Computer & Robotics

A New Approach to Promote Safety in the Software Life Cycle

Shahrzad Oveisi [a,b,*], Mohammad Ali Farsi [a], Mohammad Nadjafi [a], Ali Moeini [b]

[a] *Aerospace Research Institute (Ministry of Science, Research and Technology), Tehran, P.O.B. 14665-834, Iran*
[b] *Department of Algorithms and Computation, College of Engineering Sciences, University of Tehran, Tehran, Iran*

Received 27 May 2019; Revised 10 November 2019; Accepted 27 November 2019; Available online 31 December 2019

Abstract

Developing a reliable and safe system is one of the most important features of advanced computer-based systems. The software is often responsible for controlling the behavior of mechanical and electrical components as well as interactions between components in systems. Therefore, considering software safety and fault detection are essential in software development. This paper introduces an approach to engineering evidence that examines the software in its lifecycle according to the principles of software safety and system safety engineering. This approach ensures that software risks are identified and documented in the software lifecycle, after which the risks are reduced to an acceptable level in terms of safety according to the proposed methods. The presented approach was applied to a real master case with positive results, namely the Data and Command Unit.

Keywords: *Safety, Life Cycle, Software Development, Software Safety, Computer-Based System.*

1. Introduction

Safety based design with consideration of safety in software is an essential component in the industries related to modern sciences and their design processes. System engineers define reliability and safety requirements to achieve a system at a world-class level. This system should work in various conditions and keep its performance at an acceptable level. With the ever-increasing requirement of reliability and safety for critical and vital systems, accurate assessment of the pending failure of a system has become an active research area over the past decades [1]. The existence of software on systems increases complexity, and both hardware and software can create faults in the system. The faults in complex systems can cause a loss of features in the system. For this reason, the need for methods is felt that can prevent these faults and discuss software safety and reliability [2, 4].

According to studies conducted by National Institute of Standards and Technology, the cost of a software error in US economy is approximately 59.5 billion US dollars per year (nearly 0.6% of GDP). It is also estimated that over one-third of costs (i.e. 22.2 billion US dollars) can be cut with the improvement in infrastructure, including the use of software assurance and tests that detect, anticipate, and eliminate the fault. Considering the above statements, the application of a method to increase software reliability seems to be essential in these systems [31]. Safety addresses risk factors to analyze and monitor their control and mitigation strategies. In addition, Safety is related to the field of software development, which involves testing and modeling the ability of the software to function correctly without failures or software faults causing software failures5-6.In order to understand the importance of software safety, an example of

* Corresponding author. Email: shahrzad.oveisi@gmail.com

a software failure that resulted in system failures has been presented below.

Demonstration of Autonomous Rendezvous Technology (DART) is designed to be implemented in Multiple Paths Beyond Line of Sight Communications (MUBLCOM) in a variety of maneuvers in the satellite. These operations are accomplished without assistance personnel engaged in the ground activities. The activities of the DART team show that software causes play a significant role in the history of incidents, which has not been taken into account in many events in the past and caused a big bug in accident investigations. Also, the team believes that any changes in the system software should be documented appropriately, and safety-critical software should not be hypersensitive to erroneous data [7]. Implementing a Software Safety Program is the best routine to help confirm the identification of potential software risks and reduce the respective risks. Successful implementation of the Software Safety Program involves the selection and application of effective analytical tasks and methods tailored to specific project needs, which meet the terms of the Software Safety Program.

The Data and Command Unit is one of the main subsystems in aerospace systems, which issue the movement start signal as the basis of all subsequent activities of this type of system. The essential system operation software is responsible for issuing commands such as sending and receiving signals between internal parts of the system, setting the micro timer, and so forth. Unexpected interactions between hardware, software, and operating environment can provide potentially dangerous and/or hazardous conditions. Unlike electrical or mechanical hardware, the software is not eroded over time, and it can be said that the software does not experience the failure. However, hardware performance requires proper software running, and the desired function of the system, which is determined by software, may not be executed by the embedded system software. Failures in software can cause downtime (out of service/and or out of operation time) and poor performance of asystem [8]. The poor performance or at worst case the downtime of a system can be listed in any of the following ways: partial or entire system unavailability, system imperfections and lack of accessibility, wrong results, loss of or unusable data, and slow system performance.

The main reasons for common software failures are the inability to upgrade, a logic error in software code, a bug in a core network device, resource exhaustion, data corruption, hardware failures [9]. Also, common sources for the failure of hardware, which can internally or externally control software execution, are as follows: memory failures either in the space of code or variables, CPU failures (ALU and registers), external failures (ports, watchdog, timer, interrupt manager, and so on). Errors of software logic may occur due to defective or incompatible requirements as well as software design errors or implementation of the code. Failure conditions due to software logic errors may include endless loops, false calculations, longer time to complete the executive routines, and the like. Moreover, the software stored on an embedded system may not be correct if the tools necessary to configure and compile the software as expected are lacking. For example, memory cell defects can lead to conditions where the software jumps inadvertently to the end of a routine or middle of another routine. Failure scenarios such as returning the wrong priority or inability to return (and thereby blocking interruptions of less importance) can also be a function of memory failure. Embedded system designers can use engineering procedures of system safety to deal with software defects. However, the unique scenarios of possible failure and general complexity of software will ensure that other software-specific tasks and processes are included in the overall system safety plan. To address this requirement, an effective implementation program of safety techniques for the command system should also involve software safety practices. The Software Safety Program involves the implementation of a number of software-related tasks aimed at identifying and minimizing potential software failures [10]. While software safety requirements can be taken from software guides and other published sources, there is a deficit in research presenting all the approaches leading to software safety of computer systems in all phases of the software lifecycle. Therefore, in this paper, we have presented approaches to establish software safety to be implemented on a real case study (Data and Command Unit) in a path to safety from the first phase of software to the end. The results showed improvement in safety of this software system.

In Section II, we will review the research background. In Section III, the status of the problem is reviewed. In Section IV, the methods of safety in the software lifecycle will be discussed, and in the final section, we will use these approaches for DCU evaluation of a Data and Command Unit.

2. Research Background

Several methods, guidelines, and standards have been developed to provide information in the field of software safety and software safety processes. The requirements for implementing a systematic technique/approach to software safety have been standardized as essential and inseparable in the NASA-STD-8719.13A standard provided by the JSSC committee. This standard is applied in software issues that may contribute to reaching-detecting or taking corrective action for a system in a specific dangerous state, which is intended to mitigate the damage if a mishap occurs. During the safety analyses of the subsystem/system, safety-critical software is identified and the software safety level determined based on the category of the system and the severity of the hazard [11]. Moreover, the JSSC handbook provides guidelines and technical approaches to management and control of the risk issues associated with an acceptable safety level with high assurance levels in order to execute the software inside the system [12]. Accordingly, MIL-STD-882C has provided system safety in detail and covered the software safety process in part. Primarily, MIL-STD-882C provides a process for the assessment of software hazard risk in which the degree of control and potential hazard severity are considered as the exercises of software over the hardware [13]. Also, in the field of airborne systems performance, required safety equipment according to airworthiness requirements with high confidence intervals is provided by Radio Technical Commission for Aeronautics (RTCA): RTCA/DO-178B. In this guideline, the safety tasks on software categories are discussed in detail [14]. The MOD DEF STAN 00-55 standard emphasizes the necessary procedures for design, production, specification, coding, and maintenance activities in-service, as well as software modification of safety-critical issues. Two categories related to software are provided in the standard: safety-critical and safety-related software. The former has discussed the functions of systems that are imposed on critical conditions related to the safety of the system. The latter is related to a system safety function encompassing all Safety Integrity Levels (SILs) and covering the high risked human-life to recommend requirements for mitigating the causes from a software point of view [15]. With respect to software requirements and activities for safety phases of the lifecycle, the IEC 61508 part 3 standard is should be used to develop a safety-related system during the design and development of safety-related software, as well as requirements for software safety validation [16]. Furthermore, in the field of automotive software, Motor Industry Software Reliability Association (MISRA) has provided specific issues related to the information on guidelines. While MISRA does not directly address safety software, it presents a general approach to software development with recommendations [17]. As a final research background, APT Research Inc. provided a process for verification and definition of software's critical safety functions in 15 steps. The process involves identifying hazards of the system, detecting requirements for software safety function, and safety efforts tailored to critical situations [18].

