Study on Software Safety in Safety Critical Computer Systems

[1]K.Jayasri, [2]P.Seetharamaiah
[1]Dept. of CSE, GMR Institute of Technology, Rajam, AP, INDIA
[2]Dept. of CS & SE, AU College of Engineering, Visakhapatnam, AP, INDIA

Abstract
Real time computing systems in which the consequences of failure can be very high are termed as safety critical computer systems. Many such systems exist in application areas such as aerospace, defense, transportation and medical devices. Software safety is a composite of many factors. Software for safety-critical systems has to deal with the hazards identified by safety analysis in order to make the system safe, risk-free and fail-safe. To prevent system hazard leaded by software failure, steps insuring software safety must be done. Activities of software safety including safety analyses, safety design and safety evaluation should be deployed surrounding software safety requirements. During software development lifecycle, hazard identification, hazard analysis, testing, and hazard tracking are processed to ensure system safety. This paper examines the driving forces behind the software safety by analyzing various safety related concepts and models like CMM +Safe, for safety critical computer systems.

Keywords
Software Safety, Hazard Analysis, PHA, Lifecycle

I. Introduction

Computers are increasingly being introduced into safety and reliability critical systems. From medical equipment to air traffic control systems, from nuclear power plant controls to the anti-lock braking system in new cars, from fly-by-wire systems on board aircraft to automatic interlocking systems for trains, software has become an integral part of everyday systems upon which millions of lives depend [1]. Automatic teller machines, internet-based systems, mobile phones, communication satellites, administration and database systems, etc. are other critical systems that might not kill people in case of failure but on which our day to day life is based.

Software is a very flexible medium capable of expressing a wide range of behaviors. Computers are fast and relatively cheap, and there are some actions in system that only a computer is capable of performing. These characteristics allow engineers to do things like: make jet engines more fuel efficient, help chemical plants are more productive, and take some tedious tasks away from human operators. There are also certain amounts of myth that surrounds software and adds to its popularity. Some of the myths are: software is cheap because you only make it once, computers are more reliable than mechanical devices, and software is easy to change. Most of these apparent features stem from software's flexibility, the speed of modern microprocessors, and the notion that software never wears out. While there is truth to each myth, they all come with a downside. The downside comes from the amount of complexity we now put into software. To make the jet engine fuel efficient, the controlling software becomes very complex. So despite the absence of manufacturing or end of life wear out, software is becoming the most expensive part of the system to create, and because so much of a system depends on software for control, it can no longer be looked at as a side issue. Now the software is the system.

Software presents new problems for system engineers that do not appear in mechanical systems. Software is inherently imperfect. While the same can be said of a mechanical system, software has a much larger testing space because it has non-linear properties. Verifying a complex piece of software through testing is effectively impossible it may, at first glance, appear that the best way to write safe software is to try and make it as reliable as possible. Unfortunately, even perfectly correct software could trigger an accident because the actions a software engineer thought were appropriate really put the system into an unsafe state. Software engineers must understand the safety issues involved with the software they write. System engineers are in a situation where they have highly complex, dangerous systems being controlled by a technology that we cannot easily verify. Until software engineers have more control over software, it will be difficult to make it safe, but it is unlikely that people will stop using software to build systems. It is much too attractive an option, despite the potential risks [10]. Complex, dangerous systems are a fact of our modern day life; this paper will closely examine the risks behind using software in potentially hazardous systems, and some of the techniques that can be used to reduce that risk.

The first important issue is why software has become so prevalent in safety critical systems. Safety is an emergent system property, and one component cannot make a system safe. First it is important to consider safety from the very beginning of system design and a safety team responsible for system safety issues, should be created. Second, extensive safety analysis should be done to try and come up with as many safety issues as possible.

The safety of safety-critical systems is of great public concern which is reflected in the fact that many such systems must adhere to government regulations or industry standards and/or be certified by licensing bodies [2]. Such regulations, standards, and bodies are concerned exclusively with safety [6].Computers-based safety-critical systems have been involved in several accidents. Some well-known software-related accidents in key industrial sectors are described below, in brief.

