

SYSTEM METHODOLOGY FOR ANALYSIS, REVIEW AND TEST (SMART) FOR SYSTEM SOFTWARE SAFETY ANALYSIS (SSSA)



Sam Oleson, Marilyn Johnson

CSC North American Public Sector

soleson@csc.com

mjohns15@csc.com

CSC Papers

2008

ABSTRACT

Keywords: System, Software, Safety, Analysis, Review, Test, Verification, Validation, Methodology, SMART, System Software Safety Analysis (SSSA), Independent Verification and Validation (IV&V)

Digital systems are increasingly being used to control and monitor complex, time-critical physical processes or mechanical devices that were once controlled by a human operators or analog methods. Due to the increase of these complex digital systems, the requirement for software safety analysis has increased. The current development verification, validation and testing processes do not adequately address software safety issues and concerns.

This paper describes CSC's SMART (System Methodology for Analysis, Review and Test) process for software safety analysis. The SMART process was designed as a 3-tier approach to ensure that all potential safety critical concerns are identified and that the best analysis method is used to verify each concern. The SMART methodology focuses on identifying the safety concerns. The safety concerns are generated from the following three areas: 1) critical functions which address the proper performance of safety critical areas of software, 2) hazards within the system that can occur due to normal and abnormal conditions, and 3) best practices, which address general software practices for developing safe software. This 3-tier approach for determining safety concerns is much more comprehensive than using only one method. These concerns are then compiled into a list of Safety Requirements (SRs) which drive the safety analysis. The difference between the SMART methodology and other common SSSA (System Software Safety Analysis) methodologies is that the safety analysis is driven by the safety concerns instead of the analysis method chosen.

In addition, SMART is flexible enough to work with the developers' Independent Verification and Validation (IV&V) organization to take advantage of products generated by IV&V to avoid duplication of effort and reduce overall cost and avoid redundancy of effort.

The SMART approach is a model that streamlines and focuses the work while consistently delivering superior products. The SMART methodology is flexible enough to address the customer concerns and provide consistent and repeatable process for the users. The SMART process is capable of being tailored at any stage of the development cycle. SMART establishes the best approach in order to meet safety goals / criteria in a cost affective manner. CSC considers the SMART

process to be a living process, which will be updated and refined as process improvements are identified.

The SMART approach is not limited to just safety analysis. By establishing the customers concerns, whether they are safety, security, or critical performance issues, the SMART methodology can be adapted to focus the system analysis to assure the customer concern/goals are met

The SMART approach is based on over 25 plus years of experience in doing software safety on Nuclear Weapons Systems, conventional weapons systems, and submarine ship control systems.

ACKNOWLEDGEMENT

Thanks to Roy Fuchs (Royal Technologies, LLC) for his valuable contribution of reviewing and editing this document. Roy's past exposure to the SMART process during his tenure as a CSC employee was invaluable. Additional thanks to Edward Herbert (CSC) in reviewing and editing the SMART process document.

TABLE OF CONTENTS

ABSTRACT	i
1. Introduction	4
1.1 SMART Purpose.....	4
1.2 SMART Background.....	4
1.3 Alternative Safety Analysis Methods.....	5
1.3.1 CSU Analysis Issues.....	6
1.3.2 Hazard Analysis Issues.....	6
1.3.3 Backward Fault Tree Analysis Issues.....	6
1.3.4 Perform FMECA Analysis.....	6
1.4 Document Outline	7
2. Software Safety Goals/Definitions	8
2.1 Software Safety Goals.....	8
2.2 Software Safety Definitions.....	8
2.3 Software Safety Misconceptions.....	9
3. SMART Overview	10
3.1 The SMART Process.....	10
3.2 SMART Approach.....	10
4. Process 1 – Software safety management	13
4.1 SSSA Management Tasks.....	13
4.2 SSSA Documentation	14
4.3 SSSA Personnel Requirements.....	15
5. Process 2 – Software Safety process/Technology Improvement.....	15
6. Process 3 – Software Safety Maintenance and Support.....	16
7. Phase I – Software Safety System Concepts/Engineering	17
7.1 Purpose	17
7.2 Inputs/Outputs	17
7.3. Phase I – Joint IV&V/Safety Tasks	18
7.3.1 Ensure Compliance with Top-Level Documentation.....	18
7.3.2 Evaluate New & Upgraded System Capabilities.....	18
7.4 Phase I - Safety Detailed Tasks.....	18
7.4.1 Establish System Critical Functions	19
7.4.2 Identify System Hazard List.....	20
7.4.3 Identify System Best Practices.....	20
7.4.4 HLSSRs	20
7.4.5 Establish System Safety Plan	20
8. Phase II – Software Safety Requirements Analysis.....	21
8.1 Purpose	21
8.2 Inputs/Outputs/Task Overview.....	21
8.3 Phase II – Joint IV&V-Safety Tasks	22
8.3.1 Review Requirements Documents	22
8.3.2 Review Interface Documents	23
8.3.3 Review Trouble Reports.....	23
8.3.4 Review Proposed Software Changes.....	24
8.3.5 Support Preliminary Design Reviews	24
8.4 Phase II – Safety Specific Tasks	24
8.4.1 Derive Program Specific Safety Requirements	24
8.4.2 Perform SCP Safety Requirements Analysis	27
8.4.3 Conduct Certification Kick-off.....	27
8.4.4 Determine Software Hazard Criticality Matrix (SHCM) Ranking.....	27
9. Phase III – Software Safety Design Verification.....	30
9.1 Purpose	30
9.2 Inputs/Outputs/task Overview	30
9.3 Phase III – Joint IV&V-Safety Tasks	31
9.3.1 Design Reviews	31
9.3.2 Design Analysis.....	32
9.4 Phase III – Safety Specific Tasks	32
9.4.1 Perform SR Design Verification.....	32
9.4.2 Determine Positive Measures and Expand VRs.....	33

10. Phase IV – Software Safety Code analysis	33
10.1 Purpose	33
10.2 Inputs/Outputs/Tasks Overview	33
10.3 Phase IV – Joint IV&V-Safety Tasks.....	34
10.3.1 Perform Traceability Code Analysis	34
10.4 Phase IV – Safety Specific Tasks	35
10.4.1 Determine Verification Method for VRs	35
10.4.2 Certification Readiness Review.....	35
10.4.3 Perform VR Code Analysis.....	35
11. Phase V – Software Safety Test.....	37
11.1 Purpose	37
11.2 Input/Output/Tasks Overview.....	37
11.3 Phase V – Joint IV&V-Safety Tasks.....	38
11.3.1 Review Software Developer Test Plan.....	38
11.3.2 Conduct Formal Testing.....	38
11.4 Phase V – Safety Specific Tasks	38
11.4.1 Safety Specific Test Planning.....	38
11.4.2 Safety Specific Test Execution.....	38
12. Phase VI – Software Safety Evaluation	39
12.1 Purpose	39
12.2 Input/Output/Tasks Overview.....	39
12.3 Phase VI – Joint IV&V/Safety Tasks.....	40
12.3.1 Review Trouble Reports.....	40
12.4 Phase VI – Safety Specific Tasks	40
12.4.1 Generate Hazard Assessments	40
Determining the HRI Value.....	41
12.4.2 Submit Safety Certification Recommendation	43
13. Phase VII – Software Safety Process Review and Documentation	43
13.1 Purpose	43
13.2 Input/Outputs/Tasks Overview.....	43
13.3 Phase VII - Joint IV&V-Safety Tasks	44
13.3.1 Review Methodology.....	44
13.4 Phase VII - Safety Specific Tasks.....	44
13.4.1 Finalize SR/VR Document.....	44
13.4.2 Finalize Test Procedures.....	44
13.4.3 Prepare Final Report.....	45
13.4.4 Finalize SAFs.....	45

LIST OF FIGURES

Figure 3-1: SMART Processes and Phases	11
Figure 7-1: Phase I – Software Safety Systems Concepts/Engineering Phase	17
Figure 8-1: Phase II – Software Safety Requirements Analysis Phase	22
Figure 8-2: Derive Safety Requirements.....	25
Figure 9-1: Phase III – Software Safety Design Verification Phase	31
Figure 10-1: Phase IV – Software Safety Code Analysis Phase.....	34
Figure 11-1: Phase V – Software Safety Test Phase.....	37
Figure 12-1: Phase VI – Software Safety Evaluation Phase	40
Figure 12-2: Hazard Assessment Example	41
Figure 13-1: Phase VII – Software Safety Process Review and Documentation Phase	44

LIST OF TABLES

Table 4-1 SSSA Formal Documentation.....	14
Table 7-1: Generic Safety Requirements.....	19
Table 8-1: System Level and Computer Program Critical Functions	25
Table 8-2: System Level and Computer Program Hazards	26
Table 8-3: System Level and Program Best Practices.....	27
Table 8-4: Software Hazard Criticality Matrix (SHCM).....	29
Table 10-1: Verification Methods for Code Analysis	36



Table 12-1: Hazard Severity Categories41
Table 12-2: Hazard Probability Levels42
Table 12-3: HRI Assessment Matrix42
Table A-1: List of Abbreviations and Acronyms48

ATTACHMENTS: TABLE OF CONTENTS

Appendix A – Abbreviations and Acronyms.....47
Appendix B – System Safety Best Practices.....50
Appendix C – Software Design / Coding Guideline Options.....52
Appendix D – References.....55

1. INTRODUCTION

The CSC System Software Safety Group (SSSG) has developed the System Methodology for Analysis, Review, and Test (SMART) process, a System Software Safety Analysis (SSSA) methodology that addresses all phases of computer program development and maintenance life cycle for safety-critical systems. Use of the SMART methodology assures that SSSG achieves repeatability and consistency in the performance of the many software safety analyses that CSC has performed for Department of Defense (DoD) clients.

1.1 SMART PURPOSE

The purpose of SMART is to provide a straightforward and comprehensive SSSA methodology for use by software developers, system and software safety analysts and their management. Use of the SMART process offers the benefits of

- Lower development costs by eliminating waste , and improving efficiency; and
- Focusing safety analysis efforts upon the most safety-critical areas.

1.2 SMART BACKGROUND

CSC's SSSG experience in performing software safety analyses is that several key questions are raised virtually always during the planning phase for a complex, safety-critical system development.

- What software safety analysis methods are available and should be considered?
- How can redundancy of analysis and testing efforts be minimized by intelligent use of Independent Verification and Validation (IV&V) and/or SSSA teams?
- What are the most appropriate criteria for selection of the best software safety analysis approach from among the alternative methods?
- Can the effectiveness of the selected safety analysis method, i.e. SMART, be assessed during the software development process? If so, what metrics might be employed?
- Does SMART process meet MIL-STD-882, Joint Software System Safety Handbook, and/or equivalent safety standards?

The SMART process was developed to address the above questions. SMART supports the System Software Safety Analysis Requirements of such standards as MIL_STD_882, Joint Software System Safety Handbook, IEEE Std 1228, STANAG 4404 and 4452, and NASA Guide Book for Safety Critical Systems.

Implementation of SMART process will meet the current Requirements Manual for Submarine Fly-By-Wire Ship Control System (NAVSEA T9044-AD-MAN-010) requirements.

The SMART process was designed as a 3-tier approach to ensure that all potential safety critical concerns are identified and that the best analysis method is used to verify each concern. The SMART methodology focuses on identifying the safety concerns. The safety concerns are generated from the following three areas: 1) critical functions which address the proper performance of safety critical areas of

software, 2) hazards within the system that can occur due to normal and abnormal conditions, and 3) best practices, which address general software practices for developing safe software. This 3-tier approach for determining safety concerns is much more comprehensive than using only one method. These concerns are then compiled into a list of Safety Requirements (SRs) which drive the safety analysis. The difference between the SMART methodology and other common SSSA methodologies is that the safety analysis is driven by the safety concerns instead of the analysis method chosen.

For the second issue, the redundancy of efforts by the IV&V and SSSA teams, the SMART process is flexible enough to take advantage of products generated by IV&V to avoid duplication of effort and reduce overall cost, while achieving the safety goals required. In times of critical development with less funding and resources available, SMART is a process that is repeatable, consistent and capable of covering the IV&V requirements. The SMART process has the ability to perform and/or pick up the IV&V analysis and testing requirements when required. The normal IV&V process does not perform the software safety analysis functions required.

The SMART approach is a model that can streamline and focus the work while consistently delivering superior products. The SMART methodology is flexible enough to address the safety concerns and provide consistent and repeatable process for the users. The SMART methodology has been used numerous times on Navy software programs by Computer Sciences Corporation (CSC). The SMART process is capable of being tailored at any stage of the development cycle, establishing the best approach in order to meet safety goals / criteria in a cost affective manner. CSC considers the SMART process to be a living process, which will be updated and refined as process improvements are identified. SMART consists of seven phases and three processes. Phases are one-time events for a given certification. Processes are on-going events that support the overall development effort.

1.3 ALTERNATIVE SAFETY ANALYSIS METHODS

Many software safety methods have been used as increasingly complex software-intensive systems have emerged. Among them are the following:

- Identifying and analyzing safety-critical Computer Software Units (CSUs);
- Identifying hazards and analyzing potential failure conditions;
- Identifying safety critical data and performing backward fault tree analysis; and
- Performing Failure Modes and Effect Criticality Analysis (FMECA).

All of these analysis approaches can be effective, but by performing only one analysis technique, the safety concerns may not be fully addressed. The disadvantage of following a single approach is that analyst tends to focus more on implementing the analysis method than truly addressing the critical safety issues.

1.3.1 CSU ANALYSIS ISSUES

CSU analysis and testing nominally occurs early in the development stage prior to Computer Software Configuration Item (CSCI) integration. 100 % path testing is performed on all identified safety critical CSUs. This ensures no unused or untested code is allowed to be in system. Two problems have arisen when this is the primary way to verify safety critical code. The first is the developer does not maintain the pedigree of the CSU by performing 100 % path testing for the CSU each time it is modified. This usually occurs due to lack of resources and schedule restraints. Second, this approach is good for early stages of development to eliminate potential problems but the safety analysis tend to concentrate on ensuring 100 % path testing and does not evaluate how the different CSUs tie together as a functional operation. Once the integrations of the CSUs occurs the safety critical software needs to be evaluated from a functional point of view to determine if any abnormal and/or normal operation can defeat any safety features creating a hazardous condition.

