DEVELOPING SAFETY-CRITICAL SOFTWARE REQUIREMENTS FOR COMMERCIAL REUSABLE LAUNCH VEHICLES

Daniel P. Murray[(1)] and Terry L. Hardy[(2)]

[(1)]Federal Aviation Administration, Office of Commercial Space Transportation, 800 Independence Avenue, S.W., Room 331, Washington, DC, 20591, USA, Daniel.Murray@faa.gov
[(2)]National Aeronautics and Space Administration Goddard Space Flight Center, Mail Code 302, Greenbelt, MD 20771, USA, Terry.L.Hardy@nasa.gov

ABSTRACT

A number of inventors and entrepreneurs are currently attempting to develop and commercially operate reusable launch vehicles to carry voluntary participants into space. To reduce the risk to the public in the operation of these vehicles, a launch vehicle operator typically performs analyses to identify safety measures and develop safety requirements. The focus of these safety efforts has historically been to develop and implement safety requirements for hardware systems and subsystems. However, software and computing systems are increasingly being used in launch vehicles to control or monitor safety-critical systems, compute or transmit safety-critical data, and detect and mitigate faults. Therefore, identifying the hazards, assessing the risks, and implementing valid safety requirements for these software elements are becoming critical to public safety. This paper presents lessons learned from the failure of space vehicle systems that can be applied to the development of safety-critical software requirements for commercial reusable launch vehicles. The paper also describes a software system safety process recommended by the Federal Aviation Administration (FAA) for developing safety requirements to reduce the risks from the use of software in reusable launch vehicle operations.

1. INTRODUCTION

Software and computing systems are becoming increasingly important in assuring the safe operation of reusable launch vehicles. Software and its associated computing systems (computer system hardware and firmware) are used in on-board and ground systems to support safety-critical functions such as guidance, navigation, and health monitoring. Software is also used to produce safety-critical data and to assist in mitigating system risks. The launch vehicle operator must therefore identify, characterize, and analyze the hazards and mitigate the risks associated with the use of software and computing systems on commercial space launch vehicles to reduce the risk to the public.

Although software safety is part of the launch vehicle system safety effort that includes hardware and other factors, key differences exist between hardware and software. Hardware, including computer system hardware and associated equipment, fails most often because of such factors as deficiencies and variability in design, production, and maintenance. However, software does not fail in the conventional sense – software does not break, wear out, or fall out of tolerance like hardware. Software faults are primarily systematic, not random, and are primarily caused by design faults, particularly in defining and interpreting requirements. Deficient requirements are the single largest factor in software and computing system project failure. Deficient software requirements have contributed to a number of space vehicle failures, as described in the following section.

2. SPACE VEHICLE FAILURES

An inadequate requirements development process has been identified as a contributing cause to a number of high-profile space vehicle failures. Examples of some of those failures are described below. Although not all failures were of spacecraft, the lessons learned from such failures are instructive to those building and operating launch vehicles.

2.1 Ariane 5 launch vehicle

On June 4, 1996, the Ariane 5 launch vehicle veered off course and broke up approximately 40 seconds into launch. The vehicle started to disintegrate because of high aerodynamic loads due to an angle of attack greater than 20 degrees. This condition led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher. This improper angle of attack was caused by full nozzle deflections of the solid rocket boosters and the Vulcain main engine. The on-board computer software commanded these nozzle deflections based on data received from the active Inertial Reference System. Ultimately, these improper deflections were found to have been the result of specification and design errors in the Inertial Reference System software, including improper error handling (an unexpected horizontal velocity component led to an overflow condition which was not handled properly by the software). Contributing to the failure was the fact that this software was reused from the Ariane 4 program, including the exception handling code used in the Inertial Reference System. The source of the fault occurred in a function that was not required for Ariane 5, but rather was a function carried over from the Ariane 4 software. There was a belief by the development team that faults would be due to a random

hardware failure, handled by redundancy in the hardware. However, because the problem was a requirements problem and not due to random failure, both the primary and backup Inertial Reference Systems shut down nearly simultaneously from the same cause. In addition, no end-to-end tests were conducted to verify that the Inertial Reference System and its software would behave correctly when being subjected to the countdown sequence, flight time sequence, and the trajectory of Ariane 5 [1].