A. Korean Air 747 Disaster

In 1997, 225 of the 254 people on board were killed in the crash of the Korean Air 747, Flight 801, in Guam. National Transportation Safety Board (NTSB) investigators said that a software error might have been a contributing factor in the crash of the aircraft.

B. Therac-25 Accidents

Some of the most widely cited software-related accidents in safety-critical systems involved a computerized radiation therapy machine called the Therac-25. Its faulty operation led between June 1985 and January 1987, six known accidents involved massive overdoses with resultant deaths and serious injuries.

One of the safety features in the original design was that all of the settings for the device had to be entered through both a terminal

and on a control panel. Due to a bug in the software, some of the settings were, occasionally, not properly recorded. The bug was a race condition created because proper resource locking of the data was not exercised [3].Software engineering of a safety-critical system requires a clear understanding of the software's role in, and interactions with, the system [4-5]. The development of safety-critical systems demands a different, more rigorous approach than most other computer applications. As safety-critical systems are often real-time control systems they require the utmost care in their specification, design, implementation, operation and maintenance, as they could lead to injuries or loss of lives and in-turn result in financial loss [7-8].

From the perspective of most of the system safety community, digital control of safety-critical functions introduced a new and unwanted level of uncertainty to a historically sound hazard analysis methodology for hardware. Many within system safety were unsure of how to integrate software into the system safety process, techniques, and methods that were currently being used. System safety managers and engineers, educated in the 1950s, 60s, and 70s, had relatively no computer-, or software-related education or experience . In the late 1970s and early 1980s, bold individuals within the safety, software, and research communities took their first steps in identifying and addressing the safety risks associated with software. Although these individuals may not have been in total lock step and agreement, they did, in fact, lay the necessary foundation for where we are today. It was during this period that MIL-STD-882B was developed and published [9]. This was the first military standard to require that the developing contractor perform SSS engineering and management activities and tasks. However, due to the distinct lack of cooperation or communication between the system safety and software engineering disciplines in defining a workable process for identifying and controlling software-related hazards in developing systems, the majority of system safety professionals waited for academia, or the software engineering community to develop a "silver bullet" analysis methodology or tool.

In the literature, the researches on software safety include software safety analysis, safety critical intensive systems, software safety metrics, and validation metrics framework for software safety analysis [14, 19]. Because all of the researches just discuss a certain aspects of software safety metrics and issues [15]. It is difficult to understand the relationship among the researches. Enhancing the performance of software safety is a critical and challenging task. Large number of studies have analyzed and addressed various issues related to software safety. This section addresses the issues reported by some of the previous researchers.

II. System Safety Program

Software safety shall be an integral part of the overall system safety and software development efforts. It is the objective of the software safety effort to ensure that safety is considered throughout the software life cycle. Therefore, software safety activities take place in every phase of the system and software development life cycle beginning as early as the concept phase and on through to operations and maintenance. Software safety tasks by life cycle phase:

1. Software safety planning
2. Software requirements specification development
3. Software architectural design
4. Software detailed design
5. Software implementation
6. Software integration and acceptance testing
7. Software operations and maintenance.

A system safety program is a prerequisite to performing analysis or development of safety critical software [12]. This program maps out the types of analysis and the schedule for performing a series of systems and subsystem level analyses throughout the development cycle. It will also address how safety analyses results and the sign-off and approval process should be handled. The first system safety program activity is a Preliminary Hazard Analysis (PHA), discussed in Section 2.1. The results of this step, a list of hazard causes and potential hazard controls, are flowed to the safety requirements development activity. Before software can be analyzed or developed for use in a hazardous system or environment, a system PHA must be performed. Once initial system PHA results are available, safety requirements flow down and subsystem and component hazard analyses can begin [16]. System safety program analyses follow the lifecycle of the system and software development efforts, commencing with the Preliminary Hazard Analysis (PHA).

