Hazard Analysis Methods for Software Safety Requirements Engineering

Vu N. Tran
Capella University, USA
vu.tran@capella.edu

Long V. Tran
University of Southern California, USA
longvtra@usc.edu

Viet N. Tran
University of Southern California, USA
vntran@usc.edu

Dao N. Vu
University of California at Los Angeles, USA
daonvu@g.ucla.edu

ABSTRACT

The rise of software-based system control in safety-critical systems has made software safety a critical part of a system safety program. The risk of catastrophic software system failure increases with the growth of safety-critical technologies in autonomous transportation systems, airplanes, traffic control systems, medical surgery equipment, nuclear power centers, power grids, human-assist robotics, and military weaponry. Developing software control in safety-critical systems is challenging because the control needs to be reliable and safe. High-profile system failures in recent years, such as the crashes of the 737 MAX airliners, are constant reminders of the risk of software failure in safety-critical systems. The software quality assurance approaches used in software development today are insufficient for created for assuring software reliability but not safety. Developing functionally safe software requires incorporating a risk-driven approach that focuses on hazard identification, hazard risk anticipation, and mitigation. Software safety methods adoption in practice and across mainstream computer science and software engineering curriculums is still limited. Heeding the call for more publications on the practice of software safety, we present an integrated approach to software safety requirements engineering (SSRE). Engineering safety requirements for software is one of the most important steps in building safe software. First, we provide an overview of SSRE. Then we describe three hazard analysis methods that can be incorporated into a software requirement engineering process. Finally, discuss how we combine these distinct methods into a single SSRE approach to support safety-critical systems development.

CCS CONCEPTS

• **Software and its engineering**; • **Software organization and properties**; • **Extra-functional properties-**; • **software safety**;

KEYWORDS

Software safety, safety requirements engineering, software hazards, software hazard analysis

ACM Reference Format:
Vu N. Tran, Long V. Tran, Viet N. Tran, and Dao N. Vu. 2022. Hazard Analysis Methods for Software Safety Requirements Engineering. In *2022 The 5th International Conference on Software Engineering and Information Management (ICSIM) (ICSIM 2022), January 21–23, 2022, Yokohama, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3520084.3520087

ICSIM 2022, January 21–23, 2022, Yokohama, Japan

ACM ISBN 978-1-4503-9551-9/22/01…$15.00
https://doi.org/10.1145/3520084.3520087

1 INTRODUCTION

The deadly consecutive failures of two 737 MAX airliners between late 2019 and early 2020 that took 346 lives once against highlights the importance of software safety in safety-critical systems development [1] [2]. This importance continues to rise as more software are used in place of electro-mechanical devices to control the hazardous functions of autonomous transportation systems, airplanes, traffic control systems, medical surgery equipment, nuclear power centers, power grids, human-assist robotics, and military weaponry. Preventing and controlling software hazards is critical for assuring system safety. The IEEE Standard 1228-1994 [3] describes software hazard as "a software condition that is a prerequisite to an accident" that can cause significant harm to personnel, systems, and the environment if not adequately controlled. Software hazard denotes the contribution of software to the system hazards. Software is less hazardous when they are built to fail safely.

Controlling software hazard remains a big challenge today. According to Hatcliff et. al. [4], this is due to a lack of "the systemization of the engineering for quality requirements" such as safety. Studies by Martin and Gorschek [5] [6] corroborate on this observation. Their studies attribute this problem to 1) the lack of awareness and adoption of software safety methods and processes among organizations implementing safety-critical systems, 2) the lack of software competency within system safety organizations, and 3) the lack of software safety teaching in universities' computer science and software engineering curriculums. The failure to pay sufficient attention to software safety was attributed to several high-profile system mishaps or near mishaps, including the crash of the Korean Air Flight 801 airliner in 1997 that killed 228 passengers [7], the loss the ARIANE 5 rocket in 1996 that cost over $7B to develop and $370M to launch [8], the failure of the Patriot missile defense system against a SCUD missile attack that killed 28 and injuring

96 U.S soldiers in Dhahran in 1991 [9], and the failure of the Soviet's ballistic missile early warning system in 1983 which nearly triggered a nuclear war [10].

To improve on software safety, Hatcliff et al. recommend five guiding safe software development principles, all of which are related to software safety requirements engineering [4]:

- Software safety requirements shall be defined to address the software contribution to system hazards.
- The intent of the software safety requirement shall be maintained throughout requirements decomposition.
- Software safety requirements shall be satisfied.
- Hazardous behavior of the software shall be identified and mitigated.
- The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk.

