ISSN 2320 - 2602
Volume 3, No.1, January 2014
International Journal of Advances in Computer Science and Technology
Available Online at http://warse.org/pdfs/2014/ijacst03312014.pdf

How to design and test safety critical software systems

Syed Usman Ahmed[1], **Muhammed Asim Azmi**[2], **Charu Badgujar**[3]
[1]Department of Information Technology, JIET, India, syedusman.ahmed@jietjodhpur.com
[2]Fachhochshule Frankfurt am Main, University of Applied Science, Germany, azmi@stud.fh-frankfurt.de
Department of Computer Science and Engineering, Jodhpur National University

ABSTRACT

Safety is the foremost need that every human being desires irrespective of the impact of the breach of safety. As today software is associated with almost every field whether it be education or aerospace, so system demands more and better safety systems and mechanisms. Any system whose failure can catastrophically impact human lives, environment and equipment can be called as safety critical system. These kinds of risks are handled using safety engineering techniques elaborated in this paper. In this paper we have discussed about safety critical systems, their specifications and standards, Language support and approaches of designing safety critical systems.

Key words: Safety Critical System, Ada, Failure, Malfunction.

1. INTRODUCTION

A system whose failure or malfunctioning can lead to a catastrophic outcomes on human lives, environment and equipment is termed as safety critical system. These types of risk associated with safety system development and testing are managed by safety engineering. In today's world ideally there is no field that uses computer as its component remains untouched by safety hazards be it the field of medicine, nuclear engineering, transport, industrial simulation, telecommunication etc.

As safety critical system generally works with both hardware and software systems, both of these sub-systems need to work in coordination and securely in order to ensure a safety of the system as a whole. This paper precisely focuses on the software aspect of such system.

As stated by Lawen [1] there are three basic approaches to achieving reliability of safety critical system: Testing - a lot of it, software fault tolerance and fault avoidance – by formal specification and verification, automatic program synthesis and reusable modules. We will elaborate more on these approaches in details with some practical aspects.

2. LITERATURE SURVEY

2.1 Standards of safety critical system

Today industries have designed various different standards for the development of these safety critical systems. These standards are required as every industry want attain uniformity in the development and maintenance of critical system so developed. Standard can be general or generic in nature like ISO 9000 that are general in nature and IEC 61508 that are generic in nature. We also have some industry specific standards like EN 50128 (Railway Industry) and DO 178B (Civil Aviation). [2].

2.1.1 ISO 9000

The International Standardization Organization (ISO) has released the ISO 9000 series of standards that are related to quality assurance and quality management in general [2]. The sub-standards that fall under ISO 9000 are shown in figure 1 below:

ISO Standard	Domain
ISO 9001	Model for quality assurance in design, development, production, installation, servicing
ISO 9002	Model for quality assurance in production, installation and servicing
ISO 9003	Model for quality assurance in final inspection and test

Figure 1: ISO 9000 sub standards [2]

2.1.2 IEC 61508

The International Electrotechnical Commission (IEC) have developed the generic standard IEC 61508. This standard is generic as it takes care of electrical, electronics and related technologies and is titled as "Functional safety of electrical/electronic/programmable electronic safety-related (E/E/PE) systems". There are seven different parts of IEC 61508 standards: [2]

1. IEC 61508-1 : General Requirements
2. IEC 61508-2: Requirements for electrical / electronic / programmable electronic safety-related systems.
3. IEC 61508-3: Software requirements
4. IEC 61508-4: Definition and abbreviations
5. IEC 61508-5: Examples of methods for the determination of safety integrity levels.
6. IEC 61508-6: Guidelines for the application of IEC 61508-2 and IEC 61508-3.
7. IEC 61508-7: Overview of techniques and measures.

Figure 2 shown below highlights the overall safety life cycle of IEC 61508-1.

2.1.3 BS EN 50128 Railway Industry

As per [3] the EN 50128 standard was developed to identify "methods which needs to be used in order to provide software which meets the demand of safety and integrity".

EN 50128 has five software integrity levels (SILs) and each level is associated with certain degree of risk when software systems are being used. The table 1 shows the SIL levels with associated risk.

Table 1: SIL level and associated risk [2]

SIL Level	Risk
0	Nonsafety related
1	Low
2	Medium
3	High
4	Very High

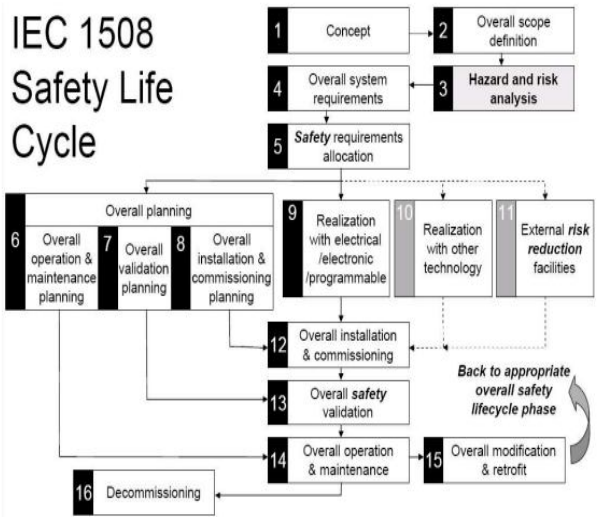

Figure 2: Overall safety life cycle of IEC 61508-1 [2]

Figure 2 shown above describes the overall safety life cycle of IEC 61508-1.