A. Preliminary Hazard Analysis (PHA)

The PHA is the first source of "specific" software safety requirements, (i.e., unique to the particular system architecture). It is a prerequisite to performing any software safety analysis. The Preliminary Hazard Analysis "documents which generic hazards (Generic Hazards Checklist for a sample checklist of generic hazards) are associated with the design and operational concept. This provides the initial framework for a master listing (or hazard catalogue) of hazards and associated risks that require tracking and resolution during the course of the program design and development. The PHA may be used to identify safety-critical systems that will require the application of Failure Modes and Effects Analysis (FMEA) and further hazard analysis during the design phases. The program shall require and document a PHA to obtain an initial listing of risk factors for a system concept. The PHA effort shall be started during the concept exploration phase or earliest life cycle phases of the program. A Preliminary hazard analysis of the entire system is performed from the top down to identify hazards and hazardous conditions. Its goal is to identify all credible hazards upfront. Initially the analysis is hardware driven, considering the hardware actuators, end effectors and energy sources, and the hazards that can arise. For each identified hazard, the PHA identifies the hazard causes and candidate control methods. These hazards and hazard causes are mapped to system functions and their failure modes. To assure full coverage of all aspects of functional safety, it can be helpful to categorize system functions as two types:

1. "Must work" functions (MWF)
2. "Must not work" functions (MNWF)

The system specification often initially defines the criticality (e.g., safety critical) of some system functions, but may be incomplete. This criticality is usually expressed only in terms of the Must-Work nature of the system function, and often omits the Must-Not-Work functional criticality. The PHA defines all the hazardous MNWFs (Must-Not-Work-Functions) as well as the MWFs (Must Work Functions).

B. Software Subsystem Hazard Analysis

After safety critical software is identified in the first cycle of the PHA, software hazard analysis can begin. The first cycle of System

Hazard Analysis and software hazard analysis are top-down only. Bottom-up analyses take place after a sufficient level of design detail is available. To prevent system hazard leaded by software failure, steps insuring software safety must be done. Activities of software safety including safety analyses, safety design and safety evaluation should be deployed surrounding software safety requirements. Software safety development runs through every stage of software developing Lifecycle. The first step in classifying safety risk requires the establishment of hazard severity within the context of the system and user environments. This is typically done in two steps, first using the severity of damage and then applying the number of times that the damage might occur. Table -1 provides an example of how severity can be qualified.

Table 1: Hazard Severity

DESCRIPTION	CATEGORY	HAZARD EFFECT
Catastrophic	I	Death Or System Loss
Critical	II	Severe Injury, Occupational Illness, Major System Or Environmental Damage
Marginal	III	Minor Injury, Occupational Illness, Minor System Or Environmental Damage
Negligible	IV	Less Than Minor Injury, Illness, System Damage Or Environmental Damage

III. Software Safety Throughout the Development Life Cycle

Software safety development is closely and inseparably related to system safety analysis and design. System safety analysis and design is the base of software safety development, and software safety development is constituent part of system safety development [13]. Software developer should communicate with system engineers, system safety engineers adequately. Understanding system safety requirements, software function and safety responsibility beard by software very well, software safety development then can be processed successfully.

A. Safety Tasks During Software Development

Safety must be an integral part of the system and software from the very start. In system and software concept stage, software developers should create the software safety plan, including planning and tailoring the safety effort. Safety requirements for software is being developed and analyzed in the next stage, software requirements [11]. All functional software safety requirements shall be incorporated into the software design. The software design will ensure that software safety features and requirements can be thoroughly tested. Software safety personnel have the responsibility to conduct an analysis of the software design. It is during software implementation that software controls of safety hazards are actually realized [18]. Safety requirements have been passed down through the designs to the coding level.

Software testing verifies analysis results, investigates program behavior, and confirms that the program compiles with safety requirements. Software testing will include safety testing at the unit level and component level, as well as system and acceptance testing. At the beginning of a device development project, nobody can predict all of the potential hazards and their causes that, ultimately, could lead to the harm of a patient, user or the environment. Part of practicing software safety is the iterative application of risk management processes at each phase or activity of the software-development life cycle. It makes no sense to try to analyze failure modes of yet-to-be designed components in the early concept phases of a development project [17]. However, if nothing is done until the end of the project, retro-implementing risk control measures into a system that is almost complete can be very costly and can lead to unexpected defects. Table 2, though far from being all-inclusive, suggests some activities that should be considered as the development of a device progresses through the life cycle. As more detail about the design and implementation of the device evolves, more sophisticated tools and methodologies are used to predict causes for hazards.