Heeding the call from these studies for more publications on software safety practices, we present our approach to software safety requirements engineering (SSRE). Our SSRE approach integrates three distinct hazard analysis methods. These methods are the System-Theoretic Process Analysis (STPA) [11], the Functional Hazard Analysis (FHA) of the MIL-STD-882E System Safety Standard [12], and the Software Failure Modes and Effects Analysis (R-SFMEA) [13] methods. These methods were selected because 1) each represents a unique approach to safety requirements engineering, 2) they have been used in software safety programs, and 3) they are complementary in their approaches. Our SSRE approach was developed for used in a V-Model [14] process which is a typical software development lifecycle process used in military weapon systems development.

In the next section, we will introduce the topic of software safety requirement engineering in safety-critical system development. We then will review the three hazard analysis methods all software engineers should be familiar with. Finally, we will show how these individual methods can be integrated together to support a more comprehensive SSRE process.

2 SOFTWARE SAFETY REQUIREMENTS ENGINEERING

Software safety is often treated synonymously with software reliability by mistake [15]. To understand software safety requires differentiating it from software reliability. Formally, software reliability is defined as "the probability that software will not cause the failure of a system for a specified time under specified conditions" [16]. Software reliability factors such as correctness, consistency, and availability are predicted through simulation and modeling and assured through intensive applications of quality assurance techniques such as requirements review, requirements tracing, design review, software testing, code review, code coverage analysis, static code analysis, and software independent verification and validation. Software reliability is further enhanced by adding redundancy to lengthen time-to-failure and watch-dog timers to shorten the time-between-failure [16]. Many quality assurance techniques for achieving software reliability mentioned above have been integrated into standardized software engineering curriculums, processes, and practices today. In the V-Model, measures to assure software reliability are applied to every level in the development process from system conception to operation and maintenance. Software reliability, some even argue, is software safety [17] and that all safety-related failures should be considered failures of reliability [18].

Unfortunately, meeting the reliability requirements for software in safety-critical systems is very challenging [12] [19] [20]. Complex software systems are often costly and challenging to test for reliability. Software reliability is affected by the difference in the system configurations and system usages in test and operation, i.e., its operation and test profiles. Unlike hardware reliability, software reliably cannot be predicted using the laws of physics as compensation for testing limitations. Software failure typically has no advanced warning. Software is easy to change but changing software can easily expose it to new failure risks. These challenges reduce our ability to leverage established hardware reliability techniques for implementing software reliability. Improving software reliability through redundancy such as N-version programming is possible but often impractical due to cost and complexity. Assuring software component failure rate less than 10^{-6} is often unachievable if relying only on software testing [12].

Software safety assumes that embedded software will fail no matter how well it is tested. For software to be considered safe then, two conditions must be met: 1) the software must be sufficiently reliable, and 2) when it fails, the system must not fail or must fail safely. Failing safely means that the failure will not lead to a system mishap. Assuring fail-safe requires paying attention to "what must not go wrong" in addition to "what must go right" throughout the entire development process. Determination of software safety has to be done in the context of system safety as the impact of software failure on the system is what determines the safety quality of the software built. Software safety requires an understanding of the potential causes of software failure, how software failures can trigger system hazards leading to consequential mishaps, how the risk contribution to system hazards due to software failures can be controlled, how are the risk-mitigating controls designed and built, how can the system be tested against different hazardous software failure conditions, and whether the overall desired system safety has been achieved.

Software hazard analysis provides a means to realize software safety. This type of analysis focuses on predicting and controlling what could go wrong in software that will contribute to system hazards. Adding these hazard analysis methods to software requirements engineering helps produce the additional requirements needed to assure the software systems are built for safe operation. Software safety requirements engineering focuses on assuring system functions contributing to system harzards are identified and that safety design constraints are added into the software requirement specifications to control these hazardous functions. Safety design constraints derived from software hazard analysis manifest as new safety constraints or safety requirements [21]. Safety constraints are design decisions expressed as requirements to force a particular safety design, e.g., "The system shall support a triple-redundancy architecture," or implementation, e.g., "Coding shall comply to the MISTRA safety coding standard." Safety requirements are statements imposing a safety restriction on the system, e.g., "The system shall not . . .," a limit on a system function, e.g., "The system

shall perform <a function> only when in this <a state>," or forcing a safety measures, e.g., "The system shall report a condition . . .". Software safety requirements engineering covers engineering of both safety constraints and safety requirements.

Hazard analysis methods such as STPA, FHA, and SFMEA are designed to address different aspects of safety requirement engineering. Each includes features that apply the five principles above to software safety. Our SSRE approach combines the strengths of these methods into a single software safety method for software safety requirements engineering. The following sections will describe these methods and their features most relevant to SSRE. Subsequently, we will describe how we combine these methods into a single comprehensive analysis method.