2.1.4 RTCA/DO 178B

RTCA stands for Requirements and Technical Concepts for Aviation and DO 178B is an official guidance document which is widely accepted as an international standard. DO 178B describe the objectives of software life cycle process, process activities, and software levels. These software levels are selected based on the severity of failure conditions on the aircraft and its occupants. Figure 3 shows the software levels of DO 178B.

Software Criticality Level	Definition	Associated Structural Coverage Level objectives, (DO-178B 6.4.4.2 & table A-7)
Level A	Software that could cause or contribute to the failure of the system resulting in a catastrophic failure condition.	Modified Condition/ Decision Coverage, Decision Coverage & Statement Coverage
Level B	Software that could cause or contribute to the failure of the system resulting in a hazardous or severe-major failure condition.	Decision Coverage & Statement Coverage
Level C	Software that could cause or contribute to the failure of the system resulting in a major failure condition.	Statement Coverage
Level D	Software that could cause or contribute to the failure of the system resulting in a minor failure condition.	None required
Level E	Software that could cause or contribute to the failure of the system resulting in no effect on the system.	None Required

Figure 3: DO 178 B levels [2]

2.2 Programming platform

When we design a safety critical system it is essential that the system should be kept as simple as possible. This approach depends on the choice of programming language used to develop the source code of the software.

Most of the modern programming are quite efficient in terms of time and complexity however when it come for developing safety related system these high end languages are mostly avoided. The reasons for avoiding such high level languages as given by [3] are:

(a) Dynamic allocation / de-location of memory.
(b) Use of pointers
(c) Use of unstructured programming constructs like goto.
(d) Multiple entry and exit points in a loop, procedure or functions.
(e) Recursion
(f) Procedural parameters

The essential features that provide reliability and are less likely to leads to errors are:

(a) Strong typing
(b) Runtime constraints checking
(c) Parameter checking

As per [4] the programming language with good features required for developing and testing safety critical system is **Ada**. However as we can know that none of the programming language can be completely secure and reliable for testing safety related systems we take small subsets of good and reliable programming language in order to avoid the risk associated with reasons mentioned above. Ada subset most commonly used for safety critical software systems is called as SPARK [3].

3. DESIGNING OF SAFETY CRITICAL SYSTEMS

While designing a critical system that is required for safety purpose the basic idea is to identify the hazards and constraints as early as possible in system development life cycle. Along with identification of such hazards we need to find out measures that can reduce them to acceptance level.

3.1 Formal method based approach

This is the first approach which is used to formally prove that the system that is being designed does not contain any construction errors by the help of formal methods [3]. The mathematical based methods that are used to specify, design and test software based systems are called formal methods. Here the term specification means well structured statements using mathematical logic and formal proves are logical deductions using rules and inference. The use of formal method is often advocated as a way of increasing confidence in such systems [5]. However to apply formal methods in designing a large system is fairly complicated and time consuming activity that has some big issues like human intervention in official specifications and proves.

Moreover some concepts are harder to prove by formal methods like complier or any utility system software based program. For smaller systems, where formal specifications and proves are easier this approach can be very successful.

The problem associated with large systems can be handled if we can separate the critical functionality from the non critical systems. In this way by creating components that have

different safety integrity levels we can manage large and complex systems as well using formal approach.

R. D. Tennent [6] in his book exemplify a real world example of how to apply formal methods with successful results. It is mention in the book [6] that "The program used to control the NASA space shuttle is a significant example of software whose development has been based on specifications and formal methods […] As of March 1997 the program was some 420,000 lines long. The specifications for all parts of the program filled some 40,000 pages. To implement a change in the navigation software involving less than 2% of the code, some 2,500 pages of specifications were produced before a single line of code was changed. Their approach has been outstandingly successful. The developers found 85% of all coding errors before formal testing began, and 99.9% before delivery of the program to NASA. Only one defect has been discovered in each of the last three versions. In the last 11 versions of the program, the total defect count is only 17, an average of fewer than 0.004 defects per 1,000 lines of code."

3.2 Prevention and recovery based approach

The prevention and recovery based approach assumes that errors do exist in the system and the ultimate aim is to design a prevention and recovery based mechanism that can protect the system from the stated hazards. This approach follows a bottom up structure in which smaller modules or functions are first checked for errors and traversing these smaller parts the complete system is scanned for possible threats. In order to recover data and information sometime we may follow what is called is redundancy approach in which data related to critical parts or modules is replicated.

