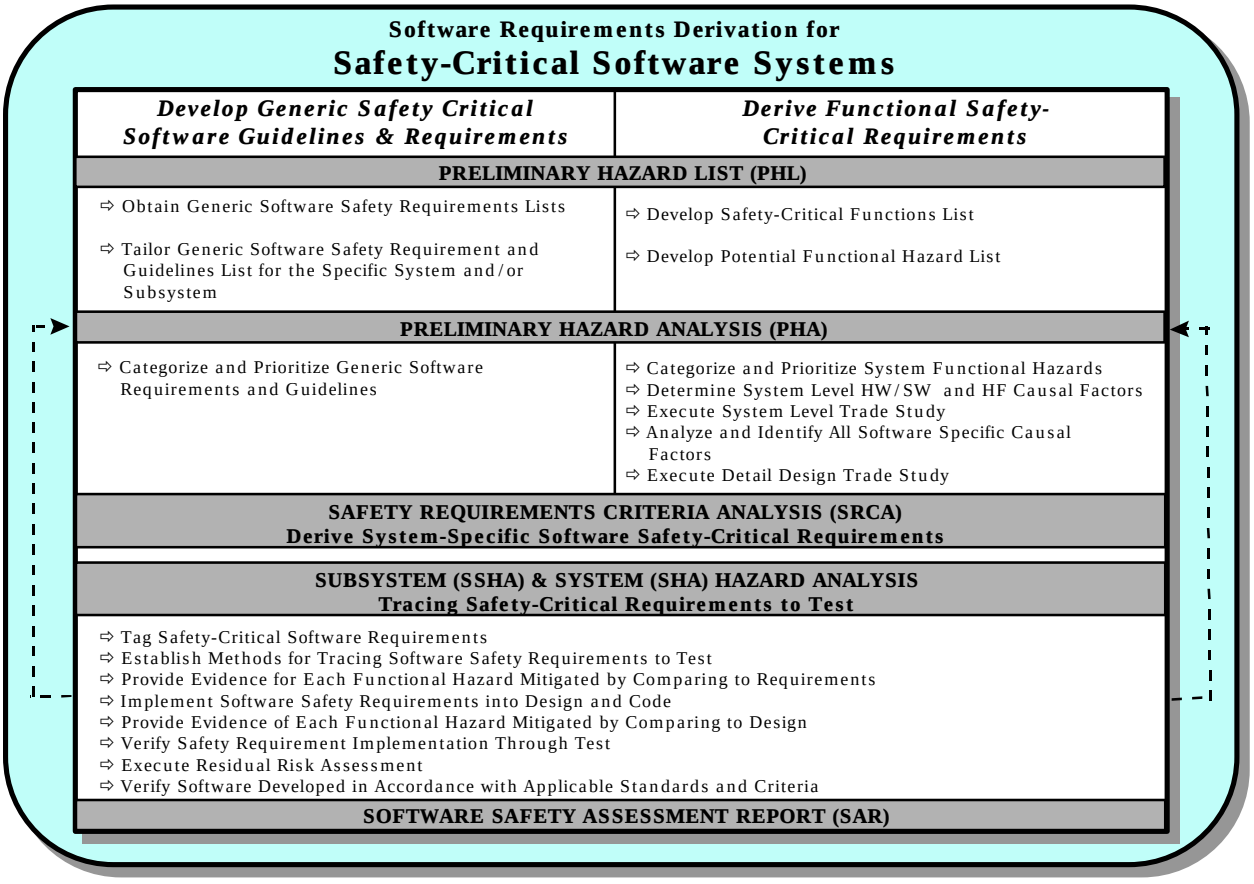


Figure 10-5: Software Safety Requirements Derivation

Preliminary Software Safety Requirements

The first "cut" at system-specific software safety requirements are derived from the PHA analyses performed in the early life cycle phase of the development program. As previously discussed, the PHL/PHA hazards are a product of the information reviewed pertaining to systems specifications, lessons learned, analyses from similar systems, common sense, and preliminary design activities. Hazards that are identified during the PHA phase are analyzed and preliminary design considerations are identified to design engineering to mitigate the risk. These design considerations represent the preliminary safety requirements of the system, subsystems, and their interfaces (if known). These preliminary requirements must be accurately defined in the hazard record database for extraction when reporting of requirements to the design engineering team.