3 SYSTEM-THEORETIC PROCESS ANALYSIS (STPA)

The System-theoretic Process Analysis (STPA) is a system hazard analysis method that is derived from the process control system model [6]. In this model, a system comprises two distinct subsystems: An operation subsystem that implements the system functions and a control subsystem whose only responsibility is to ensure the quality of the operation subsystem. The control subsystem interfaces the operation subsystem through a set of well-defined monitoring and controlling access points. From the STPA perspective, system safety is achieved by having a highly reliable safety control subsystem that continuously controls the operation subsystem, ensuring that it continues to operate safely. Applying to software, the control subsystem will be implemented in software, controlling the operation subsystem that can be implemented in software or hardware. The STPA approach is similar to the functional safety framework advanced by the International Electrotechnical Commission IEC 61598 [22].

Central to the STPA is a set of system safety constraints representing the performance limits of a system [10]. System constraints are derived from a list of known system hazards the system needs to be protected from. When one or more system constraints are violated, the operation subsystem becomes unsafe. It is the responsibility of the safety control subsystem to take remediation actions to bring the operation subsystem back into a safe state or to assure that the system will fail safely. The control subsystem continuously monitors and detects system constraint violations in the operation subsystem. Safety failure occurs when the safety control subsystem fails to maintain the safe functioning of the operation subsystem resulting in a system mishap.

Architecturally, a safety control subsystem comprises a set of redundant sensors, a safety controller, an operation subsystem performance model, a set of control actuators. The sensors continuously track and provide accurate information about different aspects of the operation subsystem to the safety controller. Using this information, the safety controller detects the system constraint violations and initiates commands to the control actuators to drive the operation subsystem back into a safe operating state. The safety controller maintains an internal performance model of the operating system that reflects its expected condition. Constraint violation happens when the actual condition of the operation subsystem deviates from the expected condition of the model. The sensors, actuators, controller, and the internal model together make up the safety control subsystem. For complex systems, system safety may be realized by a hierarchy of nested safety control subsystems.

Application of STPA starts from the top-down, with identification of the system hazards of interest. From these system hazards, a set of system constraints are identified. A safety control subsystem is designed to monitor for system constraint violations and control the system performance. To ensure the reliability of the control actions, the method focuses on identifying vulnerabilities in the safety control subsystem and in its interfaces with the operation subsystem that prevents the proper application of control actions when needed. These vulnerabilities include possible weaknesses in the sensors, actuators, model, and controller (Figure 1 [33]). For each identified vulnerability, one or more risk-mitigating measures are devised to reduce the risk contribution of the software implementing the control subsystem to system safety. The safety control subsystem and the added mitigating measures are translated to appropriate system and software safety constraints and safety requirements and included in the requirements specifications. Figure 2 summarizes the STPA process and how it interacts with the steps of the system and software requirements engineering processes. A more detailed description of the STPA method can be found in the STPA Handbook [10] and related publications [23].

In Figure 2, after customer needs are translated to system requirements, they are used by safety engineers in conjunction with historical and domain data to identify potential system hazards. From these hazards, safety engineers identify the system constraints to be monitored and controlled. Next safety engineers work with system architects to incorporate a process control model into the system architecture. Software subsystems implementing the control subsystem and their control actions are then identified. Safety engineers perform hazard risk analysis on these control actions and identify mitigating measures to assure their reliability and the safety of the system if they fail. These mitigating measures could be realized in software, hardware, or human actions. The safety engineers work with the requirements engineers to translate these mitigating measures into appropriate safety constraints and safety requirements in the requirements specifications.

4 FUNCTIONAL HAZARD ANALYSIS (FHA)

Functional Hazard Analysis (FHA) is a type of hazard analysis designed for working with the system functions described in System Requirements Specification. Central to an FHA is a set of hazardous system functions that must be controlled to minimize system hazard risk. A hazardous system function is a required system function whose failure can lead to a system mishap. Such function is considered safe after the risk of the system mishap is removed or reduced to an acceptable level. Software FHA (SFHA) is the application of the FHA approach at the software subsystem level to minimize the software contribution to the system hazards. Central to the SFHA is a set of hazardous system functions to be realized in software. Our discussion of SFHA focuses on the application of the FHA method to software hazard analysis. FHA is the hazard analysis method used in the MIL-STD-882E standard [12]. This standard is used in the federal and defense sectors in the United States. This