As mentioned, software safety considerations have been discussed in many papers, but the life cycle of the software along with an actual case study has been relatively ignored. Because of the importance of safety in a software process, in this paper, methods have been developed to provide safety and to apply a Data and Command Unit.

3. Statement of the Problem

As can be seen in Figure 1, if hardware or software faults in the system are not detected by fault detection methods/systems, they can lead to system errors, the propagation of which can result in safety failure or hazard that can ultimately cause an unfavorable event. According to the above statements, a fault in the system can lead to the occurrence of unfavorable events; therefore, the methods capable of preventing these faults are vital to the system. Considering the importance of this matter as well as the crucial role of software in computer-based systems, this paper discusses software production methods based on safety in the life cycle.

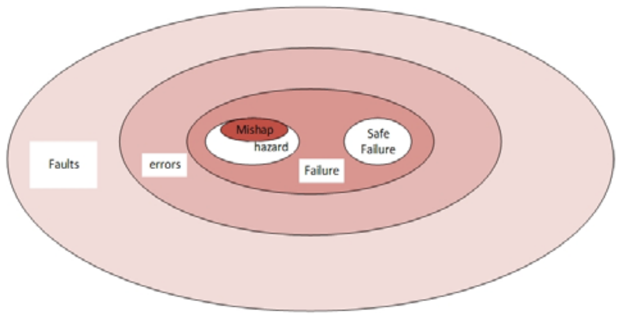


Fig. 1. A view from fault to mishap.

3.1. Software Safety Methods in Risk Management Process

The success of a software development project depends quite heavily on the amount of risk that corresponds to the level of risk associated with each project activity. As a project manager, it is not just sufficient to be aware of the risks. To achieve a successful outcome, project leadership must identify, analyze and evaluate all the major risks. Then, the identified risks are reduced to an acceptable level. Software assurance procedures are used to identify the risks. In software, assurance must be considered safety, reliability, quality, verification, and validation in the software life cycle. To maintain these disciplines, a variety of risk identification methods are used. We focus on safety and its approaches, including FTA, FMEA, PHA, etc. As shown in Figure2, software assurance methods generally play a major role in the software risk management process.

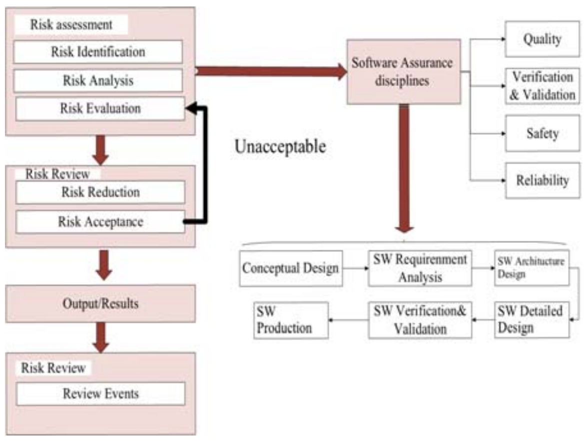


Fig. 2. The software assurance methods in risk management process.

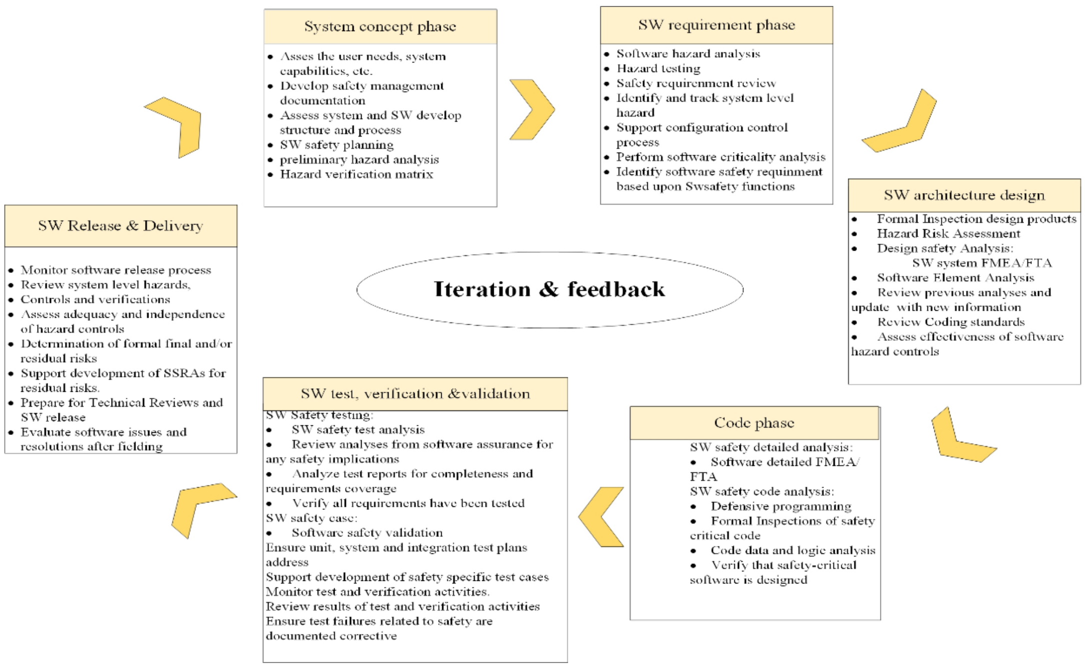
4. Safety in Software Lifecycle

In Figure 3, the safety tasks are shown in the software life cycle. In this paper, we focus on safety in software development, and in the rest of the paper, a number of important tasks are explained and applied to a real case.

4.1. The Proposed Approach

Figure 4 shows the proposed approach to increase software safety in engineering systems. The workflow of the proposed method is shown in the following Figure. Strategies are considered to reduce the risk severity of the whole approach. In the conceptual design phase of the approach, Preliminary Hazard Analysis (PHA) &Systems Theoretic Process Analysis (STPA) are performed. During the analysis phase, the requirements of Software Hazard Analysis are usually performed by Fault Tree Analysis (FTA), Failure Mode and Effects Analysis (FMEA) and Preliminary Hazard Analysis (PHA). The output of these methods is then applied to hazard testing. Critical Time is considered in hazard testing, and finally, review of software safety requirements is performed using one of the several methods. These methods are classified into (a)Top-down analysis of software safety requirements; (b) Analysis of critical requirements; (c) Specifications analysis and (d) Formal methods (B-method, Circle, KIV, Binary Decision diagram, etc.)

Subsequently, in the design phase of software architecture, SFTA, and SFMEA, the system level is designed with the help of software architecture and analysis phase outputs of requirements. In the code phase and detailed design of software, SFTA and SFMEA are done with the help of a map of sample variables as well as defragmentation of results and programming. At the end, Verification & Validation is done using a variety of tests.



System concept phase
Asses the user needs, system capabilities, etc.
Develop safety management documentation
Assess system and SW develop structure and process
SW safety planning
preliminary hazard analysis
Hazard verification matrix
SW requirement phase
Software hazard analysis
Hazard testing
Safety requirenment review
Identify and track system level hazard
Support configuration control process
Perform software criticality analysis
Identify software safety requinment based upon Swsafety functions
SW architecture design
Formal Inspection design products
Hazard Risk Assessment
Design safety Analysis:
SW system FMEA/FTA
Software Element Analysis
Review previous analyses and update with new information
Review Coding standards
Assess effectiveness of software hazard controls
Code phase
SW safety detailed analysis:
Software detailed FMEA/ FTA
SW safety code analysis:
Defensive programming
Formal Inspections of safety critical code
Code data and logic analysis
Verify that safety-critical software is designed
SW test, verification &validation
SW Safety testing:
SW safety test analysis
Review analyses from software assurance for any safety implications
Analyze test reports for completeness and requirements coverage
Verify all requirements have been tested
SW safety case:
Software safety validation
Ensure unit, system and integration test plans address
Support development of safety specific test cases
Monitor test and verification activities.
Review results of test and verification activities
Ensure test failures related to safety are documented corrective
SW Release & Delivery
Monitor software release process
Review system level hazards,
Controls and verifications
Assess adequacy and independence of hazard controls
Determination of formal final and/or residual risks
Support development of SSRAs for residual risks.
Prepare for Technical Reviews and SW release
Evaluate software issues and resolutions after fielding
Iteration & feedback

Fig. 3. The Software safety tasks in the software lifecycle.