1.3.2 HAZARD ANALYSIS ISSUES

Based on the established safety criteria for the system, determine all of the potential hazards that could be created or contributed to by software, causing a safety criteria not being met based on current architecture/design. This is very similar as to what is done for a hardware safety evaluation. The main problem is that the safety analysts concentrate on the hazards that have been identified. They forget to look at the CSCI functional area design to ensure that the developer has not design and/or implemented code that could defeat a safety feature and/or introduce a new hazardous condition. This approach tends to have a lot of duplicate effort and over lap in evaluation that at times is difficult to manage. Also when the system is updated and/or modified, a change to a single Safety Critical CSU can impact multiple hazards requiring them to be re-evaluated.

1.3.3 BACKWARD FAULT TREE ANALYSIS ISSUES

If backward fault tree analysis is the primary approach it is normally performed late in the development process. It is a good method to determine any potential failure conditions that could occur that would cause safety critical data to be modified leading to a hazardous condition. The problem is this approach is used late in the system design and implementation which causes any findings to be costly to fix.

1.3.4 PERFORM FMECA ANALYSIS

This analysis tends to look at hardware failures and the effects it could have on the software. What is lacking is the analysis required to determine how the software could fail, under both normal and abnormal conditions, and determine overall safety affect on the system. The SMART process can be a key contributor to a FMECA evaluation by helping identify potential software failure conditions and determining overall safety effect. The end goal of a software FMECA is the same as the end goal of the SMART process evaluation.

1.4 DOCUMENT OUTLINE

This document is structured as follows.

Section 1 – Introduction

Section 2 – Software Safety Overview

Section 3 – SMART Overview

Section 4 – Process 1 - Software Safety Management

Section 5 – Process 2 – Software Safety Process/Technology Improvement

Section 6 – Process 3 – Software Safety Maintenance and Support

Section 7 – Phase I Software Safety System Concepts/Engineering

Section 8 – Phase II Software Safety Requirements Analysis

Section 9 – Phase III Software Safety Design Verification

Section 10 – Phase IV Software Safety Code Analysis

Section 11 – Phase V Software Safety Test

Section 12 – Phase VI Software Safety Evaluation

Section 13 – Phase VII Software Safety Process Review and Documentation

Appendix A – Abbreviations and Acronyms

Appendix B – System Safety Best Practices list

Appendix C – Software Design / Coding Guideline Options

Appendix D – References

SMART describes the safety engineering process to be used for system software safety analysis; it provides a methodology for documenting and managing safety related risks.

2. SOFTWARE SAFETY GOALS/DEFINITIONS

2.1 SOFTWARE SAFETY GOALS

As digital systems are increasingly being used to monitor and/or control complex, time critical physical processes or mechanical devices, the issue of software safety is becoming much more critical element in the development process. Unlike system hardware, software does not fail due to wear-out or damage. Software fails due to flaws in the design and/or implementation as well as its inability to handle Abnormal Conditions and Events (ACEs). In order to understand any software safety analysis methodology, there has to be a clear definition of what software safety is. Software by itself is neither safe or unsafe, rather the environment the software executes in determines how safe or unsafe the software is.

The purpose of Software Safety is to:

- 1) ***Reduce software failures which may result in death and/or injury to personnel.***
- 2) ***Reduce software failures which may result in system loss and/or damage.***

The goal of software safety is to prevent mishaps from occurring in the system that would result in either of the two conditions stated above.

SOFTWARE SAFETY DEFINITIONS

The following definitions must be understood to establish the basis for software safety analysis.

Mishap – an unplanned event or series of events resulting in death, injury, occupational illness, or damage to or loss of equipment, property, or the environment. ¹

Hazard – a condition that is a prerequisite to a mishap. ¹

Hazard Probability – the aggregate probability of the individual events occurring that create a specific hazard. ¹

Hazard Severity – an assessment of the worst credible mishap that could occur due to a specific hazard within its operational environment. ¹

A **mishap** is an actual physical event that is going to cause damage, such as valves leaking dangerous gases, weapons erroneously firing, etc. An example of a **hazard** is software not properly handling an erroneous signal, or data being overwritten, or software queues overflowing, etc. **A software hazard is an error in the software that could ultimately lead to a mishap.** The primary objective of software safety analysis is to determine all probable software hazards; then through analysis and testing, determine if the hazard probability and severity is low enough to consider the software “safe”. A Hazard Risk Index (HRI) is determined for every hazard detected in the software. The HRI is a combination of the hazard severity and probability. The HRI process is described in Phase VI (Section 12.4). If the

¹ MIL_STD 882C Definition

HRI values deem the software is not safe enough, software modifications need to be performed to lower the hazard probability or hazard severity to an acceptable level.

SHCM: The Software Hazard Criticality Matrix (SHCM), described in Phase II (Section 8.4.4), provides a measure of the potential risk of a software component. The SHCM can be used at the Computer Software Configuration Item (CSCI) level, or lower to a computer software component and/or Computer Software Unit (CSU) level. The SHCM value **should not** be used to determine the actual risk to a real identified hazard cause and/or contributed to by the software. If a real hazard is identified, through analysis and testing, a HRI value is generated to establish actual system safety risk.

HRI: The HRI, described in Phase VI (Section 12.4), quantifies the risks of actual hazardous conditions, identified in the system, which could lead to a mishap. This allows management to properly understand the amount of risk in the system due to actual problems that have been discovered through analysis and/or testing.

Software Requirement Specification (SRS): The SRS define the performance requirements that the system must achieve to be acceptable. The CSC Team has observed in the past that some of these requirements are identified as safety critical but do not contain any safety critical goals to be achieved.

Safety Requirements (SRs): SRs are used to direct safety analysts on what functions to evaluate, how the system functions should be evaluated, and what is safety-critical about the system functions. The safety analysts can be an integral part of the development team and support the development efforts from SRS requirements generation, through the design, code, and testing phases.

SOFTWARE SAFETY MISCONCEPTIONS

There are many misconceptions about what software safety is and why it needs to be performed. The following lists some other analyses and their differences.

Independent Verification & Validation (IV&V) vs. Safety

IV&V ensures that a system correctly performs those functions for which it was designed. Safety determines what type of failures could be expected to reasonably occur, how well the system handles those failures, and what the level of risk is, if it did occur. Safety deals with missing requirements, requirement conflicts, and/or unintended functions.

Security vs. Safety

Security deals with threats to privacy or national security. Safety deals with threats to life or property. Both safety and security impose requirements that can conflict with important mission requirements. However, security focuses on malicious actions, whereas safety is concerned with inadvertent actions.

Reliability vs. Safety

Reliability is usually defined as the probability that a system will perform its intended function for a specified period of time under a set of specified environmental conditions. Safety is the probability that conditions that can lead to a

mishap do not occur. The goal of safety and reliability analysis is to reduce failures. However, reliability is concerned with making a system failure free, whereas safety is concerned with making it mishap free.

Redundant Systems Misconception:

Redundant hardware systems are commonly installed to improve reliability and safety. However, hardware redundancy does not generally protect against software failures since redundant computers are usually running the same software. Common mode failures in the software can occur, causing major safety issues.

Commercial-off-the-shelf (COTS) Misconception:

COTS software can react differently depending upon the application environments. Safe use of COTS in one safety critical system application does not automatically mean that it can safely be applied to another safety critical application. Examples of COTS software causing a serious hazardous condition are well documented. A most recent incident with a ship control system involved a library data conversion routine that locked up a redundant system to the point that the only recovery was to turn the system off and reboot. When the control system was locked up the system was extremely vulnerable to a Category I mishap occurring.

Software Re-use Misconception:

Reused Software has similar issues as COTS.

3. SMART OVERVIEW

3.1 THE SMART PROCESS

SMART is a fully integrated approach for accomplishing all SSSA work and identifying areas where IV&V efforts may be applicable. The SMART process is versatile and is supportive to address all major formal system development milestones such as Preliminary Design Review (PDR), Critical Design Review (CDR) etc.

The SMART process can also be used to train software developers about safety issues and establish proven design and coding guideline practices for safe software development. Appendix C, Table C.1 provides a list of code design and coding guidelines. Use of the SMART process will reduce the initial safety analysis cost and provide long-term life cycle maintenance cost savings as well.

3.2 SMART APPROACH

The SMART approach is comprised of seven phases and three major processes. Figure 3-1 illustrates an overview of the SMART methodology. The seven phases are performed sequentially starting phase 1, while the 3 processes are done throughout the SSSA effort providing direction to the phases.

The major processes are:

1. Software Safety Management,
2. Software Safety Process/Technology Improvement,
3. Software Safety Maintenance & Support

The seven phases are:

- I. Software Safety System Concepts/Engineering,
- II. Software Safety Requirements Analysis,
- III. Software Safety Design Verification,
- IV. Software Safety Code Analysis,
- V. Software Safety Test,
- VI. Software Safety Evaluation, and
- VII. Software Safety Process Review and Documentation.

The seven phases are the heart of the methodology. This is where the software is actually evaluated to identify hazards and determine the level of risk that the system software presents to overall system safety and contributing to potential mishaps, The three processes are required to ensure coordination between software safety analyst, developer, end-user and system procurement group. In addition the processes ensure system software safety criteria and goals have been agreed to and being properly addressed. The processes also helps determine work priority, direction and approach to meet safety criteria and goals.

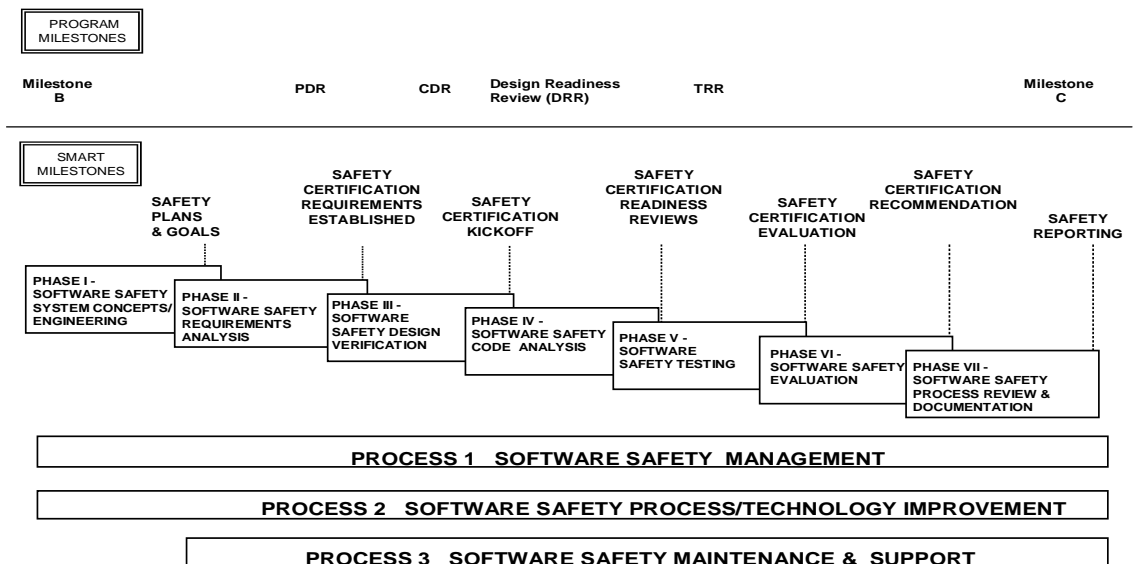


Figure 3-1: SMART Processes and Phases

The SMART approach provides the most cost savings and efficiencies when implemented during the System Concept phase. The reason for this is that the System Safety Criteria / goals are established and defined early on and the overall system architecture and design can take the safety issues into consideration. It also allows the software developer and his staff to be educated as to what the safety concerns are, what safety best practices are, why they need to be followed as part of the development process, along with ensuring the system and software requirements properly reflect the safety criteria / goals. This also allows the safety analysis team to establish communication with the development team and other support groups. This provides insight by the developer team as to what are safety issues and concerns to be aware of and insight by the software safety team as to how the system is being design and implemented to be able to help identify potential safety areas of concerns as early as possible.

It has been demonstrated that the SMART approach can be initiated at any stage of the development cycle. The key to SMART is the establishment of an agreed to set of System Safety Criteria / goals. Once the Safety Criteria has been established then it is a matter of mapping the Safety Criteria to the system software requirements and functional areas, and then determining the best approach on how to evaluate the system.

Worse case scenario to perform system software safety analysis has been when the system has already gone through system acceptance testing. It was determine during final system certification evaluation that the software needed to go through a software safety evaluation due to the control the system software had on safety critical components. Following the SMART approach the first order of business was to establish the System Software Safety Criteria and map the safety criteria to the software requirements and functional areas. Once the safety functional areas had been determine the SMART approach then evaluated the best way to verify the safety critical functional areas. For this specific case, it was decide that the best approach was to identify safety critical signals and perform a backward fault tree analysis. This identified a number of system design and development practices changes in order to ensure system safety over the life of the system. Weaknesses and problems found in the software were costly to updated and re-test at this stage of the game.

Most of the time system software safety is not addressed or brought on board until after the system architecture has been defined and the software requirements have been established. The SMART approach is very adaptable either as a standalone program or working with the developer and/or Verification and Validation (V&V) group. Again the first step in the SMART approach is the establishment of the System Safety Criteria / goals. Second step is mapping the System Safety Criteria to the software functional areas and related software requirements followed by establishing Safety Requirements (SRs). The SMART approach then determines the best way to verify the SRs. The primary issue or short coming at starting the software safety evaluation at this point in the development phase is that the software related requirements reflect only performance issues and do not address any safety concerns / goals. This requires additional effort by the safety evaluation team to ensure the system is analyzed and tested from a safety perspective. On past programs this has been achieved by the safety group performing a detailed

safety analysis on the safety critical design and code implementation. Based on finding from this analysis a separate group of safety related test are developed and executed by the SSSA team or by having the ability to influence the developer and/or V&V groups acceptance test by adding or modifying plan test to reflect potential safety concerns identified.

A key cost savings feature with the SMART approach is the ability to work jointly with an Independent Verification and Validation (IV&V) group or developer's internal V&V group to avoid duplication of effort. This is accomplished by coordinating the evaluation process between the safety and V&V group by identifying the safety critical functional areas of the software and having the safety group evaluate those sections of material while the IV&V group concentrates on the non-safety critical areas. In addition SMART approach tries to influence the developers, IV&V and/or V&V testing from a safety perspective to avoid duplication of effort. The goal of the SMART approach is to work within the current development structure to avoid duplication of effort and need for critical resources. In the discussion of the different SMART phases, the areas of effort that could be performed jointly have been identified. The key issue is SMART can perform the joint task independently but the V&V groups process do not take into account the safety related task required.