Matured Software Safety Requirements

As the system and subsystem design mature, the requirements unique to each subsystem also matures via the Subsystem Hazard Analysis (SSHA). The safety engineer, during this life cycle phase of the program, attends the necessary design reviews and spends many hours with the subsystem designers for the purpose of accurately defining the subsystem hazards. Hazards/risks identified are documented in the hazard database and the hazard "causes" (hardware, software, human error, and software-influenced human error) identified and analyzed. When fault trees are used as the functional hazard analysis methodology, the contributors leading to the risk determine the derived safety-critical functional requirements. It is at this point in the design that preliminary design considerations are either formalized and defined into specific requirements, or eliminated if they no longer apply with the current design concepts. The maturation of safety requirements is accomplished by analyzing the design architecture to connect the risk to the contributors. The causal factors are analyzed to the lowest level necessary for ease of mitigation. The lower into the design the analysis progresses, the more simplistic (usually) and cost effective the mitigation requirements tend to become. The PHA phase of the program should define causes to at least the Computer Software Configuration Item (CSCI) level, whereas the SSHA and System Hazard Analysis (SHA) phases of safety analyses should analyze the causes to the algorithm level where appropriate.

10.3.4 Design and Analyses

The identification of subsystem and system hazards and failure modes inherent in the system under developed is essential to the success of a credible software safety program. The primary method of reducing the safety risk of software performing safety-significant functions is to first identify the system hazards and failure modes, and then determine which hazards and failure modes are *caused by* or *influenced by* software or lack of software. This determination includes scenarios where information produced by software could potentially influence the operator into a wrong decision resulting in a hazardous condition (design-induced human error). Moving from hazards to software contributors (and consequently design requirements to either eliminate or control the risk) is very practical, logical, and adds utility to the software development process. It can also be performed in a timelier manner as much of the analysis is accomplished to influence preliminary design activities.

The specifics of how to perform either a subsystem or system hazard analysis are briefly described in Chapters 8 and 9. The fundamental basis and foundation of a system safety (or software safety) program is a systematic and complete hazard analysis process.

One of the most helpful steps within a credible software safety program is to categorize the specific causes of the hazards and software inputs in each of the analyses (PHA, SSHA, SHA, and Operating & Support Hazard Analysis (O&SHA)). Hazard causes can be identified as those caused by; hardware, and/or hardware components; software inputs or lack of software input; human error; and/or software influenced human error or hardware or human errors propagating through the software. Hazards may result from one specific cause

or any combination of causes. As an example, "loss of thrust" on an aircraft may have causal factors in any of the four below listed categories.

- **Hardware**: foreign object ingestion,
- **Software**: software commands engine shutdown in the wrong operational scenario,
- **Human error**: pilot inadvertently commands engine shutdown, and,
- **Software influence pilot error**: computer provides incorrect information, insufficient or incomplete data to the pilot causing the pilot to execute a shutdown.

The safety engineer must identify and define the hazard control considerations (PHA phase) and requirements (SSHA, SHA, and O&SHA phases) for the design and development engineers. Hardware causes are communicated to the appropriate hardware design engineers; and software related causes to the software development and design team. All requirements should be reported to the systems engineering group for their understanding and necessary tracking and/or disposition.

The preliminary software design SSHA begins upon the identification of the software subsystem and uses the derived system specific safety-critical software requirements. The purpose is to analyze the system, software architecture and preliminary CSCI design. At this point, all generic and functional Software Safety Requirements (SSRs) should have been identified and it is time to begin allocating them to the identified safety-critical functions and tracing them to the design.