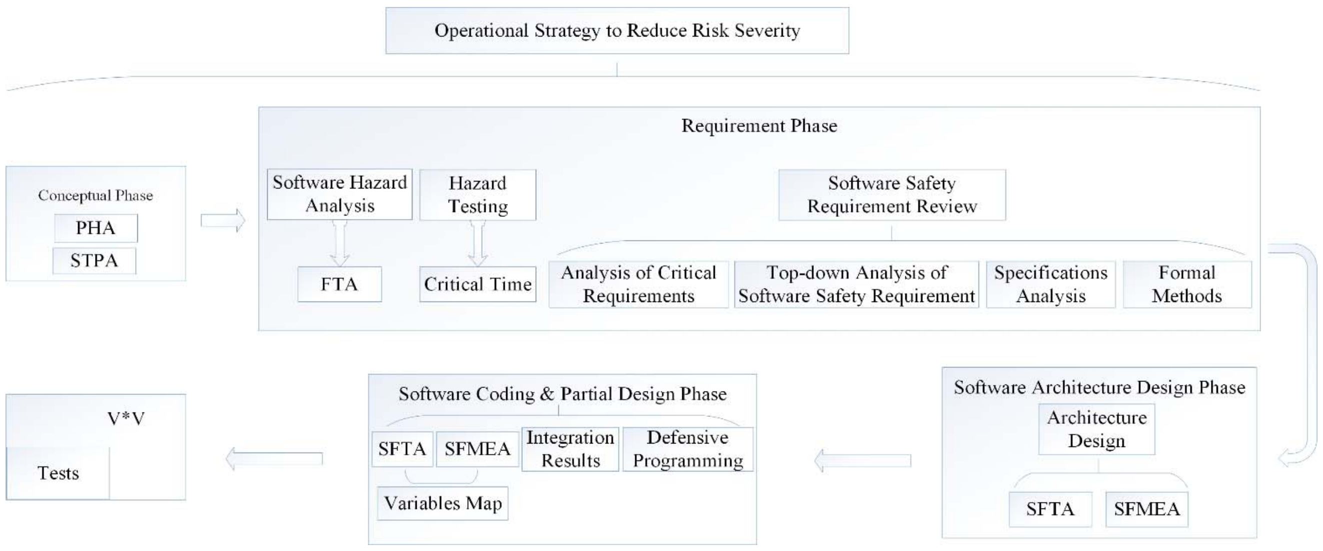

Operational Strategy to Reduce Risk Severity
Conceptual Phase
PHA
STPA
Requirement Phase
Software Hazard Analysis
Hazard Testing
Software Safety Requirement Review
FTA
Critical Time
Analysis of Critical Requirements
Top-down Analysis of Software Safety Requirement
Specifications Analysis
Formal Methods
Software Architecture Design Phase
Architecture Design
SFTA
SFMEA
Software Coding & Partial Design Phase
SFTA
SFMEA
Integration Results
Defensive Programming
Variables Map
V*V
Tests

Fig. 4. The workflow of the proposed approach.

4.2. Operational Strategies to Reduce Risk Severity

In a Real-World system, system elements are designed with high quality, and there are several strategies to diminish the severity of risk, including the strategies used to reduce the risk in designing aerospace systems as follows [19]. These methods are used throughout the software safety program approach.

- **Design for Fail-Safe:** In some systems, designers can provide a state for the system as a safe mode. When hazardous events occur or an operation may lead to a dangerous condition, the system or operator detects a hazard situation that can damage the system, operator or environment, changes in working conditions of system/operator, and transfer to a safe state. This transfer can be performed by stopping hazardous operation and starting a new mode or work with less risk. This system is called a fail-safe system**.**
- **Design for Fault-Tolerance:** In a system design process, designers can provide a situation for the system, so that when a failure occurs in the system, the system continues its operation and the failure is covered. This method of design is called a fault-tolerant design.
- **Design for Redundancy:** To increase the reliability of a system and reduce its failure likelihood, we should duplicate the critical items. In this case, the designers often use the second item as a backup for the failed item, which means having backup items that automatically “kick in” should one component fail. This scheme is recommended for single-point failures.
- **Providing early warning:** Failures that occur without warning are more dangerous than those with a warning. Catastrophic effects can be avoided by adding a warning device to system design.
- **Use of Hazard barrier:** A barrier is defined as a measure that is put in to prevent the release of a hazard or the occurrence of a top event once the hazard is released, and the barriers may be physical or non-physical, including correct/valid operating procedures. Time delays in processes decrease the speed of equipment, leading to maintenance when due. A compensating error is an accounting error that offsets another accounting error. These errors can be difficult to spot when they occur within the same account and in the same reporting period since the net effect is zero. A statistical analysis of an account may not find a compensating error.
- **Near-Miss Analysis:** Near-miss analysis, which refers to the detection and subsequent analysis of near-misses, is a technique used in the domain of risk analysis and safety, and near-miss analysis attempts to identify the root cause of the accident and prevent a recurrence, which has been used successfully in various industries for decades [9].

• **K-out-of-N system:** In a sensitive and important system such as the control system of a spacecraft or a safety system, designer use K-out-of-N scheme to improve and modify the system reliability. In this case, Nun it's provided and the final output depends on K units of the system. For instance, a system determines a parameter, and if K of the units' output is similar, then the system will work based on the output. In this system, if a unit is failed, the system performance may not be reduced. Only when the number of failed units is more than (N-K) units is the system failed. The system's reliability to achieve a safe system is increased by this scheme.

4.3. Safety in Conceptual Phase

In the conceptual phase, at first, the user’s requirements and the features that the system should have to be checked for this work in the documentation and checklist of the similar systems are assisted. Then the PHA for the system should be prepared, which can be done with Hazard Verification Matrix. PHA detects the failures and their countermeasures and then provides a list of them. Of course, PHA does not indicate those system failure scenarios that occur simultaneously, which is a disadvantage that can be mentioned. STPA is used for better identification of unsafe control actions. These control hazards are used in PHA and SFTA for phase requirement analysis. STPA provides guidance and systematic process to identify the potential for adequate control of the system, which could lead to a hazardous state resulting from inadequate control or enforcement of the safety constraints.

4.4. Safety in Requirements Analysis Phase

If software assurance assessment does not start from early phases of software development, the cost of software failures and errors is exponentially increased. The goals of Software Safety Program are to eliminate, reduce, or control the possible hazards associated with potential software failures. Software safety requirements may include national/international standards, customer requirements, or corporate needs. A matrix for the identification of software safety requirements can begin tracking requirements throughout the development process. The methods used to obtain software safety goals are categorized as into (a) Software safety requirements review; (b) Hazard testing; (c) Software hazard analysis; and (d) Operational strategies to reduce risk severity

Software hazard analysis identifies the possible software scenarios that can detect potential hazards during preliminary hazard analysis (PHA).Using the relationship created between software scenarios and potential hazards, prevention requirements of software hazards are developed, including specific software safety requirements through which hazard testing is identified at specific response times that must be provided by the operational software to help ensure that potential risks are avoided [20]. Finally, the examination of software safety requirements helps ensure that they are fully compliant and operational strategies for risk reduction, which are also considered in the design of software and related hardware. In the following sections, more detailed descriptions of software safety analysis methods are provided, which may be used to achieve the software safety objectives.

4.4.1. Software Hazard Analysis

Software hazard analysis involves the identification of possible software hazards that may lead to potential system hazards [21]. For any potential system hazard, possible software scenarios leading to hazards are identified. Software hazard analysis is obtained from a variety of sources and is usually divided into two categories: general and specific. General software safety requirements are derived from a collection of requirements that can be used in different applications and environments to solve common software safety problems [22]. Specific software safety requirements are taken from system-specific functionalities or constraints identified by methods defined in Table 1.