4. PROCESS 1 – SOFTWARE SAFETY MANAGEMENT

4.1 SSSA MANAGEMENT TASKS

The SSSA Technical Management process is a fully integrated style of management to identify how best to work within current or planned program organizational structures. This process involves review, evaluation, and approval of all tasking and direction from the customer, as well as review and approval of all deliverables to the customer. SSSA management also establishes interfaces with the developer and other support groups with the customer's approval. Management will organize staff and work to meet the customer needs, respond effectively to Statement of Work (SOW), and meet schedule requirements.

SSSA Management is responsible for the following tasks:

- Review, evaluation and approval of all safety-related tasking and direction
- Review and approval of all safety-related deliverables
- Organize work to meet safety needs and respond to safety-related schedule requirements
- Set up safety process to interface with developer and other related evaluation groups (i.e. IV&V) to work with them to be efficient in overall evaluation and testing of system and reduce duplication of effort.
- Work with customer to effectively focus safety analysis effort to assure it meets customer needs in a timely and cost effective manner.
- Identify safety resource need and acquire

- Set up safety program risk management process and work issues in a timely manner
- Plan and focus work based on key safety milestones/schedules, resources availability, and review material availability
- Set up Metrics to monitor SSSA effort
- Set up Training for safety analysis personnel and if desired development personnel
- Ensure Configuration Management (CM) process is in place and address safety issues
- Ensure Quality Assurance (QA) process is in place and address safety issues

4.2 SSSA DOCUMENTATION

The production of SSSA documentation is overseen by SSSA management. The SSSA documentation referenced in this document encompasses possible SSSA documentation. Depending on the size and level of effort, management would coordinate with the customer to determine what documentation is necessary for each project. Table 4-1 lists possible SSSA deliverable documentation. Documentation that is performed jointly with the IV&V team is not listed.

SSSA Formal Documentation		
Phase	Document Name	Preliminary or Final
I	Software Safety Plan	Final
I	PIDs/SSDD Verification Report	Final
I	High-Level System Safety Requirements	Preliminary
II	Certification Kick-off Agenda	Final
III	Positive Measures	Preliminary
III	SR Document	Preliminary
IV	Verification Requirement (VR) Code Analysis Results	Final
V	Safety Test Report	Final
VI	Hazard Assessments	Preliminary
VI	Certification Recommendation Letter or	Final
	Safety Analysis Report (SAR)	
VII	SR Document	Final
VII	Final Report	Final
VII	Software Analysis Folders	Final
VII	SSSA Final Report	Final
VII	Final SSSA Test Procedures	Final

Table 4-1 SSSA Formal Documentation

4.3 SSSA PERSONNEL REQUIREMENTS

The SSSA Management is tasked with filling all personnel positions. The amount of personnel required varies depending on the size and complexity of the software being evaluated and the level of effort defined by the established and agreed to safety goals. An SSSA effort consists of three primary positions: SSSA Manager, System/Software Safety Engineer(s), and Software Safety Analyst(s).

SSSA Manager – The SSSA Manager oversees the overall SSSA schedule and oversees all SSSA activities. The SSSA Manager must have management experience as well as experience in software (or hardware) safety analysis. The SSSA manager requires good written, verbal communication and presentation skills as well as being capable of planning, schedule and supervising safety group activities.

System/Software Safety Engineer - Depending on the number of software components in an SSSA effort, there can be one or more System/Software Safety Engineers. The engineers are the technical leaders over an SSSA effort on a software component. The System/Software Safety Engineers provide technical assistance for the development and conduct of engineering analyses and oversee the definition of safety requirements.

Software Safety Analyst – The number of Software Safety Analyst required will vary depending on the size and complexity of the software under review. The Software Safety Analyst, as a minimum, needs training in performing software safety analysis and an understanding of the relations between software and hardware safety risk. The Software Safety Analysts perform the requirements, design, code and testing analyses.

5. PROCESS 2 – SOFTWARE SAFETY PROCESS/TECHNOLOGY IMPROVEMENT

Our Software Safety Process/Technology Improvement provides a means of investigating and evaluating new technology. Both turn-key off-the-shelf tools and new development tools are considered for safety evaluation. This process keeps management abreast of the latest technology on the market and ensures that current technology is being used to provide the customer with the best support possible. The safety process is also reviewed to determine what adjustments and changes are required to effectively analyse and evaluate the targeted system under review.

To achieve software safety process improvement, emphasis is placed on the application, research and technology transfer or promising new software/hardware technologies. The goal is to identify, investigate, and development new or improved software tools, techniques, and methodologies. In addition, integration of new or improved hardware is also evaluated.

Technology improvement is motivated by both internal (company) and external (customer) forces. It is based on customer needs and lessons learned from Phase VII of the SMART process. Continued evaluation and investigation of new technology in the areas of processes, software tools, hardware tools, etc., along

with evaluation of current efforts and goals help identify candidate process improvements. The customer is informed of both the benefits and potential risks involved in implementing changes.

The following list summarizes the goals:

- Process Improvement
 - Customer Needs
 - Company Needs
 - Increase Efficiency, reduce time and costs
 - Improve Products delivered
- Technology Improvement
 - Evaluate and/or develop new or improved software tools
 - Investigate new or improved hardware
- Training Improvement
 - Improved training techniques and training schedules

6. PROCESS 3 - SOFTWARE SAFETY MAINTENANCE AND SUPPORT

The purpose of this phase is to provide support for routine tasks that span both development and life cycle support efforts. For the Safety team, these tasks include Safety Working Group (SWG) Meeting Support, Software Safety Technical Review Boards (TRBs), Software Configuration Control Board (SCCB) and general technical support.

The Safety Development, Maintenance and Support Process tasks are the following:

- Perform SCCB Technical Assistance
- Perform Special Technical Assistance and Research Support
- Perform Safety Working Group Meeting Technical Assistance
- Perform Safety Technical Review Panel Technical Assistance
- Perform Hazard Assessment (HA) Analysis
- Review all outside generated software problem trouble reports to determine if they affect system safety and ensure safety issues have been properly identified, evaluated and are being tracked. All identified safety issues will have HA report generated (see Phase VI for HA generation.)

7. PHASE I – SOFTWARE SAFETY SYSTEM CONCEPTS/ENGINEERING

7.1 PURPOSE

The purpose of this phase is to determine what the high-level safety criteria are for the system. During this phase, new/upgraded capabilities and draft top level documentation are reviewed. . A Systems Critical Functions List, System Hazard List and System Best Practices List are created which document all high-level system safety concerns that may have software impact within the system. This phase is usually performed during the system architecture development and before the final software requirements allocation has been completed. These high-level system concerns are compiled into a High-Level System Safety Requirements (HLSSR) list.

Phase I will determine what the high-level safety criteria are for the system to be developed. For some systems, the safety criteria have already been established, but for others the criteria needs to be defined during this phase. During Phase I, the customer provides technical direction about the system to determine the highest safety risk factors. The customer has final say in determining acceptable level of risk for the system

7.2 INPUTS/OUTPUTS

There are two categories of tasks. The first category is joint IV&V-Safety Tasks. These tasks are performed jointly between the IV&V and Safety teams to reduce redundancy and improve efficiency. These tasks are shown in blue in Figure 7-1. The second category is safety specific tasks that are performed solely by the Safety Team are shown in red in Figure 7-1.

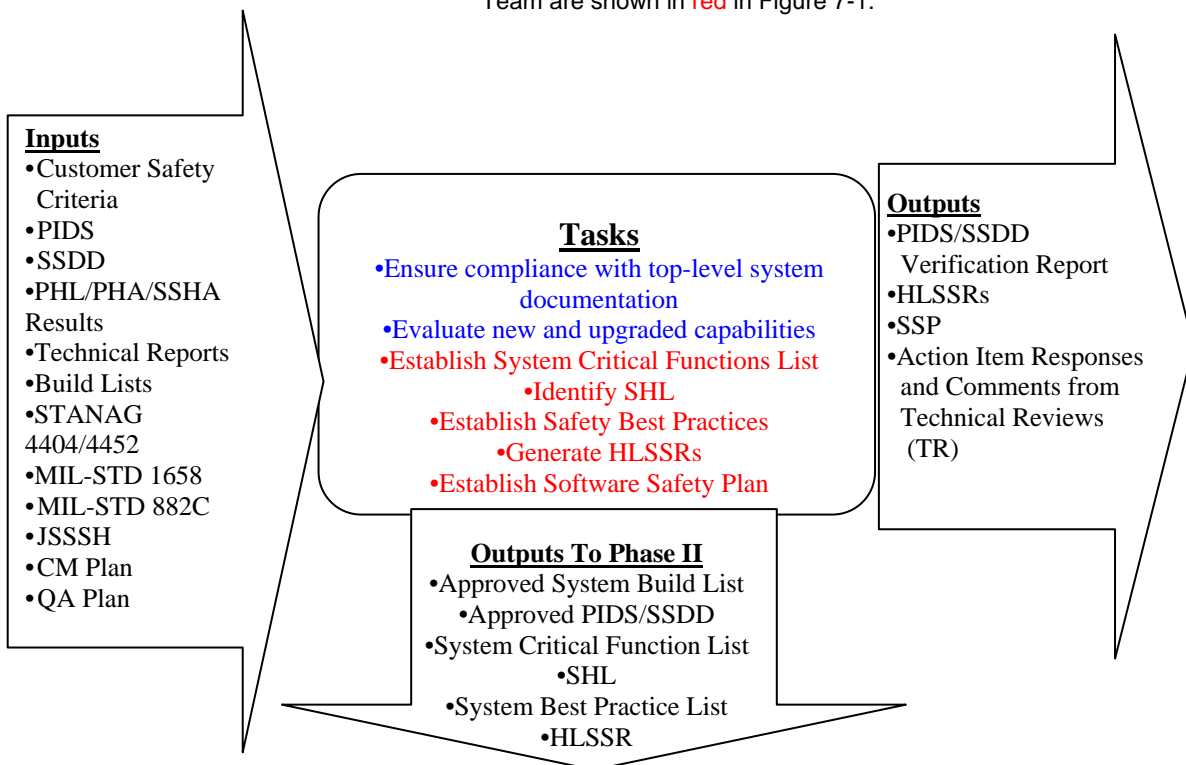


Figure 7-1: Phase I – Software Safety Systems Concepts/Engineering Phase

Paragraph 7.3 describes the Joint IV&V Safety Tasks in detail. Paragraph 7.4 describes the Safety Specific Tasks in detail. Figure 7-1 shows a graphical overview of Phase I.

7.3. PHASE I – JOINT IV&V/SAFETY TASKS

7.3.1 ENSURE COMPLIANCE WITH TOP-LEVEL DOCUMENTATION

The Prime Item Development Specification/ Software System Definition Document (PIDS/SSDD) are reviewed whether or not the document is new or an updated version of a previously delivered document. If the document is an updated version, the analyst verifies that all previous comments are implemented correctly and provides concurrence or any technical discrepancies via the PIDS/SSDD Verification Report. Technical assistance, at local review board, may or may not be necessary, depending on the volume of technical comments against the document.

If the document for review is a new delivery, the Safety and IV&V Teams convene to discuss the “big picture” and ensure that team members have a thorough technical understanding of the new program prior to beginning technical review. As before, discrepancies are noted in verification reports. Review board meetings are supported as required.

The joint IV&V/Safety team also ensures that CM and QA processes are properly in place and documented in a CM Plan and QA Plan. Both the QA Plan and the CM Plan must be in compliance with applicable standards.

7.3.2 EVALUATE NEW & UPGRADED SYSTEM CAPABILITIES

Proposed build lists for an upgrade to the targeted system are defined/established by the customer. The Safety and IV&V Teams review the list against problem reports submitted for the previous version of the program via the Software Change Control Board (SCCB).

When modifications or updates are made to the targeted system, impacts to the interfaces and current software systems are defined. Feasibility studies and technical reports document the impacts to interfaces between the different CSCIs. In some cases, impacts to hardware/firmware are defined.

As part of this task, technical review meetings may be attended and action items and comments must be documented as required.

7.4 PHASE I – SAFETY DETAILED TASKS

The safety detailed tasks for the Systems Concepts and Planning Phase consists of the establishment of the System Critical Function List (SCFL), the System Hazard List (SHL) and the System Safety Best Practices.

The SCFL and the SHL are lists of system level functions and events that have been determined to be safety critical. These lists are derived from top level requirements, and Preliminary Hazard Analysis (PHA) results. MIL-STD-882C, STANAG 4404, MIL-STD-1658 (OS) and Joint Software System Safety Handbook (JSSSH) are also used for guidance. Safety goals can also be created by customer input. The customer usually has ideas of safety concerns due to their extensive knowledge of the system. This customer input is referred to as *Customer Safety*

Criteria. The System Safety Best Practices are general software development and design guidelines that apply to all safety critical functions and events. The specific details on how to generate these lists is contained in paragraphs 7.4.1, 7.4.2, and 7.4.3.

7.4.1 ESTABLISH SYSTEM CRITICAL FUNCTIONS

Critical Functions are system functions whose correct performance is essential to the safe operation of the system. System Critical Functions are identified by the Customer Safety Criteria, generic software safety requirements and the high-level system documentation such as the PIDS. Generic software safety requirements are derived from sets of requirements and best practices used in different programs and environments to solve common software safety problems. Generic software safety requirements capture these lessons learned and provide a valuable resource for developers. A list of generic software safety requirements is contained in Table 7-1 below. Requirements are reviewed to determine safety critical involvement using guidelines from MIL-STD-882C, STANAG 4404, MIL-STD-1658 (OS) and the JSSSH. The SCFL addresses software, however, if the software is not yet developed, the System Critical Function List should include any critical functions that possibly could impact software. The System Critical Function List is used in Phase II of SMART to determine computer program (CP)-specific critical functions for each piece of software. Examples of Critical Functions are: Sensor data processing, Operator mode control processing, and Missile Launch processing. The proper performance of each of these functions is essential to the safety of the system.