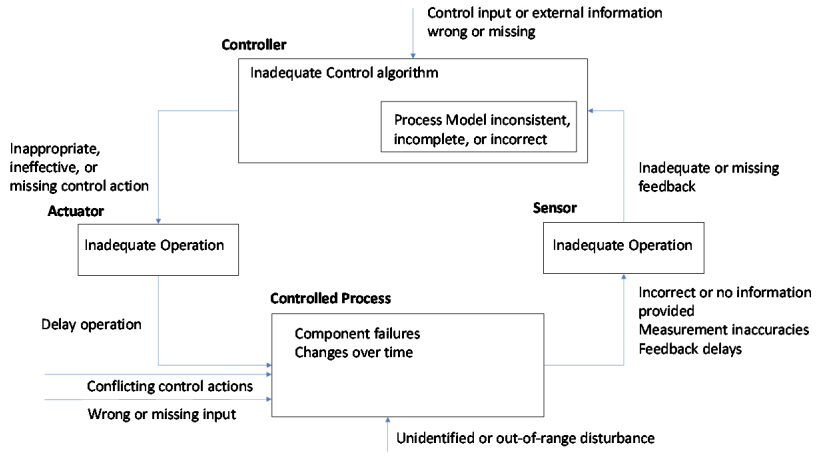

Figure 1: Control Loop Disruptions Leading to Hazardous/Vulnerability States

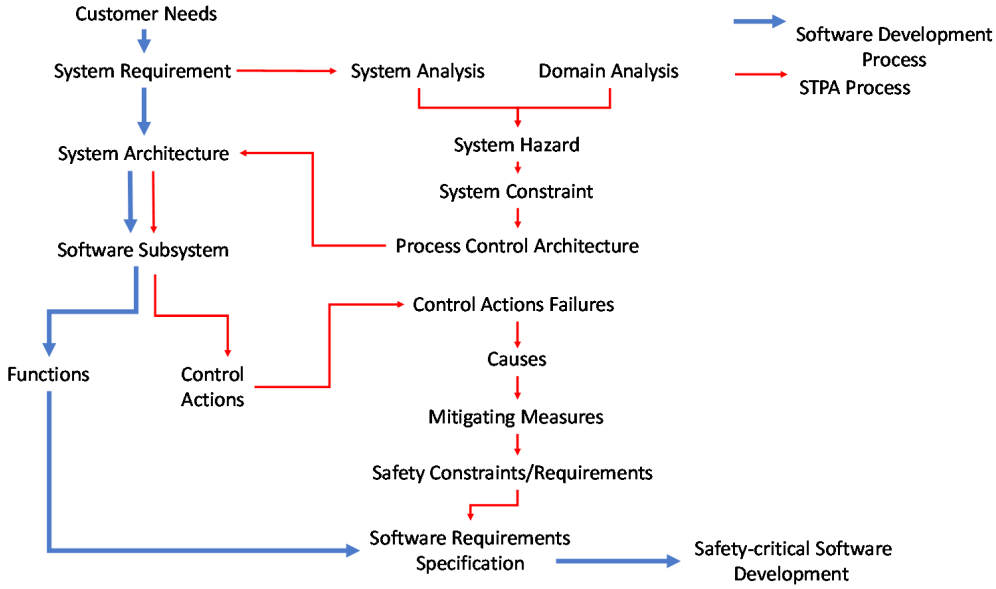

Figure 2: System-Theoretical Process Analysis (STPA)

paper reviews the use of SFHA as described by this standard and its supporting handbook and implementation guide [24] [25].

Whereas the STPA method is top-down, the SFHA method works from bottom-up, starting with individual system functions. Potential system hazards are derived from analysis of the potential failure of each system function. Through bottom-up analysis, the SFHA method allows for the discovery of new system hazards and hazard causal factors. Understanding how individual system functions can potentially cause system hazards allows for different mitigating actions, including [12]:

1. Removal of the hazardous software functions that provide low value from the system specification.

- Replacing the software functions with more reliable hardware solutions.
- Redesigning the functions to reduce risk.
- Adding devices to monitor, detect and control the function when they fail.
- Adding operating procedures for detecting and handling these functions when they fail.

Application of SFHA starts with identifying the system functions whose failure can result in harmful consequences. These functions are derived from analyzing the system requirements. For each hazardous system function, a software safety criticality is determined. Software safety criticality measures the degree of control software