Table. 1. The methods used for identifying operating system limitations or functionality

Fault Tree Analysis (FTA): FTA takes a top-down approach to system failure, proceeding from the top to the intermediate and basic events, which are all events that are undesirable and connected through cause-and-effect relationships.	Method 1
Through Primary Hazard Analysis (PHA): Preliminary hazard analysis(PHA)is a semi-quantitative analysis that is performed to: 1.Identify all potential hazards and accidental events that may lead to an accident 2.Rank the identified accidental events according to their severity 3.Identify required hazard controls and follow-up actions	Method 2
Through bottom-up analysis of design data (for example, flow diagrams, failure mode, effects and criticality analysis (FMECA). FMECA uses an inductive approach to system design and reliability. It identifies each potential failure within a system or manufacturing process and uses severity classifications to show the potential hazards associated with these failures.	Method 3

The software fault tree is the most commonly used method among the three addressed approaches. As stated, it is a top-down method that identifies a number of unwanted effects at a high level, a process that is repeated so that the causes of related events as well as their associated factors are investigated. This analysis continues until a group of causes is identified [23]. These are underlying causes for software scenarios to analyze software system-level hazards. The requirements analysis phase obtains its safety inputs from the conceptual design phase, which is defined for our system as follows. The goal of PHA is to identify all potential hazards and hazardous events that may cause accidents and to detect required hazard controls and follow-up actions. PHA is performed in the conceptual phase of system development; therefore, safety requirements to control the identified hazards can be decided and incorporated into the initial design [24].STPA provides guidance and systematic process to identify the potential for adequate control of the system, which could lead to a hazardous state resulting from inadequate control or enforcement of the safety constraints.

4.4.2.Hazard Testing

Software safety requirements have been set for each of the possible software failures. During the hazard test, the need for real testing of the system under potential risk is assessed. The results of this test specify the deviation of error response times and the level required by the system. The main principle to be considered at this stage is the critical times and safety automation, which are briefly reviewed below [25].

• **Critical Time:** Safety-critical systems sometimes have a "critical time" feature, which is the interval between the occurrence of an error and system's reaching to an unsafe state. This interval is a period during which it is possible to perform automated/manual operation or protection through software, hardware, or a human operator. The design of recovery and protective measures should fully take into account the real-world conditions and crisis time [26].

• **Automatic safety:** The automatic safety is often needed if the critical time is shorter than the actual response time by a human operator, or if no human being is involved. This can be done by hardware, software, or a combination of them, considering the best system design to achieve safety.

• **Analyzing Software Safety Requirements:** Verification of requirements analysis activities is left to the software safety requirements, which can also search for new hazards, software functions that may be used to control the hazards and ways through which the software can run unexpectedly. Exploring software safety requirements is a recognized need for software safety compromised by hazards to help ensure the integrity and compliance of the software. Delayed detection of software safety needs can affect expense, timing, and so on. The final product of this work is a collection of software safety requirements that are based on the early development of system-level safety, as well as the results of hazard analysis and hazard testing. The requirements may also include general guidelines for software safety programming, industry, government, or international programming standards to be reviewed by the software development team. The analyses commonly related to software requirements are: (a) Top-down analysis of software safety requirements; (b) Analysis of critical requirements; (c) Specifications analysis; (d) Formal methods and model checking; and (e) Model-checking.

To explain the analysis methods in above, we Top-Down analysis and analysis of critical requirements have been described in the previous section. Formal specifications remove ambiguities and uncertainties, allowing for the detection of non-configurable errors. Elimination of defects is far less expensive when they are found and corrected early in the lifecycle. In other words, formal methods are a mathematical approach for determining and validating a software system. Specification analysis evaluates the completeness, correctness, consistency, and testability of software, and requirement specification analysis should evaluate requirements individually and as an integrated set. The important techniques used to perform specification analysis are: (a) Reading Analysis; (b) Traceability Analysis; (c) Control-flow analysis; (d) Information analysis; and (e) Functional simulation.

Reading analysis examines the requirements specifications to uncover inconsistencies, conflicts and ambiguous or missing requirements. The control-flow analysis inspects the order in which software functions will be performed. It identifies missing and inconsistently specified functions. The information-flow analysis explores the relationship between functions and data. Incorrect, missing and inconsistent input/output specifications are identified. Data flow diagrams are commonly used to report the results of this activity. Functional simulation models investigate the features of the software component for predicting quality, checking the human understanding of system features and assessing feasibility.

4.5. Software Architecture Design Phase

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both. As shown in Fig.3, the methods used in this phase are as follows (Table 2).

Table. 2. The Methods in software architecture design phase

Method	**Task**
Formal Inspection design products	Formal inspection focus on the major breakdown of software components, verifying the modularity and independence of all safety-critical component.
Formal Methods and model checking	Model Checking is a form of the formal methods that verifies a finite-state system. Model checking is an automatic method, and tools exist to provide that automation(for instance: SPIN and SMV)
SW FMEA/FTA	The main purpose of SFTA is to identify possible deficiencies in software requirements, design, or implementation, which may result in undesirable events in software. FMEA aims to identify, classify, and evaluate the hazards and risks associated with them.
Hazard risk assessment	Each software element that is not safety-critical is examined to assure that it cannot cause or contribute to a hazard
Software element analysis	Each software element examines to assure that it cannot cause or contribute to a hazard.
Independence Analysis	Verify that safety-critical computer software components(CSCs) should be independent of non-critical functions independence analysis is a way to verify it.
Design Data Analysis	Evaluates the description and intended use of each data item in the software design
Design Interface Analysis	Verifies the proper design a software component interfaces with other components of a system
Design Traceability Analysis	This analysis ensures that each safety-critical software requirement is covered and that an appropriate criticality level is assigned to each software element
Dynamic Flow Graph Methodology	The dynamic flow graph methodology (DFM) is an integrated methodological approach to modeling and analyzing the behavior of software-driven embedded systems for the purpose of reliability/safety assessment and verification[45].
Rate Monotonic Analysis(RMA)	Rate monotonic analysis is a mathematical method for analyzing a set of real-time tasks that applied to priority inversion, task interactions,and aperiodic tasks.
The requirement state machine(RSM)	The requirement state machine(RSM) is sometimes called the Finite State Machine(FSM). An RSM is a computation model or depiction of a system or subsystem, showing states and the transitions between the states.

We examine the software system FMEA/FTA to review software safety in this paper. In this phase of software development, the objectives of the software safety program include the identification of important components of software and functions, the use of appropriate high-level analysis methods for these components and functions that contribute to avoidance or reduction of potential risks. The software development team defines necessary software components and functions to create a working system detecting all software safety requirements. The software risk analysis method can estimate the accuracy or criticality level for each software component or function. The criticality level depends on potential risks that can arise from a detailed defect in a software component or function. A higher level of analysis is required if there is a high level of criticality. To achieve these goals, there is a need for an extended fault tree analysis to detect specific software components or functions generating software scenarios that may lead to probable hazards. System-level SFMEA analysis identifies a wide range of possible failures. The dedicated critical software components or functions are based on maximum probable risk developed in failure trees due to potential software problems.

4.5.1.Fault Tree Analysis

At this phase of development, the existing fault tree is revised, so that it includes specific software modules. This typically involves a replacement based on structural analysis of software architecture.

4.5.2.System SFMEA

As mentioned, SFMEA can be done at different design phases to match the system design process. The system-level SFMEA during software safety architecture analysis is meant to examine the structure and protect the underlying system design, and the system risk test results and PHA are essential to system-level SFMEA inputs.

4.6. Software Coding and Detailed Design Phase

As can be seen in Figure 3, the following points are considered in the software architecture phase. Evidently, we will review the detailed FMEA/FTA and defensive programming to examine the software safety. Software safety detailed analysis includes: (a) Software detailed FMEA/FTA; (b) Software safety code analysis; (c) Defensive programming; (d) Formal inspections of safety-critical code; and (e) Code data and logic analysis.