Table 2: Risk Control Over the Development Life Cycle

Concept	Identify users, use environment Preliminary hazard and cause analysis Preliminary itemization of risk control measures(RCMs)
Safety Requirements	Identify software safety requirements for software implemented RCMs Identify non software safety RCMs for possible software failures. Review and revise activities from prior phase.
Safety Design	Design for software RCM requirements At the Architectural design level and module design level review for failure modes that could cause errors. Review and revise activities from prior phase.
Safety Implementation	Implement software RCM designs Review and revise activities from prior phase.
Software Safety Test	Fully test all software implemented RCMs. Test the effectiveness of non-software RCMs for software initiated hazards. Review and revise activities from prior phase.
Safety Maintenance	Clearly document the configuration that has been “safety validated”.

Hazard tracking process is incorporated the total system safety process in order to comprehensively track software safety related hazards [19]. When system is operating, more software safety information/data are necessary for hazard tracking.

IV. CMM +Safe Model

+SAFE is an extension of the continuous representation of CMMI for Development, Version 1.2 (CMMI-DEV, V1.2) and is intended to specifically address safety. Organizations from industry, government, and the Software Engineering Institute (SEI) jointly developed the CMMI Framework, a set of integrated models, the appraisal method, training materials, and supporting products. The purpose of +SAFE is to extend CMMI to provide an explicit, specific framework for functional safety with respect to developing complex safety-critical products [20]. +SAFE is provided in a

form that can be used standalone by experienced CMMI users with minimal support from safety domain experts. +SAFE is an extension of CMMI, so it adopts the same assumptions, model structure, conventions, and terminology as CMMI, and is affected by the general process-area and capability-level interactions inherent in CMMI. The +SAFE extension to CMMI does not solely consist of the addition of new process areas, described in the standard CMMI manner, to existing process area categories. The +SAFE framework is not specific to any safety standard, and standards that define safety principles, methods, techniques, work products, and product assessments may be used to satisfy the goals of the framework as appropriate.

V. Conclusion

Safety is an extremely complex issue that depends on many factors inside and outside a system. Software is a relatively new technology that is becoming increasingly responsible for the safety of a system. Unfortunately, it is hard to make correct software and even harder to make safe software. Correct software is hard because of the complexity engineers thrust into software and its inherent non linearity. Safety is hard because it is an emergent property of the system, and it depends on software. Software safety provides a systematic approach to identifying, analysing, and tracking software mitigation and control of hazards and hazardous functions to ensure safer software operation within a system. It's an aspect of software engineering extended from system safety. During software developing, we should develop safety engineering activities generally to reduce software failure related with system safety, mitigation/control system risk being acceptable, and make arduous efforts to advance system safety. Software Safety is a problem that will continue to plague computerized societies in the near future. The only way to overcome this issue is to realize our inability to understand software and use it will caution until we understand it better. Once we have a better understanding of software and its impact on system safety, we will be better able to use it in safety critical systems.

References

[1] R. Wallace, L. M. Ippolito,"A Framework for the Development and Assurance of High Integrity Software", National Institute of Standards and Technology, NIST Special Publication, December 1994.

[2] J. P. Bowen, M. G. Hinchey,"Formal models and the specification process", In A. B. Tucker, Jr., editor, The Computer Science and Engineering Handbook, pp. 2302–2322. CRC Press, 1997. Section X, Software Engineering

[3] N.G. Leveson, C. S. Turner,"An Investigation of the Therac-25 Accidents", IEEE Computer, 26(7), pp. 18- 41, March 1987.

[4] Robyn R. Lutz,"Software Engineering for Safety: A Roadmap", Proceedings of the Conference on The Future of Software Engineering, June 04-11, 2000, Limerick, Ireland, pp. 213-226.

[5] John C. Knight.,"Safety Critical Systems: Challenges and Directions", Proceedings of the 24th International Conference on Software Engineering (ICSE), Orlando, Florida, 2002

[6] NASA-Guide Book-8719.13 on March 31, 2004.