Table 7-1: Generic Safety Requirements

Generic Safety Requirements	
1.	Software shall perform automatic failure detection, isolation, and recovery for identified safety critical functions.
2.	Automatic recovery actions taken shall be reported to the crew, operator, or controlling executive. There shall be no necessary response from crew or ground operators to proceed with the recovery action.
3.	Override commands shall require multiple operator actions.
4.	Prerequisite conditions (e.g., correct mode, correct configuration, component availability, proper sequence, and parameters in range) for the safe executive of identified safety critical events shall be met before execution.
5.	Software shall provide an independent and unique command to control each software-critical event.
6.	Software shall provide error handling to support safety critical functions.
7.	Software shall provide caution and warning status to the operators or the controlling executive.
8.	Software shall provide for operator forced termination of any automatic safing, isolation, or switchover functions.
9.	Software shall provide fault containment mechanisms to prevent error propagation across interfaces.
10.	Software termination shall result in a safe system state.

7.4.2 IDENTIFY SYSTEM HAZARD LIST

For most systems, the Preliminary Hazard List (PHL) and PHA results are already generated prior to the SSSA effort. The PHL is a list of possible system hazards. Outputs of a PHA include system hazard descriptions, hazard causes, hazard effects, and designed control features for each hazard. On some systems, a Subsystem Hazard Analysis (SSHA) is also performed on individual subsystems of the total system. From the PHL, PHA, and SSHA results, the SHL is created. The SHL is a list of hazards that have software involvement. If the software is not yet developed, any hazards that have a possibility of software involvement should be included in the SHL. This list is used in Phase II of SMART to determine computer program-specific hazards. Areas of consideration include functional failures, timing errors, operator errors, or inadvertent functions. There can be some redundancy between System Critical Functions and System Hazards, but the primary difference is that System Critical Functions control a planned safety critical function while System Hazards respond to and/or control an unplanned hazardous condition and keep a potential mishap from occurring. Examples of System Hazards are: Sensor overtemperature condition, Restrained firing, and Missile launch with no firing delay.

7.4.3 IDENTIFY SYSTEM BEST PRACTICES

The System Best Practices are developed from MIL-STD-882C, STANAG 4404, JSSSH and past experience. These objectives are used in Phase II of SMART to determine computer program-specific required practices for software development and design. The Best Practices are concerned with the development processes and general characteristics of the software rather than implementation of specific safety critical functions. Their purpose is to ensure the production of safe maintainable software and the avoidance of unanticipated hazards caused by unintended erroneous software behaviour. Examples of System Best Practices are: 1) Constant data and executable code stored in write-protected memory, 2) Initialize all data before use, and 3) Input data validation. Appendix B contains a list of established system best practices safety requirements.

7.4.4 HLSSRS

HLSSRs provide the initial groundwork for safety design analysis, code analysis and testing in phases III thru V. HLSSRs list all the safety concerns in the system. The HLSSRs document the specific concerns around each of the System Critical Functions, System Hazards, and System Best Practices. During the Requirements Phase (Phase II), the HLSSRs are refined into Safety Requirements (SRs) for each Computer Software Configuration Item (CSCI). SRs are then verified during code analysis and testing.

7.4.5 ESTABLISH SYSTEM SAFETY PLAN

Some customers require the software safety approach and schedule be documented in a System Safety Plan (SSP). An SSP, which can also be called a Software Safety Program Plan (SSPP) or a Software Safety Evaluation Plan (SSEP), is generated during Phase I if required. An SSP formally documents the planned software safety analysis. The SMART methodology would be the basis for documenting the methodology in the SSP. A schedule or timeline of the planned

analysis is also included in the SSP. The document format would be based on industry or government standards.

8. PHASE II – SOFTWARE SAFETY REQUIREMENTS ANALYSIS

8.1 PURPOSE

The purpose of this phase is to produce software requirements that are clear, concise, and testable. During this phase, system specifications, interface documents and software requirement documents are reviewed for safety impacts. The primary goal of this phase is to ensure that the safety criteria are properly addressed in the requirements documents. In addition, Safety Requirements (SRs) are generated from the HLSSR list. SRs document all the safety concerns for a computer program. SRs drive the SSSA effort during code analysis and testing phases. SHCM rankings are established. SHCM rankings identify levels of safety analysis required for specific CSCIs or software functional areas.

8.2 INPUTS/OUTPUTS/TASK OVERVIEW

There are two categories of tasks. The first category is joint IV&V-Safety Tasks. These tasks are performed jointly between the IV&V and Safety teams to reduce redundancy and improve efficiency. These tasks are shown in **blue** in Figure 8-1. The second category is safety specific tasks that are performed solely by the Safety Team are shown in **red** in Figure 8-1.

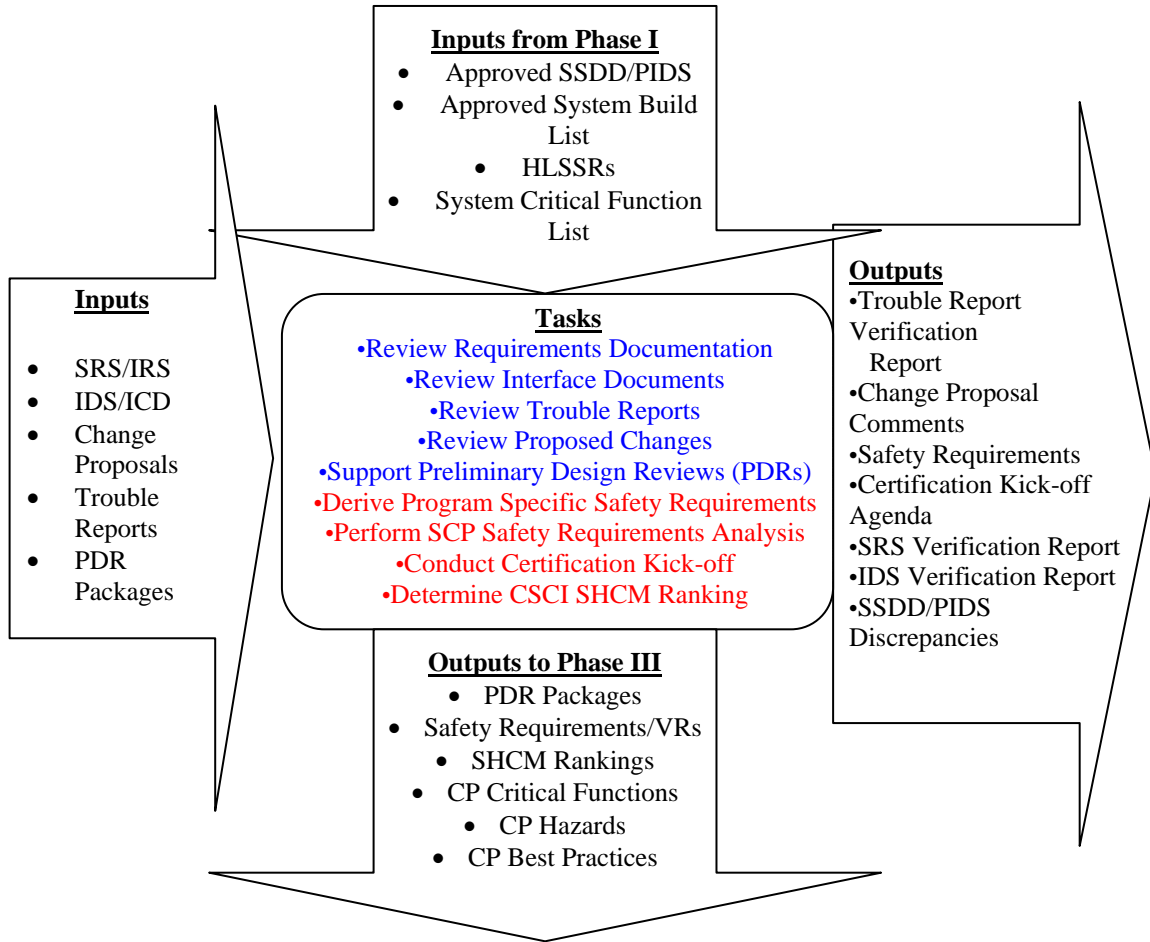


Figure 8-1: Phase II – Software Safety Requirements Analysis Phase

Paragraph 8.3 describes the Joint IV&V-Safety Tasks in detail. Paragraph 8.4 describes the Safety Specific Tasks in detail. Figure 8-1 shows a graphical overview of Phase II.

8.3 PHASE II – JOINT IV&V-SAFETY TASKS

By combining duplicate efforts into a single unified approach, the SMART process is able to efficiently perform the requirements review with minimal redundancy. Have the Safety and IV&V Team perform coordinated software requirements and interface requirements reviews and problem report investigations.

8.3.1 REVIEW REQUIREMENTS DOCUMENTS

When a SRS is received, a joint IV&V/Safety Team is assembled. One person is assigned to coordinate the document review process and interface with the customer. This joint effort makes the best use of each team member's expertise and skill.

If a new document is received as a result of a new baseline or project, a kick-off meeting is held with the customer to start the review task with a single direction and goal. The SRS is reviewed to ensure that it is in compliance with the SSDD and/or

PIDS. Any discrepancies found against the SSDD or PIDS are reported back to the customer. All SRS comments discovered by the joint team are incorporated into the SRS Verification Report, which is delivered to the customer.

A revised SRS may be received as a result of a baseline upgrade. When this occurs, a joint team is again assembled. The Safety Team Members verify that all Software Change Proposals (SCPs) with safety impact are implemented correctly. The IV&V Team verifies that all remaining SCPs are implemented correctly. When this task is completed, a SRS Verification Report is prepared either reporting the document is complete or listing all discrepancies found.

8.3.2 REVIEW INTERFACE DOCUMENTS

When a new Interface Design Specification (IDS) or Interface Control Document (ICD) is received, a review team is assembled from both the IV&V and Safety teams. Comments from all reviewers are reported to the customer in an IDS/ICD Verification Report. An Interface Working Group (IWG) meeting dispositions all comments against the document. This task of document revision and comment review is repeated until there are no comments reported and the document is ready for approval.

After a document is approved, copies are received by all organizations within the community. All comments are reviewed to ensure they were incorporated correctly in the approved document. After the document review is completed, an IDS/ICD Verification Report is delivered to the customer either stating the document is complete or listing all discrepancies found.

If a new revision or change set is received, all Interface Change Requests (ICRs) are reviewed. The updated IDS is reviewed to ensure each ICR is correctly implemented into the IDS. After the verification is completed, an IDS/ICD Verification Report is delivered to the customer either stating the revision or change set is complete or listing all discrepancies found.

8.3.3 REVIEW TROUBLE REPORTS

The first step of reviewing Software Trouble Reports (STRs) is to determine if the problem is valid. This may involve new tests or code analysis. If the problem is valid, the next step is to determine if the problem reported is redundant to an already existing Trouble Report. The STR cover sheet information is verified as to priority, Class, Safety impact, and documents/software modules affected. An SSSA Team member should evaluate if the STR has a safety impact. If the STR has safety implications, the trouble report needs to document the safety concern and the SSSA team will have to further evaluate safety impacts through a Hazard Assessment. The Hazard Assessment task is documented in Section 12.0 (Phase VI). The recommendation is analysed to determine if there is a better or alternate solution to the problem reported. When the review task is completed, a Trouble Report Verification Report containing all comments is sent to the customer.

The STR Verification Reports are discussed at the SCCB meetings as to whether the problem should be fixed or accepted as is. Proposed changes are documented in Software Change Proposals.

8.3.4 REVIEW PROPOSED SOFTWARE CHANGES

Upon receipt of the SCCB agenda containing Software Change Proposals (SCPs), the SCPs reviews are divided between the IV&V and Safety Teams. The joint team would provide comments regarding any SCPs on the approved Build List. Only requirement changes associated with the SCPs are reviewed in this phase. Design and coding changes are reviewed in phases III and IV.

8.3.5 SUPPORT PRELIMINARY DESIGN REVIEWS

After the requirements are complete, the Software Designer/Developer schedules a preliminary PDR to allow the joint IV&V/Safety team to review the preliminary design of the software. The PDR is held to gain approval to initiate the design of the software. The Joint IV&V/Safety team provides technical expertise and comments to the Software Designer/Developer and ensures the design is an accurate, complete and safe design of the approved requirements.

8.4 PHASE II – SAFETY SPECIFIC TASKS

The purpose of this phase is to produce software requirements that are clear, concise, testable, and address safety concerns. This effort consists of the derivation of Safety and Verification Requirements (SRs and VRs) and the evaluation of changes to the targeted system computer programs, documented in Software Change Proposals (SCPs), to determine which Safety Requirements are impacted by the changes. Requirements Analysis culminates in a Certification Kick-off Meeting for each computer program undergoing SSSA that establishes the SRs and VRs to be verified for each safety related change and the methods of verification to be used (requirements analysis, design and code analysis, or testing).

8.4.1 DERIVE PROGRAM SPECIFIC SAFETY REQUIREMENTS

Safety Requirements are developed for each computer program within the system. For each computer program, the System Critical Functions, System Hazards, and System Best Practices are examined to determine which ones are applicable to a specific computer program. This will result in specific list of program specific Critical Function, Hazards, and Best Practices for each computer program. After the list of program specific Critical Functions, Hazards, and Best Practices are generated, a Safety Requirement is written for each of these concerns using the HLSSR list. Under each SR, numerous VR can be written to list the detailed concerns under each SR along with possible verification methods. Figure 8-2 shows a graphical view of how the Program Specific Safety Requirements are derived.

8.4.1.1 Computer Program Critical Functions (CPCFs)

CPCFs are developed by taking the System-Level Critical Functions and determining which ones apply to this specific program. After determining a subset list of System Level Critical Functions that apply, that subset list is expanded and further defined into more specific CPCFs for that specific CSCI. For each computer program the software specifications associated with the program are used along with the Critical Function List derived in Phase I to derive Critical Functions. Safety Requirements are then derived for each Critical Function to define the specific

safety concerns for each Critical Function as well as VR which specifies a verification method required to ensure all safety concerns are addressed. Table 8-1 is an example of the Critical Function process.