At this phase of software development, the objectives of the software safety program include a detailed analysis of software design and implementation to help ensure that the required level of safety is reached. The subsystem interfaces may be analyzed to identify the potential risks associated with subsystems, and the analysis may be checked for intrinsically insecure situations caused by I/O scheduling, out-of-order events, lateral environments, etc. Detailed SFTA and SFMEA are two methods that can be used to achieve the goals of the software safety program. The high-level software risk analysis using FTA in architecture and requirements phases can be further developed to analyze the inherent risks detected in software scenarios and variables. Detailed FMEA can be performed for all or a majority of software modules at risk by tracking intrinsic failures in input variables and logical process. In this section, FTA and FMEA procedures have been described and compared in detailed design and code level.

Table. 3. The Methods of software coding and detailed design phase

Method	Task
Software detailed FMEA/FTA	detailed software FMEA method is a systematic review of real-time software of a product in order to identify the effects of potential faults in unique variables used in the software.
Defensive programming	Using the detailed design of software and existing codes, the fault tree can be extended to the detection of low-level software components, which are directly allocated to high-level software components in the previous fault tree.
Formal Inspections of safety-critical code	defensive programming is a set of methods, techniques, and algorithms designed to prevent errors that lead to failure.
Code data and logic analysis	Formal inspections are one of the best methodologies available to evaluate the quality of code components and program sets.

4.6.1.Detailed Software FTA

The existing fault tree links high-level components and functions to potential risks from software architecture phase. Using the detailed design of software and existing codes, the fault tree can be extended to the detection of low-level software components, which are directly allocated to high-level software components in the previous fault tree. These low-level software components can be considered as the core safety, and any software risk avoidance requirement can then be detected. When the results of detailed software FMEA method are available, FTA and FMEA results can be compared for compatibility and integrity. The same goal is pursued when FTA is used for software. Software risk becomes the top event with system elements, processes, procedures, and so on. In this method, the templates are given for each large structure in a given program, and the fault tree for the program is then generated by combining these templates. Unfavorable events are detected to perform software FTA.SFTA, which is considered as a verification tool, is used to identify the defective site in software design.

4.6.2.Detailed Software FMEA Method

As stated above, the detailed software FMEA method is a systematic review of real-time software of a product in order to identify the effects of potential faults in unique variables used in the software. To help perform the FMEA, various maps have been developed to design all input, output, local, and global software variables in accordance with the software programs. Therefore, each variable, whether input or output, has its own design. Each of the input variables is another hardware variable or output of the program.

4.6.3.Integration of Results

When FMEA is implemented on each of the software modules, output variables are used to present a project between these modules. The effect of the fault on the input of a module is followed by its correspondent input fault mode in the next module. Such tracking of the mode or impact of variables' faults will be repeated until the variables reach a high level of the program. To prove this, the tracking creates software sequences linking software modules and acquired data variables to final inputs. A fault map for risks is detected when a set of fault effects are followed up to high-level programs. Detailed FMEA of software is similar to the hardware FMEA with the exception that the variables replace the signals and electronic hardware signal paths. Finally, when detailed FMEA is finished, a map of possible high-level risks for top-level key variables will be provided. Top-level key variables are those necessary and sufficient to activate the potential risk of software mode. If detailed FMEA detects a probable failure mode that follows potential risks, then the false implementation of software safety will be detected and corrected. Similar to system-level FMEA, software design defects should be detected and the required information updated. Safety test information is updated using other safety software specifiying the demands in a detailed design and coding step.

4.6.4.Defensive Programming

In addition to FTA and FMEA methods applied at this phase of software development, accepted or extended programming guidelines often indicate that the developers implement defensive programming techniques. The defensive programming of software writing style can handle anything that happens on it; in other words, defensive programming is a set of methods, techniques, and algorithms designed to prevent errors that lead to failure. A strategy should be considered at the beginning of the program to deal with defects, faults, and errors. To write the program, this paper proceeds as follows: the essential functions of the code are separated from unnecessary functions to reduce the likelihood of a possible fault leading to potential risks. The use of 0 and 1 logics to interpret the scenarios or decision-making results of original safe functions is not recommended due to quantitative reasons.

4.7. Software Verification and Validation Phases

In this phase of software development, the following points can be mentioned based on safety. The goal of the software safety program is to implement the test plans ensuring that the software has met all the software safety requirements. This typically includes unit testing and an integration test. Testing programs prove that fault detection and inhibition have been fulfilled as expected. The documents that are written in this phase are summarized in Table 4.

Table. 4. Documentation for the verification and validation phase

Document	Software safety
Integration Test Plan	Testing should exercise the connections between safety-critical units and non-critical units or systems.
System Test Plan	Extreme, but possible, environments should be tested (heavy load, other stressors) to verify the system continues to function safely within all plausible environments.
Test Reports	Verify test was completed as planned and all safety-critical elements were properly tested. Report results of testing safety-critical interfaces versus the requirements outlined in the Software Test Plan. Any safety-critical findings should be used to update the hazard reports.
Configuration Management Audit Report	Verify that the configuration management system is properly used, especially for safety-critical elements.
Formal Inspections Report	All defects related to safety-critical elements should be considered major (must fix).
Analysis Reports	Identification of any safety-related aspects or safety concerns.
Problem or Failure Reports	Problem or Failure reports should be reviewed for any safety implications. Any corrective action should be verified to not cause an additional hazard or to adversely impact any other safety-critical software or hardware.
Traceability Matrix	Verify that requirements are traceable all the way into the test cases. Verify that all safety requirements have been adequately tested.

The following tests are performed at this stage. Test simulation, load testing, stress testing, boundary value tests, test coverage analysis, functional testing, performance monitoring, disaster testing, resistance to failure testing, red team testing, regression testing.

5. Case Study

We applied the results of our approach to part of a real Cyber-physical system known as Data and Command Unit.

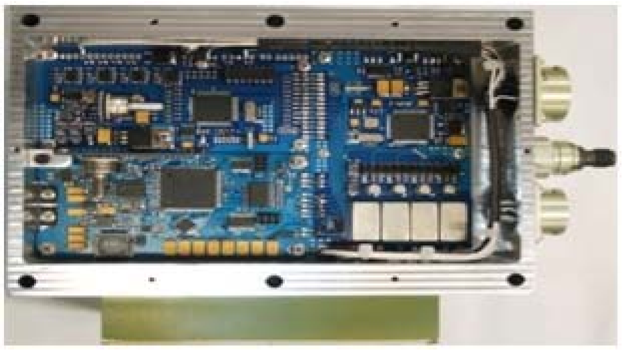


Fig. 5. The data and command unit system.

In Figures5 and 6, part of a command system is shown; the most important goal of the Data and Command Unit is to issue commands for separation of the nose, engine, and parachutes based on the simulated time and flight profile. This section requires the detection of the movement start and must, in fact, receive the Start signal as a trigger. The start signal is produced by the simultaneous cut of cord and compression of mass and spring switch. It is a command provided to start the operations of the two system processors, which use data from pressure sensors and the timeline of their internal timers to perform their operations. The movement start in cord cable is detected via separation of male and female parts. In this way, the two bases in the female part of this connector, which remains on the ground after the launch, are connected and soldered. In the male part located on the rocket, one of the bases is connected to the ground, and the other base is considered as the movement start signal of cord movement in addition to its connection to Vcc (Maximum Power Supply Value) by a resistor. When the male and female parts of cord connector are connected, the output signal is 0; however, through the separation of male and female parts of cord connector, the command signal is connected to Vcc by a resistor and equals to1. Spring-mass system is, in fact, an acceleration-sensitive system that activates a bi-directional electrical switch by sensing a minimum of acceleration. If the acceleration exceeds a certain threshold, the mass will change form downward with a certain value to activate the switch located under the mass. The spring-mass system has been designed to become active in a minimum acceleration of 2g, indicating that if an acceleration lower than this value is applied to the load, the spring-mass system will not be activated. For example, if a minor acceleration is introduced into the system when transporting the system or during the tests, a lack of function of the spring-mass system is ensured. By connecting the common side of the switch to the ground and receiving output from the normally closed part, the output signal becomes 0 before the launch, which turns into1after the launch by the Pull Up resistor. To achieve higher reliability, the outputs of these mechanisms are added to each other by AND via hardware arrangement, which is sent as the movement start signal to different parts. Schematic representation of connections in this part before and after the launch is shown in Figure 7.