[7] Pfleeger, S.L., Fenton, N., Page, S.,"Evaluating Software Engineering Standards", Computer, September 1994, pp. 71-79.

[8] Leveson N.G.,"Software Safety: Why, What and How, Computing Surveys", 18, pp. 125-163, 1986.

[9] Fenton, Norman E., Neil, Martin,"A Strategy for Improving Safety Related Software Engineering Standards", IEEE Transactions on Software Engineering, Vol. 24, No. 11, November 1998.

[10] Hansen, Kirsten M., Anders, P. Ravn, Stavridou, Victoria,"From Safety Analysis to Software Requirements", IEEE Transactions on Software Engineering, Vol. 24, No. 7, July 1998.

[11] James Bret Michael, Man-TakShing, Kristian John Cruickshank, Patrick James Redmond,"Hazard Analysis and Validation Metrics Framework for System of Systems Software Safety", Published in IEEE Systems Journal, Vol. 4, No. 2, June 2010.

[12] R. Weaver, J. Fenn, T. Kelly,"A Pragmatic Approach to Reasoning about the Assurance of Safety Arguments", Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03), 2003, Vol. 33, pp. 57.

[13] P.V.Srinivas Acharyulu, P. Seetharamaiah,"A Methodological Framework for software Safety in safety critical computer systems", Journal of computer science, Science Publications, Vol. 8, Issue 9, 2012, pp. 1564-1575.

[14] V. R. Basili, D. M. Weiss,"A Methodology for Collecting Valid Software Engineering Data", IEEE Transactions on Software Engineering, Vol. SE-I0, No. 6, November 1984, pp. 728-738.

[15] V. R. Basili, H. D. Rombach,"The TAME Project: Towards Improvement-Oriented Software Environments", IEEE Transactions on Software Engineering, Vol. SE-14, No. 6, June 1988, pp. 761.

[16] John C. Knight,"Safety Critical Systems: Challenges and Directions", Proceedings of the 24th International Conference on Software Engineering Orlando, Florida, 2002, pp. 547 – 550, 2002.

[17] Software Safety,"NASA Technical Standard", (1997) [Online] Available: http://satc.gsfc.nasa.gov/assure/distasst.pdf

[18] P. V. Bhansali, January, "Software Safety: Current Status and Future Directions", ACM SIGSOFT Software Engineering Notes, Vol. 30, No. 1, pp. 1, 2005.

[19] Kristian J. Cruickshank, James Bret Michael, Man-TakShing,"A Validation Metrics Framework for Safety-Critical Software-Intensive Systems".

[20] CMMI Product Development Team,"CMMI for Development", Version 1.2 (CMU/SEI-2006-TR- 008, ESC-TR-2006-08). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, August 2006.

Ms.K.Jayasri, Working as Assistant Professor, in Department of Computer Science and Engineering, GMR Institute of Technology, Rajam-532127, Andhra Pradesh. She obtained her M.Tech in Information Technology from Andhra University College of Engineering, Visakhapatnam. She is having around ten publications in various journals and conferences. She is having 6 years of teaching experience. At present she is pursuing part-time doctoral degree in Computer Science & Systems Engineering department from Andhra University, Visakhapatnam. Her research interests are Safety Critical Computer Systems, Software Safety, Software Engineering and Operating Systems.

Dr. P. Seetha Ramaiah, Professor, Department of Computer Science and Systems Engineering, College of Engineering, Andhra University, Visakhapatnam- 530003, Andhra Pradesh– INDIA. Dr. Panchumarthy Seetha Ramaiah obtained his PhD in Computer Science from Andhra University in 1990. He is presently working as a professor of Computer Science in the department of Computer Science and Systems Engineering, Andhra University, College of Engineering, Visakhapatnam – INDIA. He is the Principal Investigator for several Defense R&D projects and Department of Science and Technology projects of the Government of India in the areas of Embedded Systems and Robotics. He has published seven journal papers, and presented Fifteen International Conference papers in addition to twenty one papers at National Conferences in India. Recipient of "Academic Excellence Award for 2011" by The Honourable Prime Minister of India.