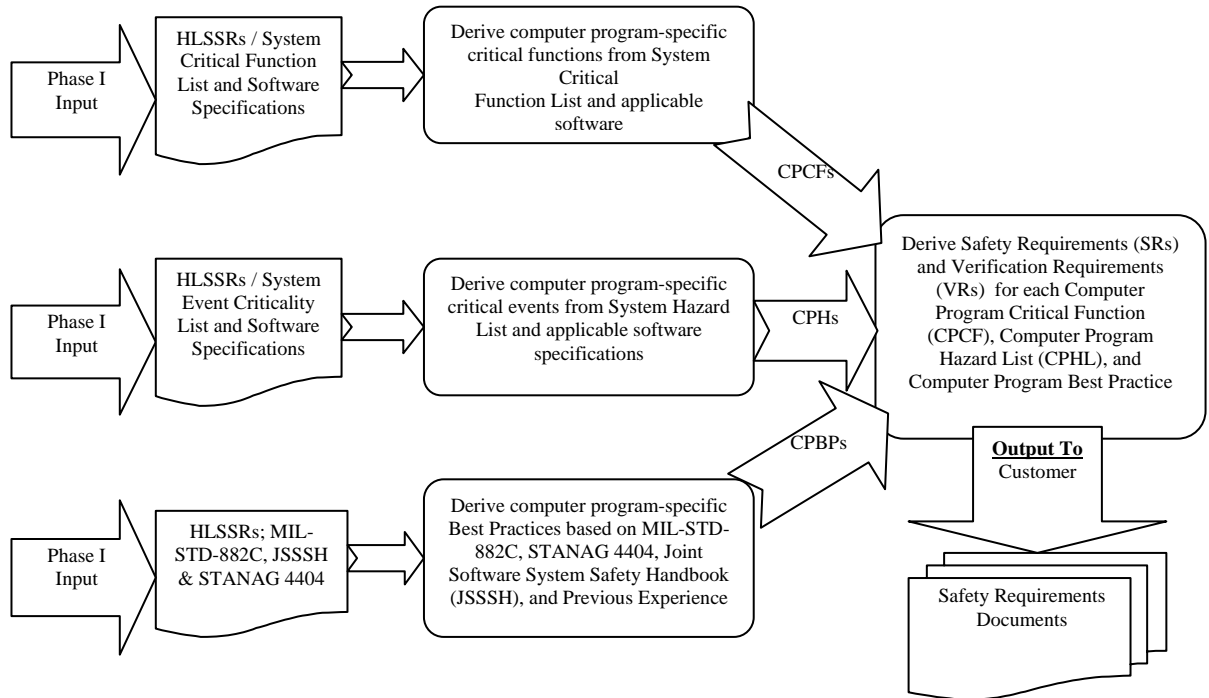


Figure 8-2: Derive Safety Requirements

System Level Critical Functions	Computer Program A Critical Functions	Computer Program B Critical Functions
Critical Command Validation	Depth Order Validation Depth Rate Order Validation	Course Order Validation Speed Order Validation Ordered Trim Validation
Program Load / Initialization	A Program Load / Initialization Adaptation / History Load	B Program Load/Initialization
Missile Selection	N/A	Tomahawk Missile Selection VLA Missile Selection SM-2 Missile Selection

Table 8-1: System Level and Computer Program Critical Functions

8.4.1.2 Computer Program Hazards

Software Hazards are derived from the SHL and the software specifications to ensure that the software will not contribute to the event and will respond appropriately to mitigate the event should it occur. Safety Requirements are then derived for each software hazard to define the verification required to certify the computer program. The analysis and testing driven by these SRs is essentially a bottom up effort to verify that there are no missing safety features or unintended interactions introduced by the computer program design or implementation that could lead to the credible possibility of identified mishaps. Along with identifying Computer Program Hazards, Positive Measures for those hazards can also be identified. Positive Measures are system design features that enhance the system level of safety. Positive Measures identify, control and/or eliminate the potential risk that a hazardous condition will occur leading to a mishap. Positive Measures can be documented through a Positive Measures Table or through the Verification Requirements (VRs) that are written for each SR. Table 8-2 is an example of System Level and Computer Program Hazards.

System Level Hazards	Computer Program A Hazards	Computer Program B Hazards
Launch with Closed Cell Hatch	Launch with Closed Cell Hatch	N/A
Control Surface Jam	Bow Planes Jam Stern Planes Jam	N/A
Overtemperature Condition	N/A	Sensor A Overtemperature Sensor B Overtemperature Sensor C Overtemperature

Table 8-2: System Level and Computer Program Hazards

8.4.1.3 Computer Program Best Practices

MIL-STD-882C, STANAG 4404, JSSSH, and past experience are used to define the Safety Best Practices SRs. The Safety Best Practices SRs establish software engineering and programming practices that are required in the development of safety critical software. The Safety Best Practices SRs are concerned with the development process and general characteristics of the software rather than implementation of specific safety critical functions in the software application and are intended to reduce the chance of specification, design, or programming errors in the software that could lead to erroneous performance. Their purpose is to ensure the production of safe maintainable software and the avoidance of unanticipated hazards caused by unintended erroneous software behaviour. The list of Computer Program Best Practices may be determined by customer input. With some of the more stringent Best Practices, the customer may determine that it is not cost-effective to require their implementation. Table 8-3 shows some System

Level and Computer Program Best Practices. A list of established Best Practices SRs is contained in Appendix B.

<i>System Level Best Practices</i>	<i>Computer Program A Best Practices</i>	<i>Computer Program B Best Practices</i>
No Unnecessary Code	No Unnecessary Code	N/A
Initialize All Data Before Use	Initialize All Data Before Use	Initialize All Data Before Use
Constant Data and Executable Code Stored in Write-Protected Memory	N/A (due to customer input)	Constant Data and Executable Code Stored in Write-Protected Memory

Table 8-3: System Level and Program Best Practices

8.4.2 PERFORM SCP SAFETY REQUIREMENTS ANALYSIS

SCP Safety Requirements Analysis is performed to determine the impact of each software change going into a computer program build on the existing Safety Requirements for that computer program. Those SRs and VRs impacted by an SCP are identified and a preliminary recommendation on how to perform the verification for each impacted VR is documented.

Each SCP needs a quick review by the Safety team to ensure that the changes do not have adverse impacts on the overall safety of the system. For many SCPs this would be a quick cursory review of the SCP.

8.4.3 CONDUCT CERTIFICATION KICK-OFF

Once all SCPs have been analyzed for impact to the SRs and VRs, a Certification Kick-off Meeting with the customer is held to present those SRs and VRs that will need to be verified to certify a particular computer program build. A project schedule and plan is also presented at the meeting, outlining what tasks are required to be done, when tasks need to be done, recommended methods for VR verification, and who will be responsible for each major task. The plan and schedule also addresses what deliverables are to be made to the government and when they are planned.

8.4.4 DETERMINE SOFTWARE HAZARD CRITICALITY MATRIX (SHCM) RANKING

The purpose of the software hazard risk assessment is to determine which software functions analysis should focus on. The SHCM is used to determine an index for a particular software functional area, is determined by combining the highest possible hazard severity for an area of software with the Software Control Category (SCC). The SCC documents the level of software functional autonomy or lack of user interactive control. The SHCM index represents a level of risk to help determine the criticality of a particular functional area. Determining the SHCM index is done

prior to analysis to determine what software functions should have priority. There are four SHCM index categories: *High Risk*, *Medium Risk*, *Moderate Risk*, and *Low Risk*. *High Risk* software requires significant analysis and testing resources. *Medium Risk* software requires detailed requirements, design, and code analysis, as well as sufficient testing to assure safety implementation. *Moderate Risk* software requires high-level requirements, design and code analysis, as well as testing as need. *Low Risk* software is acceptable with minimal requirements, design, and code analysis.

The SHCM (Table 8-4), assists the Safety Team in determining the level of safety analysis required for a particular CSCl. In order to determine an SHCM, a two-step process has been implemented that evaluates the degree of control the software exercises over a potential safety critical function/hazard and the potential related hazard severity. The following definitions, based on MIL-STD-882C, are used to establish degree of software control category and potential hazard severity:

- a. Software Control Categories
 - 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 failure to prevent an event leads directly to a hazard's occurrence.
 - 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.
 - IIb Software item displays information requiring immediate operator action to mitigate a hazard. Software failures will allow or fail to prevent the hazard's occurrence.
 - IIIa Software item issues commands over potentially hazardous hardware systems, subsystems or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.
 - 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.
 - IV Software does not control safety critical hardware systems, subsystems or components and does not provide safety critical information.
- b. Potential Hazard Severity
 - A Catastrophic: Safety Critical Event / Function could contribute to a hazard that leads to death, system loss, or severe environmental damage.

- B Critical: Safety Critical Event / Function could contribute to a hazard that leads to severe injury, severe occupational illness, major system or environment damage.
- C Marginal: Safety Critical Event / Function could contribute to a hazard that leads to minor injury, minor occupational illness, or minor system or environmental damage.
- D Negligible: Safety Critical Event / Function could contribute to a hazard that leads to less than minor injury, occupational illness, or less than minor system or environmental damage.

Based on the above Software Control Category definitions, each software requirement and/or functional area can be evaluated to determine if it should be considered safety critical software. Based on the Software Control Category assignment for each software component, CSC and CSU, the worst potential hazard severity is identified.

Once the Software Control Category and potential hazard severity have been determined, then Table 8-4 (SHCM table) is used to determine the analysis/testing priority that should be used on each software component. The SHCM rankings can be performed at lower software functional areas than CSCI if deemed necessary.

Table 8-4: Software Hazard Criticality Matrix (SHCM)

POTENTIAL HAZARD SEVERITY	Catastrophic A	Critical B	Marginal C	Negligible D
CONTROL CATEGORY				
I	1	1	3	5
IIa	1	2	4	5
IIb	1	2	4	5
IIIa	2	3	5	5
IIIb	2	3	5	5
IV	3	4	5	5

S/W Risk Index	Risk Level	Evaluation Level
1	High Risk	Significant analysis and in-depth testing required.
2	Medium Risk	Detailed requirements, design and code analysis required, with sufficient testing to assure safety implementation.
3-4	Moderate Risk	High level requirements, design and code analysis with testing as needed.
5	Low Risk	Sufficient analysis to assure it does not adversely effect identified safety critical software and contribute to a more severe condition.

In MIL-STD 882C, the SHCM index is referred to as a Software Hazard Risk Index (SHRI). Since the term “Hazard Risk Index” is also used for generating hazard assessments, the SMART methodology uses the term “SHCM Index” to avoid confusion. The process of generating an SHCM index is often confused with generating a Hazard Risk Index value due to similar terminology. Both processes use the same hazard severity categories shown in Appendix C. ***The difference is that an HRI is used to document an identified safety problem, where an SHCM index is used to determine what areas of software need the most extensive analysis.*** When generating an SHCM index, nothing within the software is unacceptable since the actual analysis and testing have not been performed. Both the SHCM and Hazard Analysis processes and tables are contained in Appendix C.

9. PHASE III – SOFTWARE SAFETY DESIGN VERIFICATION

9.1 PURPOSE

The purpose of this phase is to insure the design is complete, free of defects, and traceable to the software and interface requirements approved in Phase II, does not introduce any Abnormal Condition and Events (ACEs) and/or defeat implemented safety features. During this phase, Software Design Documents (SDDs) are reviewed along with proposed design changes to these documents. The establishment of Verification Requirements (VRs) under each SR is initiated. VRs describe how the SRs will be verified. In addition, a safety positive measures list is initiated.

9.2 INPUTS/OUTPUTS/TASK OVERVIEW

There are two categories of tasks. The first category is joint IV&V-Safety Tasks. These tasks are performed jointly between the IV&V and Safety teams to reduce redundancy and improve efficiency. These tasks are shown in **blue** in Figure 9-1. The second category is safety specific tasks that are performed solely by the Safety Team are shown in **red** in Figure 9-1.

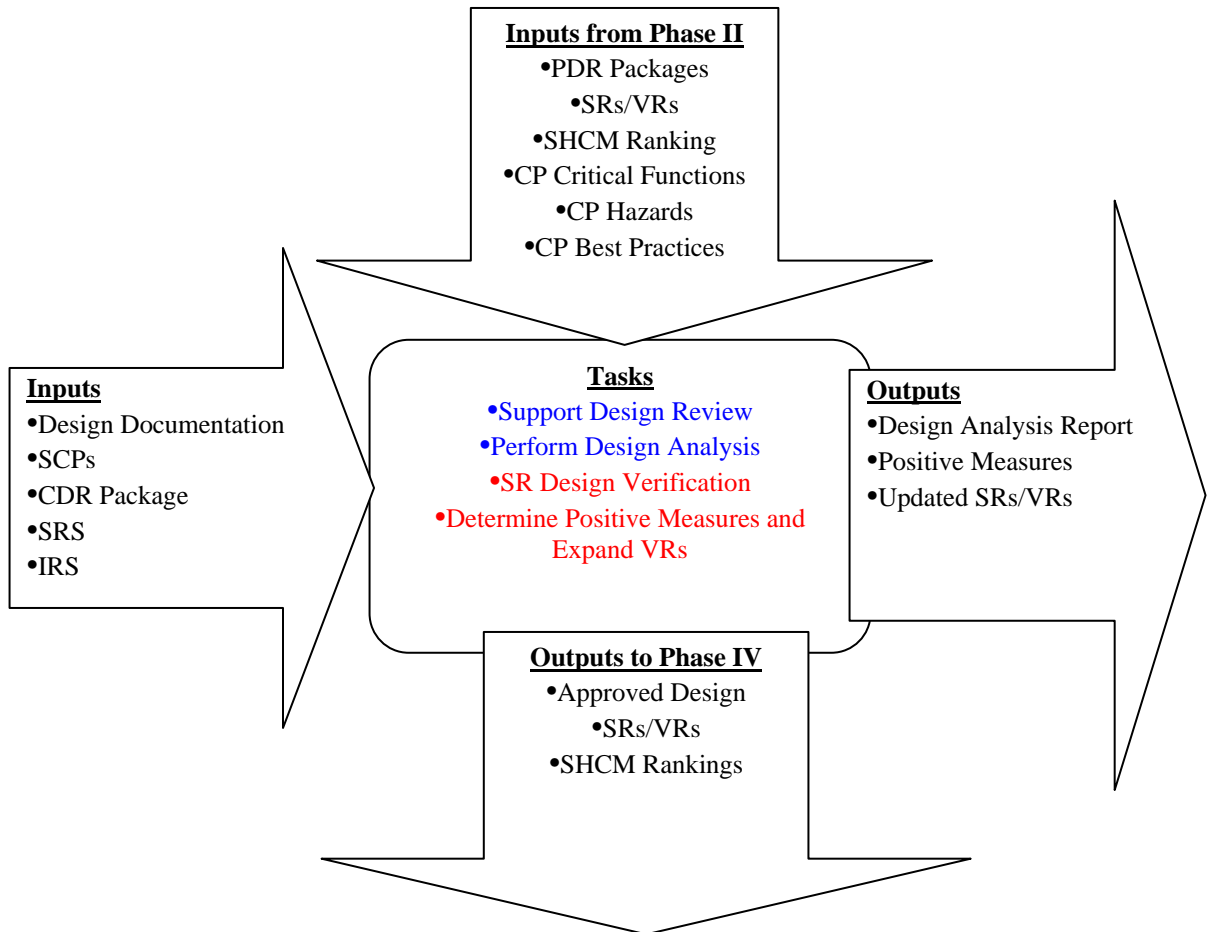


Figure 9-1: Phase III – Software Safety Design Verification Phase

Figure 9-1 shows a graphical overview of Phase III. Paragraph 9.3 describes the Joint IV&V-Safety Tasks in detail. Paragraph 9.4 describes the safety specific tasks.