2.2 Phobos 1 spacecraft

The Phobos 1 spacecraft was launched on July 7, 1988, on a mission to conduct surface and atmospheric studies of Mars. The spacecraft operated normally until routine attempts to communicate with the spacecraft failed on September 2, 1988, and the mission was lost. Examination of the failure showed that a ground control operator had omitted a single letter in a series of digital commands sent to the spacecraft. The on-board computer mistranslated this command and started a ground checkout test sequence, deactivating the attitude control thrusters. As a result the spacecraft lost its lock on the sun. Because the solar panels were pointed away from the sun, the on-board batteries were eventually drained until all power was lost. A significant contributor to the failure was a lack of requirements regarding the human and software interface [2].

2.3 Mars Polar Lander

The Mars Polar Lander (MPL) was launched on January 3, 1999. Upon arrival at Mars, communications ended according to plan as the vehicle prepared to enter the Martian atmosphere. Communications were scheduled to resume after the lander and the probes were on the surface. However, repeated efforts to contact the vehicle failed. The cause of the MPL loss was never fully identified, but the most likely scenario was that the problem occurred during deployment of the three landing legs during the landing sequence. Each leg was fitted with a Hall Effect magnetic sensor that generates a voltage when the leg contacts the surface of Mars. The descent engines were to be shut down by a command from the flight software when touchdown was detected. It is believed that the software interpreted spurious signals generated at leg deployment as valid touchdown events, leading to premature shutdown of the engines at 40 meters above the surface of Mars, resulting in the vehicle crashing into the surface. Although it was known that a possible failure mode existed whereby the sensors would falsely detect that the vehicle had touched down, the software requirements did not account for this failure mode and the software was not programmed to avoid such an occurrence [3].

2.4 Failure Trends and Lessons Learned

In addition to these specific failures, recent analyses of launch vehicle failure trends have shown that software and computing systems have become a much more frequent cause of failures recently than has occurred in the past. Despite only one failure during the 1950s to 1980s, five software and computing system failures have occurred in both the 1990s and 2000s. The trends are shown in Tab. 1.

Table 1. Worldwide Subsystem Failures by Decade [4]

Subsystem	**1980s**	**1990s**	**2000s**
Propulsion	42%	38%	54%
Guidance and navigation	6%	16%	4%
Electrical	6%	8%	8%
Operational ordnance	2%	8%	0%
Structures	4%	6%	0%
Software and computing	0%	8%	21%
Pneumatics and hydraulics	4%	2%	0%
All other subsystems	0%	0%	0%
Unknown	37%	16%	13%

One lesson learned from specific software and computing system failures and anomalies and from the trend data is that strong launch vehicle safety processes are necessary to prevent future accidents, especially in the development of valid, verified software safety requirements. One such approach that can be applied to the development of reusable launch vehicle software requirements is a software system safety process. Such a software system safety process can help assure the following:

- The software and computing system hazards are identified, described, and characterized
- The software and computing system risk is analyzed and assessed
- Unacceptable risk is mitigated
- The effectiveness of risk mitigation strategies is assessed and monitored
- Changes are monitored throughout the project or program lifecycle

This process is described in detail in [5]. The following summarizes that process.

3. SOFTWARE SYSTEM SAFETY PROCESS

An RLV operator uses a three-pronged approach to ensure that public health and safety and the safety of property would not be jeopardized by the conduct of an RLV mission. The three safety-related elements reflected in this strategy for RLV mission and vehicle operations are as follows:

- Using a logical, disciplined system safety process to identify hazards and to mitigate or eliminate risk.
- Establishing limitations of acceptable public risk as determined through a calculation of the individual and collective risk, including the expected number of casualties (E_c).
- Imposing mandatory and derived operating requirements.

A launch vehicle is a complex and integrated system comprised of hardware, software, human interactions, environmental interactions, and so on. Therefore, a software and computing system safety process should be considered as one part of the larger integrated system safety process.