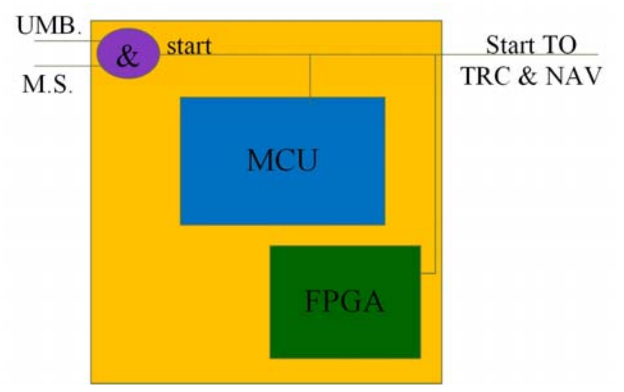


Fig. 6. Detection of data and command unit of the movement start.

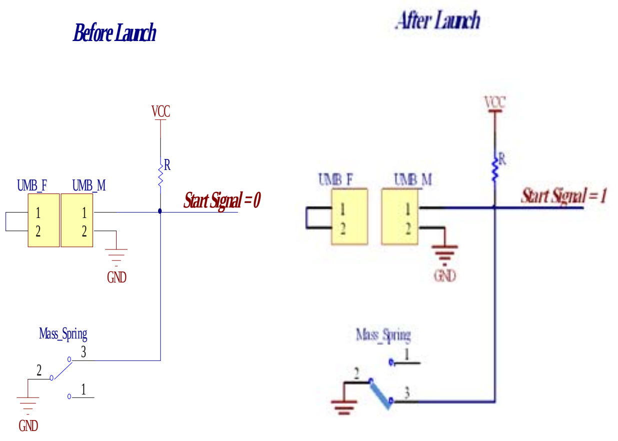


Fig. 7. The block diagram of the movement start detection unit.

Using the above connections, the start signal is equal to1before the launch and turns0 if the umbilical cord connector is separated and both the mass and spring are compressed. The potential risk of such a system is that the movement start signal is not produced, which is an unwanted system operation. The unwanted operation of the system can lead to undesirable system behavior that could potentially be dangerous. Some causes of unwanted system operation are shown in Table 5. A system lacking safety strategy has a high potential risk because an unwanted event may occur. However, the potential risk decreases after incorporating appropriate safety features specified by the safety strategy.

5.1. Conceptual Phase

As mentioned, the input of the requirements analysis phase is taken from the conceptual design phase where the PHA&STPA is applied to the command system, and the results are presented as follows (Figure8 and Table 5).

Motor pressure is monitored to increase system reliability. The safety strategy of 1.2 in table 5is described as follows. By detecting an average engine pressure higher than a bar, its relevant t1 time is recorded. If the engine pressure remains more than A bar for B seconds after t1, then t1 time is considered as the movement start time for micro (A and B are determined based on engine type and processing conditions). With this proposed strategy, the movement start signal can be triggered as follows. The movement start is detected via the activation of (spring-mass) and (cord) for two seconds with or without the proposed safety strategy. By detecting each of these two cases for movement signal, the next item is ignored. The movement start is detected by one of the logic methods: (a) Activation of spring-mass and cord eruption; or (b) Pressure change based on the proposed safety strategy.

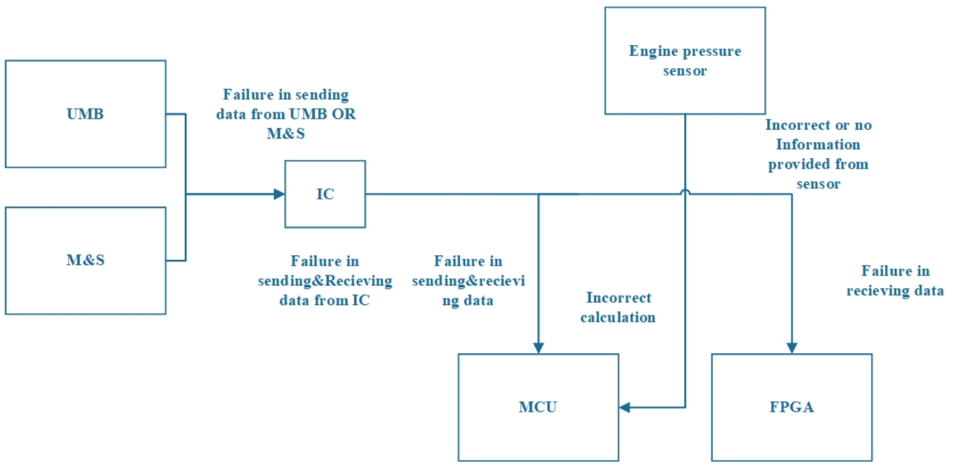

UMB
M&S
Failure in sending data from UMB OR M&S
IC
Engine pressure sensor
Incorrect or no Information provided from sensor
Failure in sending&Recieving data from IC
Failure in sending&recieving data
Incorrect calculation
Failure in recieving data
MCU
FPGA

Fig. 8. Identification of unsafe control actions (STPA).

Table. 5. PHA to identify the movement start in the conceptual design phase

Hazard Name	Causes	Hazard Risk	Safety Strategy	Revised Hazard Risk
Failure in the movement start	1. Physical damage of mass and spring. 2. Crash and damage of pins. 3. Improper disconnection of cord on the launcher. 4. The signal sending function does not work correctly. 5. The connector between the cords, mass, spring, and the mainboard is disconnected. 6. The wire or connector is opened because of impact during or before throwing. 7. The power supply IC is disconnected. 8. IC is out of order.	high	1. The integrity of mass and spring must be checked before final assembly/ 1.1. Use of two spring-mass systems 1.2. Sending the command issue signal by checking the engine pressure 2. The shipment should move in safe conditions and not be hit in any of its parts. 3. There should be a safe mechanism to unplug the cord on the launcher. 4. Use of fault-tolerant system approaches in the software (multiple versions). 5. The connector should withstand the applied acceleration. 6. The tightness of the connector at the moment of assembly should be ensured. 7. Soldering of the connector should be such that it is not disconnected by the applied vibration before and after throwing off the wire.	Low

5.2. Software Requirement Phase

Considering the mentioned cases in section 4.4, the methods in this section are shown by a real case study (a part of the data and command unit system).

With respect to PHA, which was derived from the conceptual design phase, software hazards led to hazards according to Table5. Figure 9 shows the software fault tree analysis (SFTA).

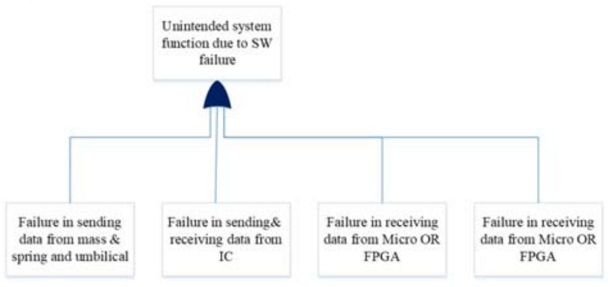

Unintended system function due to SW failure
Failure in sending data from mass & spring and umbilical
Failure in sending& receiving data from IC
Failure in receiving data from Micro OR FPGA
Failure in receiving data from Micro OR FPGA

Fig. 9. The Software fault tree in the requirements analysis phase.

For the software system of the mentioned system, the software fault tree in Figure 9 shows potential detected software faults. For each of these potential software faults, the high level of software safety requirement is highlighted.

Table. 6. Software safety requirements

Req. No.	Software Safety Requirement
SW-SAFETY-1	The data are sent in two ways (multiple versions).
SW-SAFETY-2	If the signal is not sent by mass & spring and umbilical or Engine Pressure, send an alert signal.
SW-SAFETY-3	Using single-version programming techniques, multi-programming techniques (software fault tolerance techniques) to prevent incorrect calculation of micro
SW-SAFETY-4	Sending an alert message if the ICs fail.