9.3 PHASE III - JOINT IV&V-SAFETY TASKS

The IV&V and Safety Teams perform a coordinated design analysis reviews to reduce duplicate efforts. The tasks within the Joint Detailed Process are: Design Review Support, and Design Analysis.

9.3.1 DESIGN REVIEWS

During the development process, the software developer may present the completed portions of the design at PDR. When the Design is complete, the software developer schedules a CDR to present the total system design to the sponsor for approval. The joint IV&V-Safety Team provides technical support at these meetings to answer questions and ensure approved design meets the requirements. In addition to ensuring the design is accurate, the safety team needs to ensure that that the design is safe and that no abnormal conditions or events are being created in the design.

9.3.2 DESIGN ANALYSIS

Design analysis is a visual inspection verifying requirements are implemented in the software design. Defects, missing requirements, and design errors are reported for design redlines.

If the system under review is a new build, the entire design needs to be mapped to the requirements. The approved requirements must be reviewed to gain a full understanding of the software's intended purpose. After the background research is completed, the review of the design redlines is initiated. Design traceability includes performing a cross-reference of the approved requirements to the design to insure processing adheres to requirements.

If the system under review is a maintenance build, each design change should be mapped to a SCP. The first step in the design analysis process is to research the SCP. Each SCP needs to be reviewed to ensure the appropriate design changes were implemented.

In addition to performing design traceability, other analyses can also be performed for a more thorough analysis. Examples of these analyses are:

Data Analysis – Evaluates the description, and usage of each data item in the software design

Data Flow Diagrams – A graphical representation of a system or software at any level which demonstrates the flow of data.

Control Flow Analysis - A graphical representation of a system or software at any level which demonstrates the flow of control.

Fault Tree Analysis – Analysis technique where undesired events are specified and analysed to find all credible ways in which the undesired event can occur.

Using the SHCM rankings determined in Phase II, the SSSA could decide to perform a more detailed design analysis on only the High Risk or Medium Risk areas due to budget and time constraints. All analysis results are documented in a Design Analysis Report.

9.4 PHASE III - SAFETY SPECIFIC TASKS

9.4.1 PERFORM SR DESIGN VERIFICATION

The objective of SR Design Verification for the System Software Safety Analysis effort is to ensure Safety Requirements derived in Phase II have been properly allocated to design.

After SRs are developed in Phase II, the SRs and their associated VRs must be mapped to the safety critical portions of the design. Analysis needs to be performed to ensure that the SRs are being met in the design. This is done in conjunction with the IV&V team using the tasks specified for design analysis in paragraph 9.3.2.

9.4.2 DETERMINE POSITIVE MEASURES AND EXPAND VRS

As design features are further developed, the VRs can be expanded to give the code analyst and tester further insight into what needs to be examined during the coding and testing phases.

The design must also be reviewed to determine if the design has implemented any Positive Measures. Positive Measures is a safety term for design features that identify, control, and/or eliminate the potential risk that a hazardous condition will occur leading to a mishap. There should be positive measures for each potential software hazard within the system. Any positive measures created in the design should be documented and the SRs/VRs updated accordingly. By updating the SRs/VRs, it will ensure that the Positive Measure are adequately analysed and tested during the code and analysis and testing phases.

Preliminary Positive Measures are provided as input to the Customer for establishing Technical Data Packages (TDPs) for Software Safety Technical Review Panels (TRPs).

10. PHASE IV – SOFTWARE SAFETY CODE ANALYSIS

10.1 PURPOSE

The Safety Code Analysis Phase is a two-step process. The first step consists of code traceability analysis performed with the IV&V team. The second step ensures that safety code analysis is performed for each Safety Requirement. The SHCM rankings created in Phase II are used to prioritize the code analysis effort.

The SSSA goals are is to inspect the code to detect both performance and safety problems. This phase can be done prior to or at the same time as Phase V (Software Safety Testing). As code analysis is being performed, test ideas can be forwarded to the test group. Code analysis will have to inspect the code to investigate anomalies detected during testing.

10.2 INPUTS/OUTPUTS/TASKS OVERVIEW

There are two categories of tasks. The first category is joint IV&V-Safety Tasks. These tasks are performed jointly between the IV&V and Safety teams to reduce redundancy and improve efficiency. These tasks are shown in **blue** in Figure 10-1. The second category is safety specific tasks that are performed solely by the Safety Team are shown in **red** in Figure 10-1.

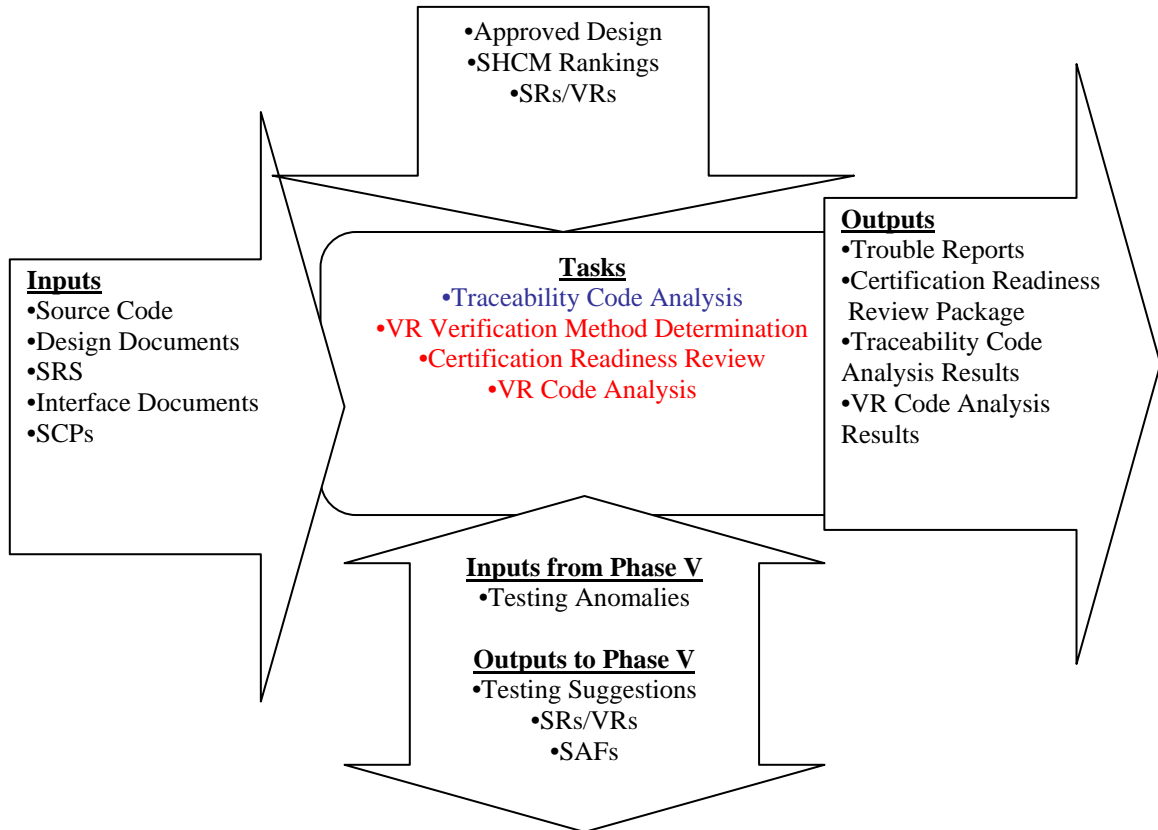


Figure 10-1: Phase IV – Software Safety Code Analysis Phase

Paragraph 10.3 describes the Joint IV&V-Safety Tasks in detail. Paragraph 10.4 describes the Safety Specific Tasks in detail.

10.3 PHASE IV – JOINT IV&V-SAFETY TASKS

The joint IV&V-Safety Teams perform a coordinated code analysis review of all changes to reduce duplicate efforts. This unified approach allows the Safety Team the opportunity to focus on those issues which impact safety and the IV&V team to focus on meeting technical requirements.

10.3.1 PERFORM TRACEABILITY CODE ANALYSIS

Code Analysis verifies that both requirements and design were successfully implemented in the final version of the software. On a new software build, a complete code traceability should be performed which maps the requirements to the design and coding modules to ensure 100% traceability. On a maintenance build, this traceability is usually performed on just the new requirements or change proposals.

Traceability is the cross-reference of requirements to design, and design to code. The analyst reviews the code to ensure the processing adheres to the requirements and design. Trouble Reports are used to document technical problems with the requirements, the design, or the source code. Any areas of code that are not directly traceable to design and requirements should be further analysed to

determine if the code is necessary. All non-traceable code should be documented in a Code Analysis Report.

10.4 PHASE IV – SAFETY SPECIFIC TASKS

10.4.1 DETERMINE VERIFICATION METHOD FOR VRS

The purpose of Phase IV and Phase V is to ensure all of the Safety Requirements are met and addressed. Each VR must be verified for an SR to be complete. The Safety Team will evaluate all SRs and associated VRs to determine how the VRs will be verified. The easiest method for completely verifying each VR should be chosen; e.g. many VRs can be verified through tests, (software developer regression tests, software developer demonstration tests, or Safety stress tests etc), other VRs will not be testable and VR code analysis will have to be performed. Testing is described in Phase V – Section 11.0. The VR Code analysis is described in paragraph 10.4.3.

10.4.2 CERTIFICATION READINESS REVIEW

The purpose of the Certification Readiness Review (CRR) is to present the safety analysis plan for code analysis and testing to the customer for approval. The CRR is a combination of a Test Readiness Review (TRR) as well as a presentation of a code analysis plan. The Certification Readiness Review package contains a final version of the SRs/VRs with a plan on how each VR would be verified. To avoid duplicate effort with IV&V, some VRs can be verified through IV&V analysis as long as the results are documented. All code analysis results should be documented in Software Analysis Folders (SAFs). The CRR provides the status of all Requirements and Design Analysis as well as the status on test simulators and equipment in order to ensure the SSSA Team is ready for code analysis and testing on the system. A final schedule is also presented to the customer for final approval.

10.4.3 PERFORM VR CODE ANALYSIS

The purpose of VR Code analysis is to ensure those VRs assigned to code analysis from paragraph 10.4.1 are thoroughly verified. The means for verifying the VR may vary. Some VRs can be verified by simple code inspection. For VRs that must ensure something does not happen, a fault tree approach would be better. For VRs that must ensure a safety critical process executes properly a critical flow analysis may be used. Unlike some safety approaches, which choose one code analysis method such as Fault Tree Analysis, Data Flow Analysis, Critical Flow Analysis etc., the SMART approach focuses on the elements that need to be verified (VRs) and then implements any verification method that best verifies the VR. Table 10-1 gives a brief description of some code analysis tasks

Some VRs are more safety critical than others. In order to focus on the most critical VRs, the SHCM rankings from Phase II are also used to prioritize analysis. A more thorough analysis approach should be used to address VRs contained in *High Risk* software. For example, fault tree analysis may only be performed on hazards in which the software addressing those hazards is *High Risk*. The SHCM rankings are used as a guide to focus the analysis on the most critical areas of software.

Table 10-1: Verification Methods for Code Analysis

Code Analysis Method	Description
Data Flow Analysis	A graphical representation of a system or software at any level which demonstrates the flow of data
Interface Analysis	An analysis that verifies the compatibility of internal and external interfaces of a software component.
Interrupt Analysis	Interrupt Analysis involves 1) determining impact of interrupts in the system, 2) analysing overall program prioritization and 3) determining impact of undefined interrupts.
Critical Flow Analysis	A graphical representation of a system or software at any level which demonstrates the flow of control
Fault Tree Analysis	Analysis technique where undesired events are specified and analysed to find all credible ways in which the undesired event can occur
Petri-net Analysis	A graphical technique that shows progression of state transitions. It is used to analyse recoverability, deadlock and fault tolerances.
Self-Modifying Code Analysis	Verifies that the design prevents unauthorized, inadvertent or self-modification of code. This would involve checking write-protection of code areas and performing periodic checksumming.
Unused and Unnecessary Code Analysis	Determine code which is logically excluded from execution. In addition, determine code which does not contain capabilities required by the system. *Unused and unnecessary code is undesirable for the following 3 reasons : 1) a symptom of design implementation error, 2) unnecessary complexity, and 3) might contain code which could be hazardous if inadvertently executed.

Regardless of which verification method is used, any code analysis results need to be thoroughly documented in SAFs.

Any anomalies detected result in the generation of a Trouble Report. When the discrepancy violates a Safety Requirement, the Trouble Report is marked Safety, and a Hazard Assessment is prepared. The Hazard Assessment process is described in Section 12.0.

11. PHASE V – SOFTWARE SAFETY TEST

11.1 PURPOSE

The purpose of this phase is to insure that the formal safety critical testing of the targeted system is conducted properly. This phase insure that the formal testing presented in the Certification Readiness Review is completed. During this phase, test execution and data analysis occurs. Code analysis is performed on any unexpected test results.

11.2 INPUT/OUTPUT/TASKS OVERVIEW

There are two categories of tasks. The first category is joint IV&V-Safety Tasks. These tasks are performed jointly between the IV&V and Safety teams to reduce redundancy and improve efficiency. These tasks are shown in blue in Figure 11-1. The second category is safety specific tasks that are performed solely by the Safety Team are shown in red in Figure 11-1.