Fig. 1 shows a software system safety process recommended for launch vehicles. Each of these steps is described below. Note that although this process is presented in a linear, one-pass fashion for ease of discussion, the software system safety process is in fact iterative over the life of the project. Analyses and processes are updated and additional information is obtained as the launch operator discovers new hazards, finds that certain hazards are no longer applicable, makes changes to the system, and better defines the system.

3.1 Software safety planning

The purpose of software safety planning is to define the approach that will aid in producing software that will satisfy launch vehicle system safety requirements. Planning helps ensure that safety is designed and incorporated in from the beginning of the life cycle. Early hazard identification and risk reduction will typically provide the most effective and lowest cost approach to addressing safety concerns. Software safety plans include a System Safety Program Plan, which describes the software and hardware safety tasks and activities, and the Software Development Plan. A Software Development Plan includes management elements of safe software development (organization and responsibilities, policies and procedures, schedule and tasks, etc.) and engineering elements (hazard analyses, verification approaches, configuration management, quality assurance, etc.). Additional information about software safety planning can be found in [5] and [6].

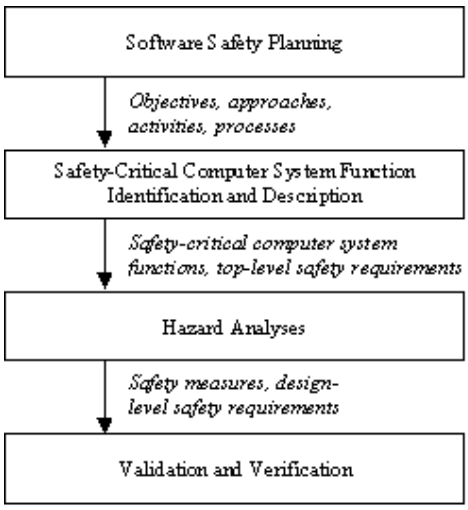

Figure 1. Software system safety process

3.2 Safety-critical computer system function identification

When software is integrated as part of a system to command, control, or monitor safety-critical launch vehicle functions, special measures are required to understand and mitigate safety risks. Therefore, it is important first to identify those launch vehicle functions that are essential to safe performance or operation. Identifying these vehicle functions helps prioritize the safety effort to focus the resources and activities on the most important safety concerns. Some examples of potentially safety-critical launch vehicle functions include the following:

- Operation of a flight safety system to safely abort the flight if the vehicle poses a risk to the public
- Propulsion system control, including rocket engine start or shutdown operations
- Propulsion system health monitoring sensing and display (e.g., pressure and temperature)

Once an operator has identified its safety-critical launch vehicle functions, the operator should then identify the safety-critical computer system functions. Safety-critical computer system functions are essentially those software features that are used to monitor, control, or provide data for the safety-critical functions.

At this stage the operator should also define top-level, or generic, requirements. These requirements are in

general not tied to a specific hazard but rather are derived from knowledge of the safety-critical functions, design standards, safety standards, mishap reports, experience on similar software, and lessons learned from other programs. Some examples of generic requirements include the following:

- Upon detecting an anomaly or failures, the software should remain in or revert to a safe state
- Override commands should require multiple operator actions
- The software should notify crew, ground, or the controlling executive during or immediately after execution of an automated hazardous process

Further examples of generic requirements are provided in [5].

3.3 Software and computing system hazard analyses

Once the safety-critical computer system functions have been identified, an operator should perform analyses to identify the hazards, assess the risks, and identify risk mitigation approaches associated with those functions. In software-intensive systems, mishaps often occur because of a combination of factors, including component failure and faults, human error, environmental conditions, procedural deficiencies, design inadequacies, and software and computing system errors. In such systems software often cannot be divorced from the system where it resides. The launch vehicle operator should therefore first perform a preliminary analysis that considers software hazards on a system or subsystem level as part of a larger system safety effort. An example of such a system would be a flight display, which might include both hardware and software components. An operator can perform these system-level hazard analysis and risk assessments in a manner similar to that used for systems consisting only of hardware. Typical approaches include Preliminary Hazard Analyses and Failure Modes, Effects, and Criticality Analysis. The analysis will result in mitigation measures to reduce risk and system-level requirements to implement those mitigation measures. For example, a mitigation measure for the loss of the flight control display might be to use redundant displays or abort the mission and shut down the propulsion system; a resulting safety requirement would be to develop detailed procedures that specify the abort and shutdown conditions.