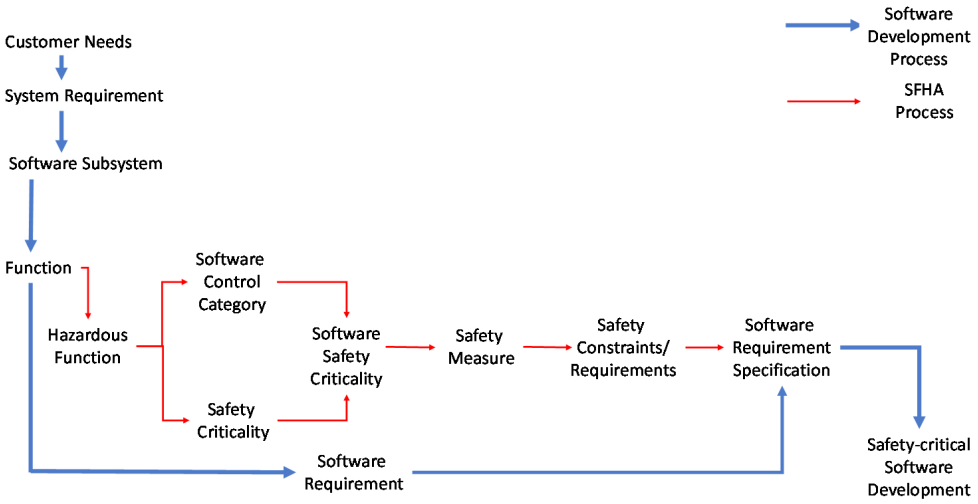

Figure 3: Software Functional Hazard Analysis (SFHA)

has on the hazardous function. Functions with high software safety criticality have higher software risk contributions thus requiring more safety analysis and assurance. The SFHA method does not rely on probability estimation to determine software hazard risks. Instead, it relies on a set of qualitative risk factors: the mishap severity, the software safety criticality, LOR tasks compliance, the allocation of system functions to software subsystems, the design of the safety measures, and the software test results [18]. The safety measures derived from the SFHA process translated to new safety constraints and safety requirements to add to the system and software requirements specifications. Figure 3 summarizes how the SFHA can be integrated into a software requirement engineering process, starting with analyzing individual system functions allocated to software.

Additional discussions of the SFHA method and a case study can also be found in [26] [27].

5 SOFTWARE FAILURE MODES AND EFFECTS ANALYSIS (SFMEA)

The Software Failure Modes and Effects Analysis (SFMEA) is a hazard analysis method that works with software requirements [13]. This method adapts the Failure Modes and Effects Analysis (FMEA) method, a system hazard analysis developed in the early 1980s [28], for analyzing hazards of software components in safety-critical systems. The SFMEA method approaches software hazard analysis from eight viewpoints: functional, interface, code, maintenance, usability, vulnerability, serviceability, and production. For this paper, only the functional and interface viewpoints are relevant as they are the requirements-level viewpoints. Requirements-focus SFMEA (R-SFMEA) method focuses on the software risk contribution to system hazards due to faulty software functional and interface requirements. The outcome of the R-SFMEA is a set of recommended hazard risk-reducing safety requirements to be added into the software requirement specification of a software component. Software hazard analysis methods that work on software requirements are also known generally as Software Requirements Hazard Analysis (SRHA) [29] or Software Safety Requirements Analysis (SSRA) method [1].

Central to the R-SFMEA are the software requirements. Software requirements are considered hazardous if the functions or interfaces they describe can lead to a system mishap once realized. Defective requirements are not necessarily hazardous, although some are. Defective requirements include conflicting, ambiguous, unverifiable, incorrect, implicit, and incomplete requirements resulting from translation errors of the system functions' intention or the operating conditions' constraints [30]. Eliminating or reducing the safety risk of defective requirements often involves removing the defects from the requirements before analyzing for hazards.

Non-defective software requirements could also be hazardous if failure in their implementation could lead to a system mishap. Eliminating or reducing the safety of complete requirements often involves removing the hazardous requirements from or adding new risk-mitigating safety requirements into the software requirements specifications. The R-SFMEA method seeks to improve the safety of individual requirements in the software requirements specification.

The R-SFMEA method begins with a software requirements specification and associated software interfaces specifications of a software component or subsystem, as seen in Figure 4. Using this method, all software requirements are reviewed to identify the hazardous requirements. Next, these hazardous requirements are categorized based on their risk contribution to system mishaps. Like the SFHA method, the R-SFMEA method uses non-probabilistic likelihood factors such as software complexity, historical information on system failure, detectability of failure, and available mitigating measures to determine individual software requirements' risk contribution. Recommended improvements include correcting the