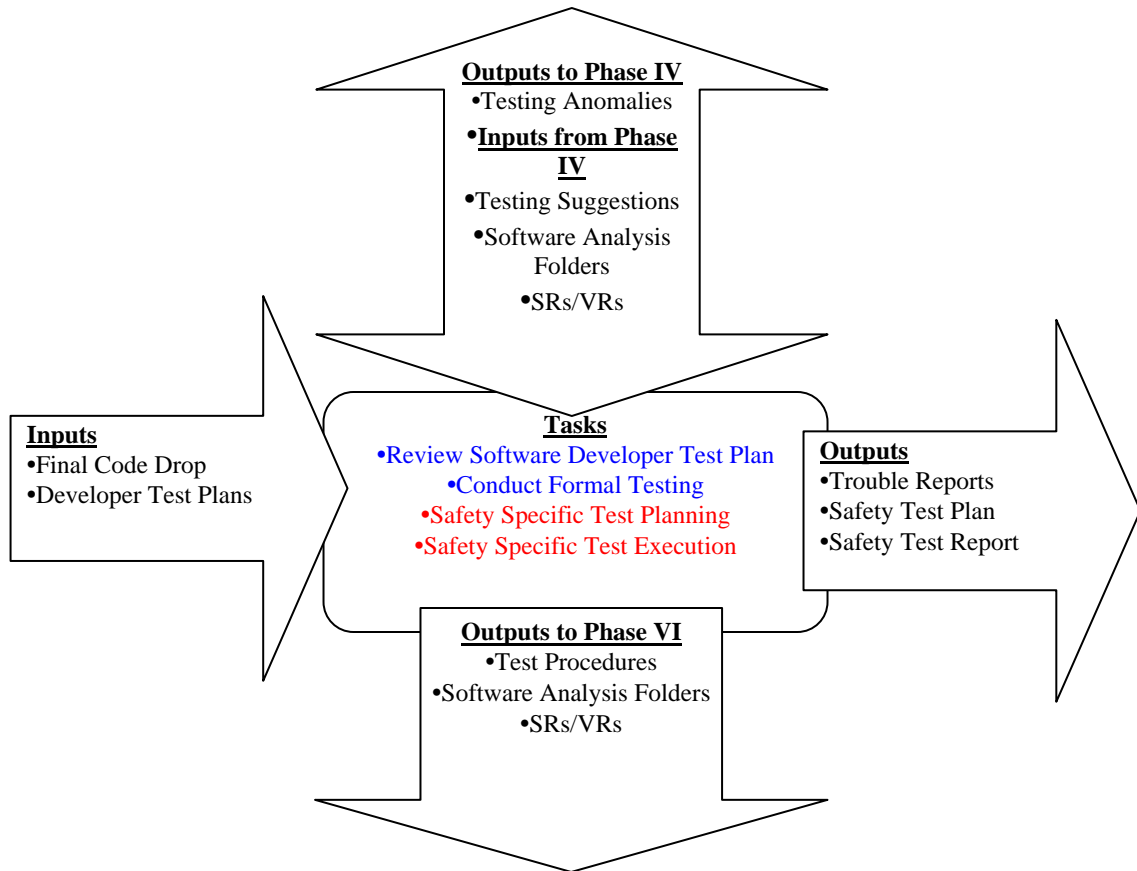


Figure 11-1: Phase V – Software Safety Test Phase

11.3 PHASE V – JOINT IV&V- SAFETY TASKS

The purpose of the IV&V- Safety joint tasks in this section is to review all software developer and or IV&V tests to determine overall effectiveness of testing effort and to determine if other testing is necessary.

11.3.1 REVIEW SOFTWARE DEVELOPER TEST PLAN

Upon receipt of the Test Plan from the developer, a thorough review is performed. The Test Plan is evaluated for thoroughness, scheduling, and manpower impacts. The Software Safety Analysis team may evaluate the test plan to determine how it could be enhanced to satisfy safety testing needs.

A comparison analysis of the Test Procedures against the Software Requirements Specifications is performed to ensure that all requirements are completely tested

The Software Safety Team can map testing requirements to VRs that have been adequately addressed

For a maintenance build, each of the changes should be tested thoroughly along with a set of regression tests to insure that functions that were previously tested still perform correctly after the software changes.

11.3.2 CONDUCT FORMAL TESTING

Formal testing is the execution of tests described in the formal Test Plan under formal conditions. Formal testing requires that the test be conducted under direct customer supervision with quality control (QC) personnel present. The data collected from these tests is then analyzed. All problems are reported via a Trouble Report. Upon completion of all testing and data analysis, a Test Report is generated.

11.4 PHASE V – SAFETY SPECIFIC TASKS

11.4.1 SAFETY SPECIFIC TEST PLANNING

The objective of this task is to create a test plan for any VRs that were assigned to testing that could not be verified through software developer tests. These VRs usually address areas with high stress or invalid data scenarios. This plan is presented at a formal Certification Readiness Review which was addressed in Section 10.4.2.

The goal is to test only those VRs that can not be verified through other software developer tests or code analysis. Therefore, the number of VRs being tested in Safety Specific Tests is minimal.

The number of tests is determined by the VRs that need to be verified. Usually, one large Safety Stress Test can verify most of the VRs in this category. VRs that are testing requirements such as faulty load media etc. may have to be verified in a very small test.

11.4.2 SAFETY SPECIFIC TEST EXECUTION

This objective of this task is to implement the Safety Specific Test Plan created in section 11.4.1 described above. Subsequent to customer approval, the formal

testing process begins. On some programs, this test plan will be executed by part of the Software Safety Team. On other programs, the test plan is submitted to the software developer who creates and executes the actual test with a Software Safety team member as a witness. The Software Safety Team may also be required to perform data analysis on recorded test data to ensure all applicable VRs are met. All anomalies are documented through Trouble Reports.

12. PHASE VI – SOFTWARE SAFETY EVALUATION

12.1 PURPOSE

The purpose of the Software Safety Evaluation Phase is to evaluate all SSSA analyses and test results and generate a Safety Certification Letter or Safety Analysis Report (SAR). The Safety Certification Letter provides a safety recommendation on whether or not to certify the computer program and hardware component undergoing Safety Analysis. A SAR report also provides a safety recommendation along with a summary of the findings normally found in the Final Report. Whether a Certification Letter or SAR report is provided depends on customer requirements.

The purpose of the Safety Evaluation Phase is to evaluate all SSSA analyses and test results and generate a certification letter. The certification letter provides a recommendation on whether or not to certify the computer program as being safe and approved for operation.

12.2 INPUT/OUTPUT/TASKS OVERVIEW

There are two categories of tasks. The first category is joint IV&V-Safety Tasks. These tasks are performed jointly between the IV&V and Safety teams to reduce redundancy and improve efficiency. These tasks are shown in blue in Figure 12-1. The second category is safety specific tasks that are performed solely by the Safety Team are shown in red in Figure 12-1.

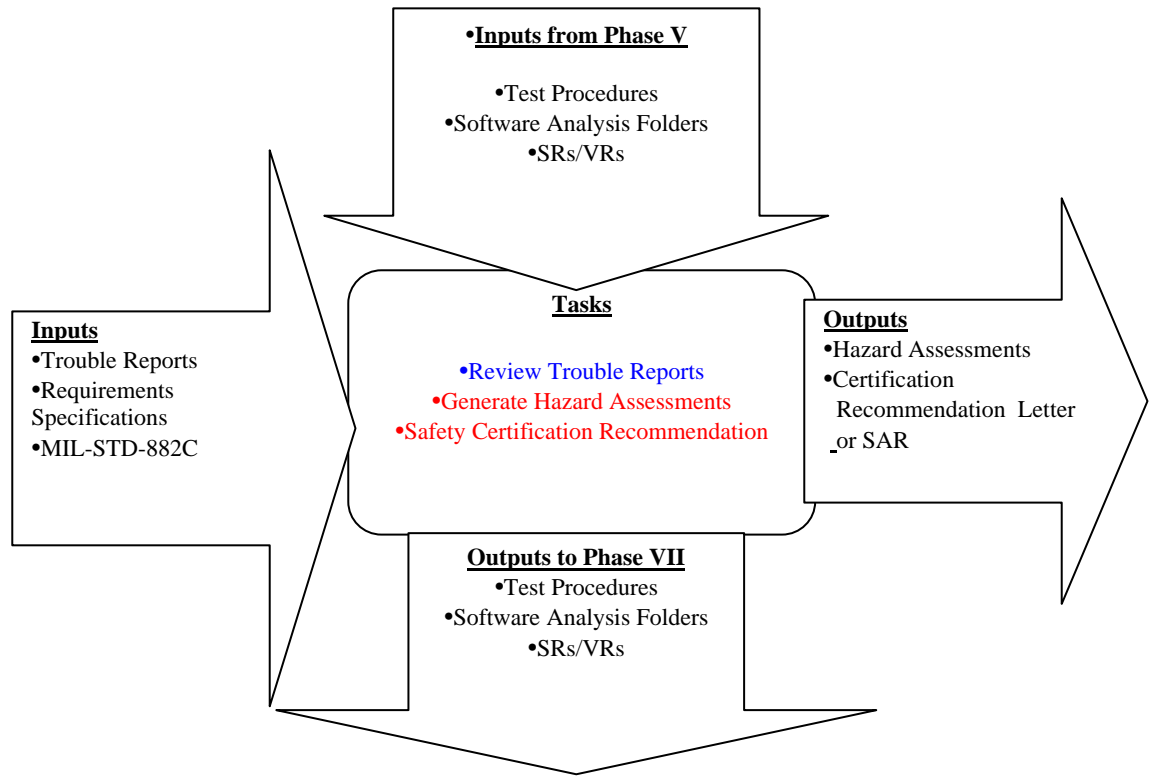


Figure 12-1: Phase VI - Software Safety Evaluation Phase

12.3 PHASE VI – JOINT IV&V/SAFETY TASKS

During this phase the only joint IV&V - Safety task is to thoroughly review all Trouble Reports to determine if a new build is required.

12.3.1 REVIEW TROUBLE REPORTS

Each trouble report is reviewed to assess the impact on performance and safety. The effect on performance will be evaluated and the Trouble Report will be given an appropriate priority based on the impact of the problem. For those Trouble Reports with a safety impact, Hazard Assessments (HA) are generated. The HA addresses the specific safety concerns for the Trouble Report. The safety impact is generally very different than the performance impact. Some Trouble Reports may have a very low performance impact but a very high safety impact.

12.4 PHASE VI – SAFETY SPECIFIC TASKS

12.4.1 GENERATE HAZARD ASSESSMENTS

Once a Trouble Report has been identified as safety, a Hazard Assessment needs to be written. Hazard Assessments address the specific safety hazard which could occur. Hazard Assessments address actual problems that were found during code analysis or testing. An example of a Hazard Assessment Form is contained in Figure 12-2. The HRI value (see Table 12-1) denotes both the hazard severity and hazard probability of the hazard occurring. The process of determining an HRI value is documented in MIL-STD 882 and in the following paragraphs.

Hazard Assessment Form
STR # - 0756CA-R-040
Safety Title: Failure to execute Checksumming
HRI: 1E (or 10) (Acceptable with review)
Effect on System: Failure to perform Program Memory Checksum to ensure instruction, constant data, and Adaptation data areas of code have not been modified.
Risk Evaluation: Background tasks in the program are initiated by an operating system and may be suspended for higher priority processing. Currently, the only background task is LCU BITE (i.e., Checksumming). During testing, the operating system suspended the LCU BITE Background task as was allowed, but failed to ever unsuspend the task. At this point LCU BITE no longer executed, thus checksumming on the instruction, constant data, and Adaptation data areas of code was not performed. Some of the constant data is critical to the safety of the system. If data were to be modified the safety effect could be catastrophic. However, it is extremely unlikely that constant data would be modified inadvertently.
Status: Open
SRs/VRs Violated: LC207/LC20704; LC304/LC30401

Figure 12 -2: Hazard Assessment Example

DETERMINING THE HRI VALUE

HRI is a combination of two items: Hazard Severity Category and Hazard Probability

Hazard Severity Definition. Hazard severity categories are defined to provide a qualitative measure of the worst credible mishap resulting from the hazard. Severity levels are defined in MIL-STD-882C and show in Table 12-1.

<i>Description</i>	<i>Category</i>	<i>Definition</i>
CATASTROPHIC	1	Death, system loss, or severe environmental damage.
CRITICAL	2	Severe injury, severe occupational illness, major system or environmental damage.
MARGINAL	3	Minor injury, minor occupational illness, or minor system or environmental damage.
NEGLIGIBLE	4	Less than minor injury, occupational illness, or less than minor system or environmental damage.

Table 12-1: Hazard Severity Categories

Hazard Probability Definition. The probability that a hazard will be created during the planned life expectancy of the system can be described in potential occurrences per unit of time, events, population, items, or activity. Assigning a quantitative hazard probability to a potential design or procedural hazard is generally not possible early in the design process. A qualitative hazard probability may be derived from research, analysis, and evaluation of historical safety data from similar systems. Supporting rationale for assigning a hazard probability shall be documented in hazard analysis reports. Hazard probability rankings provided in MIL-STD-882C are shown in Table 12-2.

<i>Description*</i>	<i>Level</i>	<i>Specific Individual Item</i>	<i>Fleet or Inventory**</i>
FREQUENT	A	Likely to occur frequently	Continuously experienced
PROBABLE	B	Will occur several times in the life of an item.	Will occur frequently
OCCASIONAL	C	Likely to occur some time in the life of an item	Will occur several times
REMOTE	D	Unlikely but possible to occur in the life of an item	Unlikely but can reasonably be expected to occur
IMPROBABLE	E	So unlikely, it can be assumed occurrence may not be experienced	Unlikely to occur, but possible

Table 12-2: Hazard Probability Levels

Hazard severity and hazard probability when integrated into a table format produces the Hazard Risk Index (HRI) assessment matrix and the initial HRI risk categorization for hazards. The HRI Assessment Matrix is provided in Table 12-3.

Table 12-3: HRI Assessment Matrix

FREQUENCY of OCCURRENCE	HAZARD SEVERITY CATEGORY			
	(1) CATASTROPHIC	(2) CRITICAL	(3) MARGINAL	(4) NEGLIGIBLE
(A) FREQUENT	1	3	7	13
(B) PROBABLE	2	5	9	16
(C) OCCASIONAL	4	6	11	18
(D) REMOTE	8	10	14	19
(E) IMPROBABLE	12	15	17	20
Legend:				
Cell numbers 1-5:	UNACCEPTABLE , condition must be resolved. Design action is required to eliminate or control hazard.			
Cell numbers 6-9:	UNDESIRABLE , MA decision is required. Hazard must be controlled or hazard probability reduced.			
Cell numbers 10-17:	ALLOWABLE , with MA approval. Hazard control desirable if cost effective.			
Cell numbers 18-20:	ACCEPTABLE without review. Normally not cost effective to control. Hazard is either negligible or can be assumed will not occur.			