In addition to the system or subsystem hazard analysis, the operator should perform software-specific hazard analyses. Software-specific hazard analyses identify what can go wrong, what are the potential effects, and what mitigation measures can be used to reduce the risk. Note however that because of the difficulties in assigning probabilities to newly developed software, the software-specific hazard analysis does not usually include an assessment of the likelihood of a software fault. Typical software-specific hazard analysis techniques include Software Failure Modes and Effects Analysis and Software Fault Tree Analysis. Examples of these analysis approaches are provided in [5] and [7].

An operator's software-specific hazard analyses should consider multiple error conditions. Some of the error conditions to consider are as follows:

- Calculation or computation errors (incorrect algorithms, calculation overflow, etc.)
- Data errors (out of range data, incorrect inputs, large data rates, etc.)
- Logic errors (improper or unexpected commands, failure to issue a command, etc.)
- Interface errors (incorrect messaging, poor interface layout and design, etc.)
- Environment-related errors (improper use of tools, changes in operating system, etc.)
- Hardware-related errors (unexpected computer shutdown, memory overwriting, etc.)

The software-specific analysis should provide specific mitigation approaches for each potential hazard identified. The recommended order of precedence for eliminating or reducing risk in the use of software and computing systems is the same as that for hardware, as follows:

1. Design for minimum risk
2. Incorporate safety devices
3. Provide warning devices
4. Develop and implement procedures and training

Mitigation measures can include, but are not limited to, approaches such as the following [5, 8]:

- Software fault detection (for example, built-in tests, incremental auditing, etc.)
- Software fault isolation (for example, isolating safety-critical functions from non-safety-critical functions, etc.)
- Software fault tolerance (for example, recovery blocks that use multiple software versions of progressively more reliable construction should faults occur, etc.)
- Hardware and software fault recovery (for example, incremental reboots, exception handling, etc.)

Software Failure Modes and Effects Analysis and Software Fault Tree Analysis can be performed on requirements, design, or code. Analyses at lower levels, such as at the code level, provide the most information but also require the most resources. The scope of the analysis will depend on the particular software and development program.

Software and computing system safety analyses should consider safety aspects of the following items:

- Computer system hardware, which includes physical devices that assist in the transfer of data and perform logic operations. Examples include central processing units (CPU), busses, display screens, memory cards, and peripherals.
- Computer system firmware, which is resident software that controls the CPU's basic functioning.
- Computer system software, including operating system software and applications programs.

In addition, because software safety is a systems issue, software and computing systems must be considered with respect to other aspects of the system, such as the following:

- Physical entities whose function and operation are being monitored or controlled, often called the application.
- Sensors (thermocouples, pressure transducers).
- Effectors that take an instruction from the computing system and impart an action on the system (valves, actuators).
- Data communication to other computers.
- Humans who will interact with the system.

Safety is enhanced through the use of layers of protection that include both software- and hardware-specific safety measures.

The output from the software-specific hazard analysis process includes design-level safety requirements based on safety measures developed to mitigate hazards. These design-level requirements could include specific hardware mitigation measures (such as redundant functionality using hardware) or coding requirements that must be implemented. Design-level requirements are statements that can be translated into code without interpretation, or specific mitigations that must be implemented. Examples of design-level requirements include the following:

- Time must not be less than 0
- Oxidizer tank pressurization time must not exceed 30 seconds
- A software function must be developed and used to detect out of range temperature and pressure conditions

The launch vehicle operator should obtain input to the software requirements from environmental requirements, program specifications, facility requirements, tailored generic requirements, and system functionality.