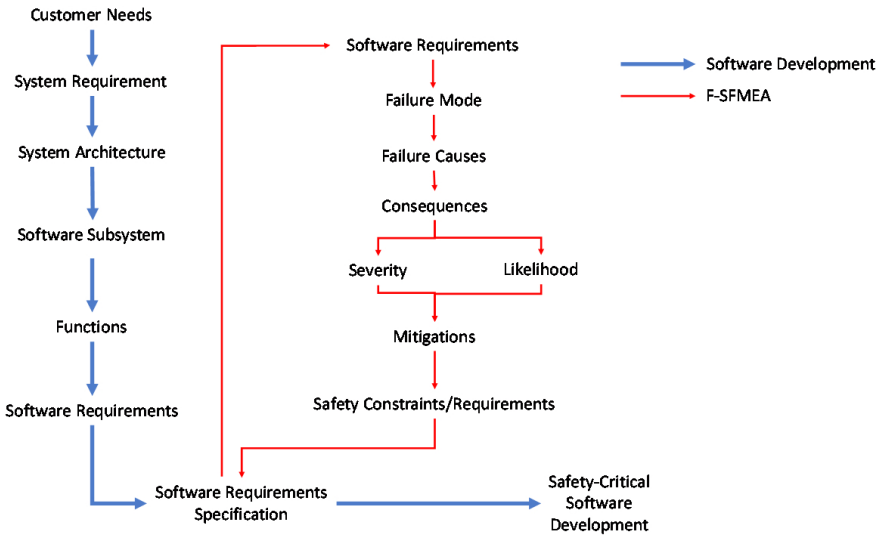

Figure 4: Functional Software Failure Modes and Effects Analysis (F-SFMEA)

original software requirements or adding new software safety constraints and safety requirements to the specifications. More detailed information on the R-SFMEA is available in [13].

6 INTEGRATING THE STPA, SFHA, AND R-SFMEA METHODS

STPA, SFHA, and R-SFMEA methods can be employed together to achieve greater functional safety in software (see Figure 5). STPA assures that all system hazards of interest are accounted for in the system constraints that subsequently drive the development of system control actions. This is a strength of a top-down hazard analysis approach. Another strength of the method is that it centers around the well-known process control system model. This allows the method to leverage existing process control system knowledge and technologies to develop reliable safety monitoring and control solutions. Using the STPA method at the system design level allows for greater reuse of software safety architecture, design, and implementation. The software safety requirements generated from the STPA process help establish the system's baseline safety monitoring and controlling architecture and functionality.

A key weakness of the STPA method is its inability to account for novel system hazards that may arise from unique functions of a particular system. In theory, every function within a system can introduce new system hazards or contribute to existing system hazards in new ways. New system hazards and their causal factors associated with the system functions may be overlooked if the individual functions are not analyzed for hazards. In addition, failure to identify new causal factors of existing system hazards prevents establishment of new safety requirements for monitoring these causes. In this situation, failures occur in the unmonitored part of the operation subsystem that should have been monitored. Third, separating the control and operation subsystems through could introduce unacceptable communication latency between these subsystems during the period of time when the system is under stress, i.e., becoming unsafe.

The SFHA method can be used in conjunction with the STPA method to reduce the risk of overlooking new system hazards, hazard causal factors, or communication latency between the occurrence and control of a safety incident. As a bottom-up method, SFHA reviews the potential failure of individual system functions for potential hazards for failure causal factors. New system hazards uncovered could lead to the creation of new system constraints. New causal factors could lead to improving existing safety monitor and control functions. Enhancing the mission functions with safety features directly can help speed up the system response to a software failure. Furthermore, the SFHA method can also be used to analyze the risk contribution of the software safety measures introduced by the STPA method. The software safety requirements generated from the SFHA process improves on the initial baseline safety requirements from the STPA process.

While both STPA and SFHA methods focus on identifying risk-mitigating opportunities for incorporating into the software requirements, the R-SFMEA method focuses on eliminating the safety risk due to defective software requirements, including defects created in the process of generating safety requirements from the safety measures identified using the STPA and SFHA methods. The outcome of the R-SFMEA method is a complete set of software safety requirements for developing the safety-critical system. These requirements are incorporated into the various software requirements specifications.

Figure 6 summarizes how we integrate these three methods into a single SSRE process. Within the software requirements engineering process, our approach assures every process step is evaluated for safety implications. While incorporating all three hazard analysis methods into a single SSRE process does increase the complexity of the process, failure to detect and control system hazards and hazard