For each of the possible software faults, the lowest level of FTA is shown in (Figure 9). The software safety requirements are set and during the hazard test, the requirements are developed for real testing of the system under a possible hazard. The results of this test specify the deviation of the error response times and the level required by the system.

5.3. Analyzing Software Safety Requirements in DCU Systems

According to the methods described in Section 3.5, methods 1 and 2 are used in this paper.

- Top-down analysis of software safety requirements
- Analysis of critical requirements

Figure 8 (Top-down analysis) and Table 7 (Analysis of critical requirement) present results and requirements of the safety analysis in this phase. Table 4 presents the updated requirements.

Table. 7. Modified Software Safety Requirements

REQ. No	Requirements
SW-SAFETY-1	The system sending warning signal must detect the signal deviation in TBD ms (mass and spring). The system should send an alert signal after ms if the signal is not sent by mass & spring and umbilical or Engine Pressure).
SW-SAFETY-2	The system sending warning signal must detect the signal deviation in TBD ms (IC). (The system should send an alert signal after ms if the message is not sent from IC).
SW-SAFETY-3	All software must comply with MISRA C programming guidelines.
SW-SAFETY-4	The software failure management routine should start turning off the system controller immediately after detecting the failure.
SW-SAFETY-6	The data received by the two data transmission pathways should be compared. In case of a difference, a warning message must be sent.

For the movement start detection system, safety results and requirements are reviewed for compatibility and safety in Tables 2 and 3, as well as Figure 6. Table 4 shows the updated requirements. SW-Safety-1 and SW-Safety-2 conditions are obtained with values that are directly specified. The SW-Safety-3 and SW-Safety-4 conditions are added to determine the system behavior after identifying a defect. SW-Safety-5 shows that the software must follow the programming instructions of Motor Industry Software Reliability Association (MISRA) to help ensure that the best programming techniques are employed.

5.4. Software Architectural Design

To help understand the analytic methods presented in this section, software architecture for the movement start is shown in Table 8. This software architecture must comply with detected safety requirements specified in Table 7, and in some cases, it should involve specific software modules (warning message system). This architecture includes adjusting the micro timer and checking the micro pin inputs in the initial state, and there is also the main loop and a final mode. The information in this Table is written according to Table 6.

Table. 8. The software architecture of movement start

Init:
Check input of micro pins
Set the timer
Main loop:
Receive Sensor Information
Diagnose Sensor Input
Alert message system (sensor)
Detect motor mode
Receive m&s and umb information
Alert message system (m&s and umbilical)
Send the movement start signal
Final:
Finish (the movement start signal is sent)

5.4.1.Fault Tree Analysis

As discussed in the previous sections, at this phase of development, the revised fault tree in which the fault tree includes certain software modules typically involving the replacement of software part contained in the fault tree (a point that is essentially based on software knowledge but is not present in structural software) is substituted with a new subtree according to structural analysis of software architecture. The developed subtree is compared with the old one to ensure that no knowledge has disappeared. For the software architecture of movement start, the software part of the failure tree depicted in Figure 10 is replaced with a tree developed by identifying the immediate causes of the main software event. This tree is created by traversing the software architecture shown in the Figure to detect software designs of software components. The event description in the tree is measured based on Table requirements. Part of the revised tree is presented in Figure 10.

5.5. Coding and Detailed Design Phase of Software

As shown in Figure. 11, the main cause of system failure is the failure in engine turnoff detection. Accordingly, given the higher sensitivity level of this module, it has been examined in more detail.

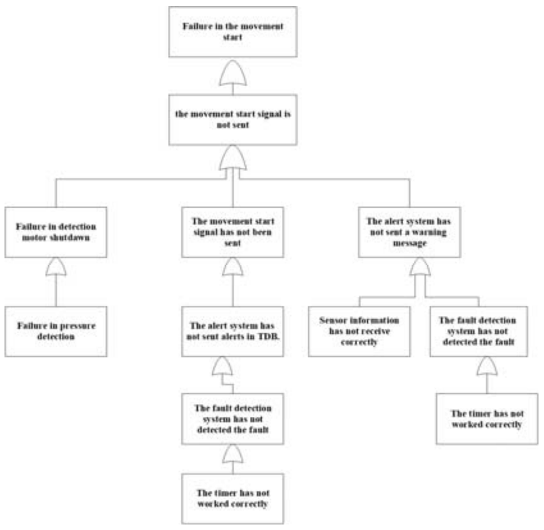

Failure in the movement start
the movement start signal is not sent
Failure in detection motor shutdawn
The movement start signal has not been sent
The alert system has not sent a warning message
Failure in pressure detection
The alert system has not sent alerts in TDB.
Sensor information has not receive correctly
The fault detection system has not detected the fault
The fault detection system has not detected the fault
The timer has not worked correctly
The timer has not worked correctly

Fig. 10. Revised Fault Tree.

```
Determine Engine Mode (Boolean Flag 1, Boolean Flag 2)
{
Enumerated System State = (Normal, off);
System state = Lookup state (Flag 1, Flag 2)
If (Engine state = off)then
Call off Task ()
}
```

Fig. 11. The pseudo code for determination of engine mode.

5.6. Detailed SFMEA Method

As described in Section 5.2, to help the Detailed SFMEA, various maps are created in order to outline all input, output, local, and global variables in accordance with software programs. As stated, when an FMEA is executed in each of the software modules, the output variables are used to present a design between these modules. Finally, SFMEA is shown in the design phase in Tables 9-11.

Table. 9. Variables map

Variable/Routine	Read Sensor Information	Diagnose sensor input	Alert message system	Detect motor mode	The movement start signal	Alert message system	Determine relays state	Diagnose sensor input	Alert message system
Variable-1	Output	Input							
Variable-2			Input	output	output				
Variable-3						Input			
Variable...	…	....	...	...	...	....	...	...	...
Variable-n							Output	Input	Output

Table. 10. Detailed SFMCEA

Software Element	Failure Modes	Local Effect	System Effect	Potential Severity	Recommendation
Sensor input	Fails to execute	No Sensor signal read	The system uses the last read signal value and the processor will not detect the failure. If the output is different from the calculated output of the system, it may cause a mishap.	10	In specific time units, the specified amount of data is read and the average is calculated and compared with the average of previous times if the difference is greater than the specified value, and the processor must be given an alert.
Sensor input	Erroneous	Some or all of the sensor signals are incorrect	The processor takes the wrong information and if it is within range, it does not recognize the wrong inputs and may cause a mishap.	10	It may be due to a problem in the analog to digital section, sensor failure, the computational error of the processor, and so on. For hardware bugs, regular checking of components and systems, and for fault detection and diagnostic methods for software bugs.

Table. 11. Detailed SFMCEA

variables	Failure Modes	Software modules affected	Local effect	System effect	Potential Severity	Recommendation
Flag1	Normal and off mode is not correctly detected	Determine engine mode	The engine state does not correctly detect and cause a false action.	The system will shut down and thus cause loss of performance	8	Replace Boolean flags with the enumerated data type. Use of fault detection and diagnostic methods
Flag 2	Normal and off mode is not correctly detected.	Determine engine mode	System mode may be normal when it should be failed, so no call when there is a call to off	The system may provide an incorrect output	10	Replace Boolean flags with the enumerated data type. Use of fault detection and diagnostic methods.

Following the solutions presented in Tables 10 and 11, using the Delphi method, the investigations show the potential severity reduction, all failure modes, and system acceptability.

5.7. Use of Operational Strategies to Reduce Risk Severity in the Movement Start

According to the methods mentioned in section 4.2, the methods used to reduce the risk severity are explained in this section. As noted, data transmission pathways are doubled to reduce risk, improve performance, and promote the reliability of data transmission pathways. To prevent the occurrence of common cause failure, two different redundant mechanisms are used for sending and communicating. Moreover, to provide early warnings, an alert system was considered, and as mentioned, safety requirements were also considered. Multiprogramming and single programming techniques are the software fault tolerance methods used to reduce risk, which is shown in Table 6 as Software Safety Requirements.

Following the methods outlined in this section to reduce risk severity and the Delphi method of success, it is seen that severity has decreased.