This prioritization of hazards will allow the program manager, safety manager, and engineering manager the ability to prioritize the expenditure and allocation of critical resources. Although it seems simplistic, a hazard with an HRI of 11 or higher

should have fewer resources expended in its analysis, design, test, and verification than a hazard of 4. Without the availability of the matrix, the allocation of resources becomes more arbitrary.

Another benefit of the Hazard Risk Index, is the accountability and responsibility of program and technical management to the system safety effort. The System Safety Plan (SSP) identifies and assigns specific levels of management authority with the appropriate levels of safety hazard severity and probability. The HRI methodology holds program management and technical engineering accountable for the safety risk of the system during design, test and operation, and the residual risk upon delivery to the customer.

From the perspective of the safety analyst, the HRI is a tool that is used throughout the product life cycle.

12.4.2 SUBMIT SAFETY CERTIFICATION RECOMMENDATION

The final task of the Evaluation phase is generation and submittal of the formal certification letter or SAR report which states the final recommendation of the SSSA effort. The recommendation is determined by the evaluation results. If the number and severity of open problems are unacceptable, a letter or SAR report recommending no certification is generated.

The certification letter is not a synopsis of all analysis results. The SSSA Final Report will document the analysis and testing performed on each SR. The certification letter serves to recommend or not recommend formal use of the software.

The SAR report contains a summary of the findings contained in the final report along with a certification recommendation.

The recommendation for certification is a combined decision between SSSA agent, the customer and program office. Certification can be recommended when a HRI of “Undesirable” exists, but Program Office acceptance is required. For an HRI of “Acceptable with Review”, the customer will make the final evaluation.

13. PHASE VII – SOFTWARE SAFETY PROCESS REVIEW AND DOCUMENTATION

13.1 PURPOSE

Phase VII concludes the seven phases of the SMART Methodology. This phase allows time for final documentation. This phase also provides for review of the process and lessons learned. The lessons learned are inputs to Process 2 (Software Safety Process/Technology Improvement).

13.2 INPUT/OUTPUTS/TASKS OVERVIEW

There are two categories of tasks. The first category is joint IV&V-Safety Tasks. These tasks are performed jointly between the IV&V and Safety teams to reduce redundancy and improve efficiency. These tasks are shown in **blue** in Figure 13-1. The second category is safety specific tasks that are performed solely by the Safety Team are shown in **red** in Figure 13-1.

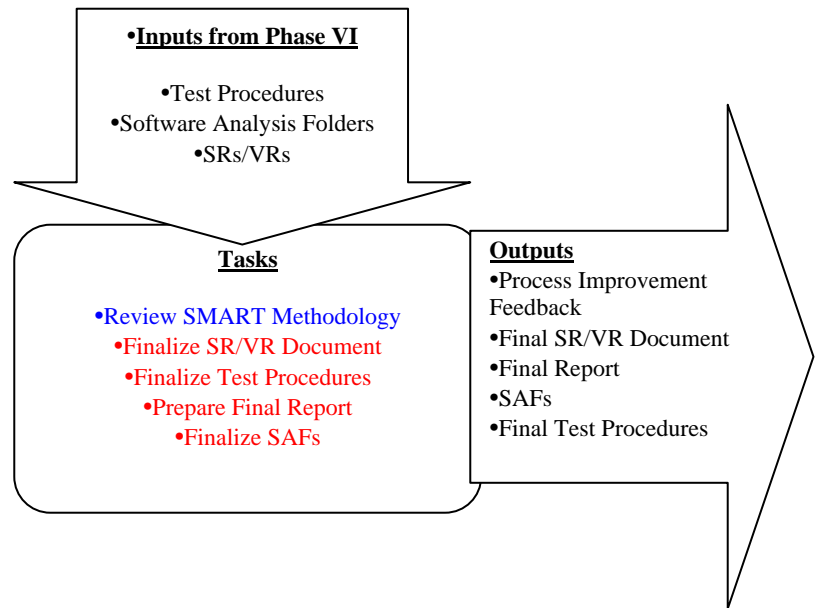


Figure 13-1: Phase VII – Software Safety Process Review and Documentation Phase

13.3 PHASE VII – JOINT IV&V-SAFETY TASKS

13.3.1 REVIEW METHODOLOGY

The IV&V and SSSA teams jointly review the joint task performed to determine and address issues and improvements. In addition the SSSA team will review the certification process to address any weak areas and evaluate customer concerns. All types of analyses performed are reviewed to determine if they were adequate given the final results and problems found. Any feedback or lessons learned are forwarded to Process 2 which will provide changes for the next safety certification effort.

13.4 PHASE VII - SAFETY SPECIFIC TASKS

The purpose of the safety detailed processes is to prepare all documentation which includes SR/VR Document, SAFs and a Final Report.

13.4.1 FINALIZE SR/VR DOCUMENT

The Preliminary SR/VR Document was delivered in Phase II. This final phase ensures that all technical and editorial comments generated in Phases III through VI are incorporated into the final document.

13.4.2 FINALIZE TEST PROCEDURES

The test procedures are formalized by incorporating the redlines that were generated during test execution. The test procedures need to accurately reflect the actual procedures that occurred for historical purposes.

13.4.3 PREPARE FINAL REPORT

A final report is prepared for each computer program undergoing Safety Analysis. The report provides conclusions and recommendations based on the Safety Analysis effort. The report shall also contain an executive summary, all Hazard Assessments (including HRI), the results of all analyses, and a summary of all Safety STRs. The Final Report is delivered to the customer.

13.4.4 FINALIZE SAFS

The SAFs contain the details of all the analyses and testing data. Unlike the Final Report, which is a summary of all the analysis results, the SAF contains all the details of the analysis. Due to the volume of data contained in the SAF, the results are usually more geared to the SSSA team for the next maintenance upgrade of the program than to the customer. However, the customer may want a formal delivery of the SAF.



This page intentionally left blank

APPENDIX A

Abbreviations and Acronyms

Abbreviations and Acronyms

Abbreviations and Acronyms used in this document are defined in Table A-1

Table A-1: List of Abbreviations and Acronyms

ACE	Abnormal Conditions and Events
CDR	Critical Design Review
CM	Configuration Management
COTS	Commercial-Off-The-Shelf
CP	Computer Program
CPCF	Computer Program Critical Function
CRR	Certification Readiness Review
CSC	Computer Software Corporation
CSCI	Computer Software Configuration Item
CSU	Critical Software Unit
FMECA	Failure Mode and Effect Criticality Analysis
HA	Hazard Assessments
HLSSR	High-Level System Safety Requirement
HRI	Hazard Risk Index
ICD	Interface Control Document
ICR	Interface Change Request
IDS	Interface Design Specification
IV&V	Independent Verification and Validation
IWG	Interface Working Group
JSSSH	Joint Software System Safety Handbook
PDR	Preliminary Design Review
PHA	Preliminary Hazard Analysis
PHL	Preliminary hazard List
PIDS	Prime Item Development Specification

QA	Quality Assurance
SAF	Safety Analysis Folder
SAR	Safety Analysis Report
SCC	Software Control Category
SCCB	Software Configuration Control Board
SCFL	System Critical Function List
SCP	Software Change Proposal
SDD	Software Design Document
SHCM	Software Hazard Criticality Matrix
SHL	System Hazard List
SHRI	Software Hazard Risk Index
SMART	System Methodology for Analysis, Review and Test
SR	Safety Requirement
SRS	Software Requirements Specification
SSDD	Software System Definition Document
SSEP	Software Safety Evaluation Plan
SSHA	Subsystem Hazard Analysis
SSP	Software Safety Plan
SSPP	Software Safety Program Plan
SSSA	System Software Safety Analysis
STR	Software Trouble Report
SWG	Safety Working Group
TDP	Technical Data Package
TRB	Technical Review Board
TRP	Technical Review Panel
TRR	Test Readiness Review
VR	Verification Requirement

APPENDIX B

System Safety Best Practices

Best Practice Safety Requirement (SR)

Both MIL-STD-882C and STANAG 4404 have numerous software safety requirements (or objectives) interspersed throughout the written text. These requirements aid in the development of safe software and identify, from past experience, potential areas in which serious safety problems could be created if not implemented correctly. Application of these SRs ensures that software safety objectives are effectively and consistently applied to all software developed for an identified safety critical component.

This appendix provides a set of Best Practice SRs, Labeled the 300 series.

SR 301 – The system software shall be design to isolates critical data from non-critical data and isolates system specific parameters (i.e. adaptation data) and functionality for easy maintenance.

SR 302 – The system software shall not contain unnecessary or extraneous code.

SR 303 – The system software shall only interface with other system and/or subsystem software through input and output parameters passed in the procedures and functions.

SR 304 – The system software shall have processing in place to handle invalid input data.

SR 305 – The system software design shall initialize all data before use.

SR 306 – The system software design shall have only one entry and one exit point to all procedures and loops.

SR 307 – The system software design shall contain one and only one execution path for safety critical functions.

SR 308 – The system software shall not contain Terminate, Abort, or Delay instructions.

SR 309 – All data contained in the system software shall be of appropriate types and ranges.

SR 310 – The system software design shall ensure timers have appropriate timer values.

SR 311 – The system software design shall ensure that the software is tamper-proof against inadvertent software modifications.

SR312 – The Ship Control System Algorithms design shall ensure that safety critical data follows a naming convention.

APPENDIX C

Software Design / Coding Guideline Options

Safety Design/Coding Guideline	Safety Documentation
Ensure at least two people are familiar with the design, code, testing and operation of each software module in the system.	JSSS Handbook – D.2.3 STANAG 4404 – 6.3
Due to the inability to perform design analysis on COTS software, the system should be designed to isolate safety-critical functions from the COTS software to reduce its influence on the safety-critical functions in the system.	JSSS Handbook – 4.7, D.7 882C – 205.2.1.f(1)
Patches are discouraged throughout the development process. All software changes shall be coded in the source language and compiled prior to entry into operational or test equipment	JSSS Handbook – D.2.4 STANAG 4404 – 6.5
CPUs that process entire instructions or data words are preferred to those that multiplex instructions or data (e.g. , an 8-bit CPU is preferred to a 4-bit CPU emulating an 8-bit machine).	STANAG 4404 – 9.1.a JSSS Handbook – D.5.2
CPUs with separate instruction and data memories and busses are preferred to those using a common data/instruction buss. Alternatively, memory protection hardware, either segment or page protection, separating program memory and data memory is acceptable.	STANAG 4404 – 9.1.b JSSS Handbook – D.5.2
CPUs, with microprocessors and computers that can be fully represented mathematically are preferred to those that cannot.	STANAG 4404 – 9.1.c JSSS Handbook – D.5.2
Changes to safety critical computing system functions on deployed or fielded systems should be issued as a complete package for the modified unit or module and shall not be patched.	STANAG 4404 – 16.1 JSSS Handbook – D.12.1
Firmware changes should be issued as a fully functional and tested circuit card. Design of the card and the installation procedures should minimize the potential for damage to the circuits due to mishandling, electrostatic discharge, or normal or abnormal storage environments, and should be accompanied with a proper installation procedure.	STANAG 4404 – 16.2 JSSS Handbook – D.12.2
Safety kernels, if implemented, should reside in non-volatile read only memory (ROM) or in protected memory that cannot be overwritten by the computing system.	STANAG 4404 – 1.4 JSSS Handbook –D.7.4
Where practical, safety critical functions are performed on a standalone computer. If this is not practical, safety critical functions shall be isolated to the maximum extent practical from non-critical functions.	STANAG 4404 – 2 JSSS Handbook –D.3.2, D.3.19F
The system and its software should be designed for ease of maintenance by future personnel not associated with the original design team.	STANAG 4404 – 3, 15.4 JSSS Handbook –D.3.3, D.11.17 STANAG 4452 –2.2.4
Safety critical variables should be identified in such a manner that they can be readily distinguished from non-safety critical variables.	STANAG 4404 –15.12 JSSS Handbook –D.11.25
Perform testing of Safety Critical Computing Systems Functions in accordance with approved test plans, descriptions, procedures, cases and criteria. Results should be accurately recorded, documented , and reported.	STANAG 4452 –2.6, 8.2.3.a JSSS Handbook – 4.2 NASA –4.6.1
Ensure values for timers are annotated in the code. Comments shall include a description of the timer functions, and the rationale for the timer value.	STANAG 4404 – 5.11 JSSS Handbook –D.11.24



This page intentionally left blank

APPENDIX D

References

1. MIL-STD-882C, System Safety Program Requirements, January 19, 1993
2. Joint Software System Safety Handbook (JSSSH), Joint Software System Safety Committee Software System Safety Handbook, A Technical and Managerial Team Approach.
3. IEEE STD 1228-1994, IEEE Standard for Software Safety Plans
4. STANAG 4404 (Edition 1), NATO Standardization Agreement (STANAG), Safety Design Requirements and Guidelines for Munition Related Safety Critical Computing Systems.
5. STANAG 4452 (Draft), NATO Standardization Agreement (STANAG), Safety Assessment Requirements for Munition Related Computing Systems
6. NASA Guidebook for Safety Critical Systems, NASA-GB-1740.13-96, July 13, 1995
7. MIL-STD-1658 (OS), Notice 2, 1 March 1978, Shipboard Guided Missile Launching System Safety Requirements, Minimum
8. Requirements Manual for Submarine Fly-By-Wire Ship Control Systems, NAVSEA T9044-AD-MAN-010 REV A

Disclaimer

The information, views and opinions expressed in this paper constitute solely the authors' views and opinions and do not represent in any way CSC's official corporate views and opinions. The authors have made every attempt to ensure that the information contained in this paper has been obtained from reliable sources. CSC is not responsible for any errors or omissions or for the results obtained from the use of this information. All information in this paper is provided "as is," with no guarantee by CSC of completeness, accuracy, timeliness or the results obtained from the use of this information, and without warranty of any kind, express or implied, including but not limited to warranties of performance, merchantability and fitness for a particular purpose.

In no event will CSC, its related partnerships or corporations, or the partners, agents or employees thereof be liable to you or anyone else for any decision made or action taken in reliance on the information in this paper or for any consequential, special or similar damages, even if advised of the possibility of such damages.

Copyright © 2008 Computer Sciences Corporation.