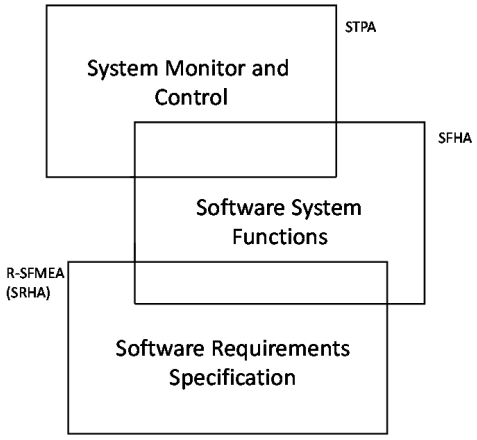

Figure 5: System, Subsystem, and Requirements Hazard Analyses

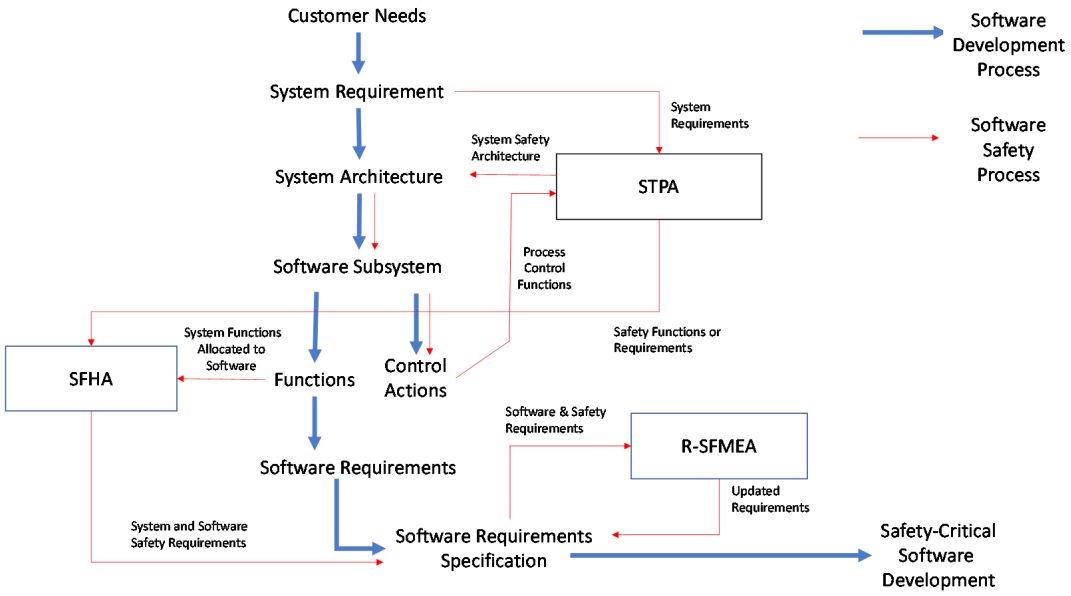

Figure 6: Integration of the STPA, SFHA, and R-SFMEA Methods

risks in safety-critical systems can result in highly consequential system failures. The decision to use one or more complementary hazard analysis methods in a single SSRE should be commensurate with the degree of risk tolerable by the system stakeholders, including the organization acquiring the system, the organization building it, and the customers who will eventually use it.

7 CONCLUSION AND NEXT STEPS

Integration of software hazard analysis methods into software engineering processes is critical in safety-critical systems development. Traditional software engineering processes rely heavily on reliability techniques to assure software systems are developed to perform all required functionality. For software safety, reliance only on assuring "what must go right" is not sufficient. Software hazard analysis methods focus on the "what must not go wrong" aspect of the system. Software safety requires incorporating these hazard analysis methods into the software processes, especially the requirements engineering process. Our paper reviews three distinct hazard analysis methods used in software requirements engineering to develop software safety requirements. Each method has been used in practice in various safety programs. We strongly recommend combining these methods into a single SSRE method, leveraging

their complementary strengths to improve the development organizations' ability to identify and control software hazards at the requirements level.

We are working on adapting our SSRE approach to Agile development. While the V-model remains a popular software process in safety-critical systems development, some organizations adopt the agile software approach [31]. Agile adoption promises the delivery of high-value software solutions to end-users quickly and incrementally. New requirements are welcome in agile development as the system delivered is expected to continue to evolve functionally. These system or software requirements are expressed in increments of light-weight and evolving user stories [32] [33]. Agile software development then requires a minimalist approach to safety requirements engineering, thus a challenge to software safety.

REFERENCES

[1] 737 MAX. 2021. Boeing 737 MAX groundings. Retrieved from https://en.wikipedia.org/wiki/Boeing_737_MAX groundings.

[2] Martin S. Chizek. 2020. 737 Max System Safety Lessons Learned. A tutorial presented at the 38th International System Safety Conference 2020.

[3] IEEE 1228. 1994. IEEE Standard for Software Safety Plans, IEEE Computer Society.

[4] John Hatcliff, Alan Wassyng, Tim Kelly, Cyrille Comar, and Jones Paul. 2014. Certifiably Safe Software-Dependent Systems: Challenges and Directions. In the Proceedings of the Federal Office Systems Expo 2014 (FOSE' 14).

[5] Luiz E. G. Martins and Tony Gorschek. 2017. Requirements engineering for safety-critical systems: overview and challenges. In IEEE Software.

[6] Luiz E. G. Martins and Tony Gorschek. 2020. Requirements engineering for safety-critical systems: An interview study with industry practitioners. IEEE Transactions on Software Engineering, 2020.