3.4 Software and computing system validation and verification

Software safety is based upon (1) developing valid requirements as a result of efforts to identify, characterize, and reduce the hazards and risks and (2) assuring the integrity of the software and proper implementation of the safety requirements. The validation and verification process is used to manage the set of safety requirements to help ensure the integrity of the software.

Validation determines that the correct requirements are implemented. To do this, the validation effort ensures that each requirement is unambiguous, correct, complete, consistent, testable, and operationally and technically feasible. In addition, the validation process demonstrates that those implementing the requirements (designers, programmers, etc.) understand them.

Verification determines that safety requirements are effective and have been properly implemented. Acceptable methods of verification include the following:

- Analyses: logic analysis, data analysis, interface analysis, etc.
- Inspections: structured technical reviews of software documentation
- Testing: unit tests, interface tests, system tests, etc.

These methods are often used in combination. The acceptability of one method over another depends on the feasibility of the method and the maturity of the vehicle and operations. Further information on validation and verification is found in [5] and [9].

Other significant factors in software and computing system integrity include the following:

<u>Development standards</u>

The launch vehicle operator should identify software development standards that define the rules and constraints for the software development process. These standards should enable uniformly designed and

implemented software components and prevent the use of methods that are incompatible with safety requirements. Software development standards include requirements, design, coding, and safety standards

Configuration management and control

Changes to the software, especially on safety-critical systems, can have significant impacts on public safety. The launch vehicle operator should implement a software configuration management and control process that will at a minimum:

- Identify components, subsystems, and systems.
- Establish baselines and traceability.
- Track changes to the software configuration and system safety documentation.

Quality assurance

Quality assurance is used to verify that objectives and requirements of the software system safety program are being satisfied and to confirm that deficiencies are detected, evaluated, tracked, and resolved. This function is usually performed through audits and inspections of elements and processes, such as plans, standards, and problem tracking and configuration management systems. In addition, the software quality assurance personnel can evaluate the validity of system safety data. The launch vehicle operator should perform quality assurance activities suitable to the objectives of the program.

Anomaly reporting and tracking

Software anomaly reports (also known as problem reports) are a means to identify and record:

- Software product anomalous behavior and its resolution, including failure to respond properly to nominal and off-nominal conditions.
- Process non-compliance with software, requirements, plans, and standards, including improperly implemented safety measures.
- Deficiencies in documentation and safety data, including invalid requirements.

To help prevent recurrence of software safety-related anomalies, the launch vehicle operator should develop a standardized process to document anomalies, analyze the root cause, and determine corrective actions.

4. CONCLUDING REMARKS

Lessons learned from space vehicle failures have shown the importance of developing valid software requirements and verifying that those requirements are effective and have been implemented properly. Software and computing systems are becoming critical to safe launch vehicle operations. Therefore, systematic approaches are needed to define hazards and safety risks, determine appropriate measures to reduce the risks, and then develop safety requirements based on those risk reduction measures. This paper describes an approach recommended by the FAA to develop software safety requirements to reduce the risk to the public during the operation of reusable launch vehicles. However, this approach is not limited to launch vehicles, and should be considered wherever safety-critical software is used.

5. REFERENCES

1. O'Halloran, C., "Ariane 5: Learning from Failure," 23rd International System Safety Conference, San Diego, CA, August 2005.

2. Perminov, V.G., "The Difficult Road to Mars," National Aeronautics and Space Administration Monographs in Aerospace History, Number 15, July 1999.

3. JPL Special Review Board, "Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions," NASA's Jet Propulsion Laboratory, March 22, 2000.

4. "Analysis of Launch Vehicle Failure Trends," Futron Corporation, August 7, 2006.

5. FAA/AST Guide to Reusable Launch and Reentry Vehicle Software and Computing System Safety, version 1.0, July 2006.

6. IEEE STD 1228-1994, *IEEE Standard for Software Safety Plans*, 1994.

7. Dunn, W., *Practical Design of Safety-Critical Computer Systems*, Reliability Press, 2002.

8. Storey, N., *Safety Critical Computer Systems*, Addison-Wesley, 1996.

9. Kit, E., *Software Testing in the Real World*, Addison-Wesley, 1995.