6. Conclusion

This paper presents software safety methods in the software lifecycle and then examines the effective application of these processes on one subsystem of aerospace systems, namely the data and command unit system. The successful implementation of the software safety program is dependent on the experience of the involved individuals, selection and application of safety analysis, and evaluation methods during the software development cycle. For further research, we suggest other methods proposed in this paper (those presented in Tables 2, 3 and 4), as well as analysis and noise detection in software systems and the safety in multi-thread programming methods. Also, the maintenance policy based on risk and safety criteria can be considered in the next study.

References

[1] Farsi, M. A., "Develop an Adaptive Prognostic Approach for RUL Estimation", Technical Report, Aerospace Research Institute (Ministry of Science, Research and Technology) (2016).

[2] Van Driel, W.D.; Schuld, M.; Wijgers, R.; Van Kooten, W.E.J., "Software reliability and its interaction with hardware reliability", In thermal, mechanical and multi-physics simulation and experiments in microelectronics and Microsystems (eurosime), IEEE 15th international conference, pp. 1-8 (2014).

[3] Kooli, M.; Kaddachi, F.; Di Natale, G.; Bosio, A.; Benoit, P.; Torres, L., "Computing reliability: On the differences between software testing and software fault injection techniques", Microprocessors and Microsystems, vol. 50, pp.102-112 (2017).

[4] Park, J.; Kim, H.J.; Shin, J.H.; Baik, J., "An embedded software reliability model with consideration of hardware related software failures", In Software Security and Reliability (SERE), IEEE 6th International Conference, pp: 207-214 (2012).

[5] Lutz, R.R., "Software engineering for safety: a roadmap", ACM In Proceedings of the Conference on the Future of Software Engineering, pp. 213-226 (2000).

[6] Habli, I.; Hawkins, R.; Kelly, T., "Software safety: relating software assurance and software integrity", International Journal of Critical Computer-Based Systems, vol. 1 no. 4, pp. 364-383 (2010).

[7] Wong, W.E.; Debroy, V.; Restrepo, A., "The role of software in recent catastrophic accidents", IEEE reliability society 2009 annual technology report, vol. 59, no. 3 (2009).

[8] Pertet, S.; Narasimhan, P., "Causes of failures in Web applications", Carnegie Mellon University: Parallel Data Lab, Technical Report CMU-PDL-05-109 (2005).

[9] Bella, M. B.; Eloff, J. H., "A near-miss management system architecture for the forensic investigation of software failures", Forensic science international, vol. 259, pp. 234-245 (2016).

[10] Oveisi, SH; Farsi, M.A, "Software Safety Analysis with UML-Based SRBD and Fuzzy VIKOR-Based FMEA", International Journal of Reliability, Risk and Safety: Theory and Application (ijrrs), vol. 1, pp.1-9 (2018)

[11] NASA,1987, Software Safety: NASA Technical Standard, NASA-STD-8719.13A.

[12] Albericoet, D. and et al. "JSSC Software System Safety Handbook; A Technical & Managerial Team Approach", (1999).

[13] Department of Defense, System Safety Program Requirements, MIL-STD-882C (Department of Defense). 1984.

[14] RTCA, SW Considerations in Airborne Sys. and Equip. Cert., RTCA/DO-178B (RTCA);1994.

[15] MOD, Requirements for Safety Related Software in Defense Equipment; Part 1: Requirements; Part 2: Guidance, MOD DEF STD 00-55 (Ministry of Defense); 1997.

[16] IEC, International Standard; Functional Safety of Electrical /Electronic /Programmable Electronic Safety-Related Systems – Part 3: Software Reqs., IEC 61508-3 ;1998.

[17] MISRA, Development Guidelines for Vehicle Based Software (MISRA, November 1994).

[18] Kuettner Jr, H. D.; Owen, P. R, "Definition and Verification of Critical Safety Functions in Software", In Proceedings of the International System Safety Conference (ISSC), pp. 337-346 (2001).

[19] FAA system safety handbook, chapter; system software safety, December 2000.

[20] NASA-STD-8719.13A NASA Software Safety Standard, September 1997.

[21] Swarup, M. B.; Ramaiah, P. S., "A software safety model for safety-critical applications", International Journal of Software Engineering and Its Applications, vol. 3, no. 4, pp. 21-32 (2009).

[22] Hiraoka, Y.; Murakami, T.; Yamamoto, K.; Furukawa, Y.; Sawada, H., "Method of Computer-Aided Fault Tree Analysis for High-Reliable and Safety Design", IEEE Transactions on Reliability, vol. 65, no. 2, pp. 687 – 703 (2016).

[23] Farsi, M. A., Principles of Reliability Engineering (2016).

[24] NASA-STD-8719.13A, NASA Software Safety Standard;1997.

[25] NASA Software Management Guidebook, NASA-GB-001-96, November, 1996.

[26] Mastrangelo, C., "Effective FMEAs: Achieving Safe, Reliable, and Economical Products and Processes Using Failure Mode and Effects Analysis", Journal of Quality Technology, 44.4: 395 (2012).

[27] Wu, F. J.; Kao, Y. F.; Tseng, Y. C., "From wireless sensor networks towards cyber physical systems", Pervasive and Mobile computing, vol. 7, no. 4, pp. 397-413(2011).

[28] Murali, D. V., "Verification of Cyber Physical Systems", Unpublished Master of Science Thesis. Virginia Polytechnic Institute and State University, Blacksburg (2013).

[29] Oveisi, SH.; Ravanmehr, R., "Analysis of software safety and reliability methods in cyber physical systems", International journal of critical infrastructures, vol. 13, no. 1, pp. 1-15 (2017).

[30] NASA Program and Project Management Processes and Requirements, NPG 7120.5A, (1998).

[31] Czerny, B. J.; D'Ambrosio, J. G.; Murray, B. T.; Sundaram, P., "Effective application of software safety techniques for automotive embedded control systems", SAE transactions, pp. 194-204 (2005).

[32] Oveisi, SH.; Ravanmehr, R., "Safety and reliability of software", Sanagostar (2017).

[33] Czerny, B. J.; D'Ambrosio, J. G.; Jacob, P. O.; Murray, B. T.; Sundaram, P., "An Adaptable Software Safety Process for Automotive Safety-Critical Systems", SAE Technical Paper (2004).

[34] Oveisi, SH.; Ravanmehr, R., SFTA-Based Approach for Safety/Reliability Analysis of Operational Use-Cases in Cyber-Physical Systems", Journal of Computing and Information Science in Engineering, vol. 17, no. 3 (2017).

[35] Li, S.; Duo, S., "Safety analysis of software requirements: model and process", Procedia Engineering, vol. 80, pp. 153-164 (2014).

[36] Johansson, C., "On System Safety and Reliability in Early Design Phases: Cost Focused Optimization Applied on Aircraft Systems", Doctoral dissertation, Linköping University Electronic Press (2013).

[37] Jet Propulsion Laboratory, Software Systems Safety Handbook.

[38] Lawrence, J. D., "Software safety hazard analysis (No. NUREG/CR--6430)", Nuclear Regulatory Commission (1996).

[39] Oveisi, SH; Farsi, M. A, "Software Assurance for aerospace systems", Technical Report, Aerospace Research Institute (Ministry of Science, Research and Technology (2018).

[40] Plattsmier, G.; Stetson, H., "Autonomous real time requirements tracing", In IEEE Aerospace Conference, PP. 1-9 (2014).

[41] Department of Defense, Software System Safety Handbook, A Technical & Managerial Team Approach, Dec. 1999, by Joint Software System Safety Committee.

[42] Pham, H., "System Software Reliability", in Springer series in Reliability Engineering, vol. 79, London, Springer, pp. 45-52 (2006).

[43] Cinque, M.; Cotroneo, D.; Pecchia, A., "Event logs for the analysis of software failures: A rule-based approach", IEEE Transactions on Software Engineering, vol. 39, no. 6, pp. 806-821 (2013).

[44] Garrett, C. J.; Guarro, S. B; Apostolakis, G. E., "The dynamic flowgraph methodology for assessing the dependability of embedded software systems", IEEE Transactions on Systems, Man, and Cybernetics, vol. 25, no. 5, pp. 824-840 (1995).