[7] Flight 801. 2021. Korean Air Flight 801. Retrieved from https://en.wikipedia.org/wiki/Korean_Air_Flight_801.

[8] Ariane 5. 2021. The Explosion of the Ariane 5. Retrieved from https://www-users.cse.umn.edu/~arnold/disasters/ ariane.html

[9] Patriot Missile. 2021. The Patriot Missile Failure. Retrieved from https://www-users.cse.umn.edu/~arnold/ disasters/ patriot.html

[10] Soviet 1983. 2021. 1983 Soviet nuclear false alarm incident. Retrieved from https://en.wikipedia.org/wiki/1983_Soviet_nuclear_false_alarm_incident #:~:text=On%2026% 20September%2019 83% 2C% 20 the, bases% 20in% 20the%20United% 20States. &text=Investigation% 20of%20the% 20satellite%20warning, the%20system%20had% 20indeed%20malfunctioned.

[11] Nancy G. Leveson and John P. Thomas. 2018. STPA Handbook. Retrieved from https://psas.scripts.mit.edu/home/get_file.php?name= STPA_handbook.pdf

[12] MIL-STD-882E. 2012. System Safety. Department of Defense Standard Practice.

[13] Ann M. Neufelder. 2017. Effective Application of Software Failure Modes Effects Analysis. 2nd Ed.

[14] V-Model. 2021. V-Model. Retrieved from https://en.wikipedia.org/wiki/V-Model. html.

[15] Nancy G. Leveson. 2017. The Therac-25: 30 years later. In Computer, vol. 50, no. 11, pp. 8-11, 2017.

[16] IEEE 1633. 2008. Recommended Practice on Software Reliability, IEEE Reliability Society.

[17] Ron Moore. 2021. A reliable plant is a safe plant is a cost-effective plant. Retrieved from https://www.lce.com/A-Reliable-Plant-is-a-Safe-Plant-1266.html.

[18] Editorials. 1981. Reliablity vs. Safety. Transactions on Reliability, Vol R-30, No. 2, June 1981.

[19] Bev Littlewood & Lorenzo Strigini (1993). Validation of Ultra-High Dependability for Software-based Systems. Communication of the ACM. Vol. 36, No. 11, 1993.

[20] Ricky W. Butler & George B. Finelli (1993). The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software. IEEE Transactions on Software Engineering. Vol. 19, No. 1, 1993.

[21] Donald Firesmith. 2004. Engineering Safety Requirements, Safety Constraints, and Safety-Critical Requirements. Journal of Object Technology. Vol 3. No 3. March-April 2004.

[22] IEC 61508-1. 1997. Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1 General requirements. International Electrotechnical Commission.

[23] Nancy G. Leveson. 2004. A System-Theoretic Approach to Safety in Software-Intensive Systems.

[24] JSSSEH. 2010. Joint Software Systems Safety Engineering Handbook. Department of Defense, 2010.

[25] JS-SSSA. 2018. Software System Safety – Implementation Process and Tasks Supporting MIL-STD-882E. Joint Services – Software Safety Authorities, 2012.

[26] Vu N. Tran, Long N. Tran, and Viet N. Tran. 2021. Functional Hazard Analysis for Engineering Safe Software Requirements. In the Proceedings of the 4th International Conference on Information and Computer Technologies (ICICT), H.I., USA, 2021 pp. 142-148. doi:10.1109/ICICT52872.2021.00031

[27] Vu N. Tran, Long V. Tran, Viet N. Tran, and Dao N. Vu. 2022. Functional Hazard Analysis of an Adaptive Cruise Control System – A Software Safety Requirements Engineering Case Study. In the 68th Annual Reliability and Maintainability Symposium (RAMS). Paper under review.

[28] STD-882B. 1977. MIL-STD-882B System Safety Program Requirements. Department of Defense, 1977.

[29] FMEA. 1980. MIL-STD-1629A Procedures for Performing a Failure Mode, Effects and Criticality Analyis.

[30] IEEE 830. 1998. Recommended Practice for Software Requirements Specifications.

[31] Lise T. Heeager & Peter A. Nielson (2018). A Conceptual Model of Agile Software Development in a Safety-Critical Context; A Systematic Literature Review. Information and Software Technology. doi: 10.1016/j.infsof.2018.06.004.

[32] Hajou A. , Batenburg, R. S., & Jansen, S. (2015). Method æ, the Agile Software Development Method Tailored for the Pharmaceutical Industry. Lecture Notes on Software Engineering 3 (4):251.

[33] Jane Cleland-Huang & Michael Vierhauser (2018). Discovering, Analyzing, and Managing Safety Stories in Agile Projects. In 26th IEEE International Requirements Engineering Conference. doi: 10.1109/RE.2018.00034.