The allocation of the SSRs to the identified hazards can be accomplished through the development of SSR verification trees that links safety critical and safety significant SSRs to each Safety-Critical Function (SCF). The SCFs in turn are already identified and linked to each hazard. By verifying the nodes through analysis, (code/interface, logic, functional flow, algorithm and timing analysis) and/or testing (identification of specific test procedures to verify the requirement), the Software Safety Engineer (SwSE) is essentially verifying that the design requirements have been implemented successfully. The choice of analysis and/or testing to verify the SSRs is up to the individual Safety Engineer whose decision is based on the criticality of the requirement to the overall safety of the system and the nature of the SSR. Whenever possible, the Safety Engineer should use testing for verification.

Numerous methods and analytical techniques are available to plan, identify, trace and track safety-critical CSCIs and Computer Software Units (CSUs). Guidance material is available from the Institute of Electrical and Electronic Engineering (IEEE) (Standard for Software Safety Plans), the Department of Defense (DOD) Defense Standard 00-55-Annex B, DOD-STD-2167, NASA-STD-2100.91, MIL-STD-1629, the JSSSC Software System Safety Handbook and DO-178B.

10.3.5 Testing

Two sets of analyses should be performed during the testing phase:

- Analyses before the fact to ensure validity of tests
- Analyses of the test results

Tests are devised to verify all safety requirements where testing has been selected as appropriate verification method. This is not considered here as analysis. Analysis before the fact should, as a minimum, consider test coverage for safety critical Must-Work-Functions.

Test Coverage

For small pieces of code it is sometimes possible to achieve 100% test coverage (i.e., to exercise every possible state and path of the code). However, it is often not possible to achieve 100 % test coverage due to the enormous number of permutations of states in a computer program execution, versus the time it would take to exercise all those possible states. Also there is often a large indeterminate number of environmental variables, too many to completely simulate.

Some analysis is advisable to assess the optimum test coverage as part of the test planning process. There is a body of theory that attempts to calculate the probability that a system with a certain failure probability will pass a given number of tests.

"White box" testing can be performed at the modular level. Statistical methods such as Monte Carlo simulations can be useful in planning "worst case" credible scenarios to be tested.

Test Results Analysis

Test results are analyzed to verify that all safety requirements have been satisfied. The analysis also verifies that all identified risks have been either eliminated or controlled to an acceptable level of risk. The results of the test safety analysis are provided to the ongoing system safety analysis activity.

All test discrepancies of safety critical software should be evaluated and corrected in an appropriate manner.

Independent Verification and Validation (IV&V)

For high value systems with high risk software, an IV&V organization is usually involved to oversee the software development. The IV&V organization should fully participate as an independent group in the validation of test analysis.

10.4 System Safety Assessment Report (SSAR)

The System Safety Assessment Report (SSAR) is generally a CDRL item for the safety analysis performed on a given system. The purpose of the report is to provide management an overall assessment of the risk associated with the system including the software executing within the system context of an operational environment. This is accomplished by providing detailed analysis and testing evidence that the software related hazards have been identified to the best of their ability and have been either eliminated or mitigated/controlled to levels acceptable to the FAA. It is paramount that this assessment report be developed as an encapsulation of all the analyses preformed. The SSAR shall contain a summary of the analyses performed and their results, the tests conducted and their results, and the compliance assessment. Paragraphs within the SAR need to encompass the following items:

- The safety criteria and methodology used to classify and rank software related hazards (causal factors). This includes any assumptions made from which the criteria and methodologies were derived,
- The results of the analyses and testing performed,
- The hazards that have an identified residual risk and the assessment of that risk,
- The list of significant hazards and the specific safety recommendations or precautions required to reduce their safety risk; and
- A discussion of the engineering decisions made that affect the residual risk at a system level.

The final section of the SSAR should be a statement by the program safety lead engineer describing the overall risk associated with the software in the system context and their acceptance of that risk.