As mentioned in [4], an example of redundancy focusing only in the software part of a critical-system is the N-version programming technique also known as multi-version programming. In this approach, separate groups develop independent versions of the same system specifications. Then, the outputs are tested to check that they match in the different versions. However, this is not infallible as the errors could have been introduced in the development of the specifications and also because different versions may coincide in errors.

4. TESTING SAFETY CRITICAL SYSTEM

As software testing is gaining momentum many researchers are getting associated with this domain to explore the real potential of testing [8]. Quality of software work product depends on the amount and quality of testing being done software reliability, scalability and performance are some of the factors that are very much valued by the customer [9]. Testing of safety critical system will use all or part of existing legacy software testing techniques, in addition to the existing testing techniques we have to supplement some special techniques in order to minimize the risk and hazard associated with safety of software and environment. The two important factors that distinguish legacy software testing techniques from safety critical testing techniques are:

(a) **Degree of rigor:** The degree of rigor is the amount of formality involved in testing a system. The degree of rigor for safety critical system is more than the normal system.

(b) **Organizational structure:** Independent verification is usually required in those systems by means of a separate team within the overall project structure or a verification team supplied by an external company who may not ever meet the development team, depending on the criticality of the system.

It should be remember and empathized that when testing software at different stages of its development, tests are always performed to verify correct behavior against specifications, not against observed behavior. For this reason, design of test cases for coding, should be done before coding the software system. Otherwise, software developers are tempted to design test cases for the behavior of the system which they already known, rather than for the specified behavior [7].

Some specific techniques to test and verify safety critical systems are:

1. Probabilistic risk assessment (PRA): It is a method that combines failure modes and effect analysis (FMEA) with fault tree analysis (FTA).
2. Failure mode effects and critically analysis (FMECA): It is an extension of former FMEA.
3. Hazard and operability analysis (HAZOP): It is used for hazard and risk analysis.
4. Fishbone diagram: It is a tool for cause and effect analysis.

The steps involved in probabilistic risk assessment (PRA) are as follows:

1. It perform a primary hazard analysis to find out the impact of some predefined hazard on safety of system.
2. Next, the severity of each impact is calculated. The severity levels can be classified as follows:
    a. Catastrophic
    b. Hazardous
    c. Major
    d. Minor
    e. Not safety related
3. The probability of occurrence is then calculated and it can also be classified as:
    a. Probable
    b. Remote
    c. Extremely remote
    d. Extremely improbable
4. Now the assessment of risk is calculated by combining both impact and probability of occurrence in a matrix.

Failure modes and effect analysis (FMEA) is a procedure for analysis of potential failures within a system for classification by severity or determination of the effect of these failures on the system. Failure modes can be defined as any errors or defects in a process, design or item, especially those that affect the customer and can be potential or actual. Effects analysis refers to studying the consequences of these failures. A failure mode, effects and criticality analysis (FMECA) is an extension to this procedure, which includes criticality analysis used to chart the probability of failures against the severity of their consequences.

Fault trees analysis is a graphical technique that provides a systematic description of the combinations of possible occurrences in a system which can result in an undesirable outcome (failure). An undesired effect is taken as the root of a tree of logic. Each situation that could cause that effect is added to the tree as a series of logic expressions. Events are labeled with actual numbers about failure probabilities.

5. CONCLUSION

In this paper, a basic overview of safety-critical software systems has been given. They were first defined and some standards to cope with their development were named. Programming features and languages related to these kinds of systems have also been mentioned. Then, the two main approaches used when designing safety-critical software were explained. Finally, some techniques used to test safety-critical software have been described, both general techniques also used to test typical software systems and special techniques from safety engineering aimed at safety-critical software. The main idea behind the testing techniques mentioned is to reduce risks of implementation errors.

Throughout this paper, it has been showed that safety-critical software systems is a very complex topic. It is this complexity and their relevance in our nowadays society what makes safety-critical systems development to require huge efforts both in time and budget.

The key to the success of any software development, critical or not, are the people in charge of it. They need to have enough training and experience. Both of them become absolutely essential in safety-critical software development.

REFERENCES

[1] J. D. Lawrence, Workshop on Developing Safe Software – Final Report, FESSP, Lawrence Livermore National Laboratory, 1992.

[2] Robert Traussnig, "Safety-Critical Systems:Processes, Standards and Certification" Seminar “Analysis, Design and Implementation of Reliable Software”, 2004.

[3] IPL Information Processing Ltd, An Introduction to Safety Critical Systems, Testing Papers.

[4] Marcos Mainar Lalmolda, "Testing safety-critical software systems", Quality Assurance and Testing report, University of Nottingham, 2009.

[5] Jonathan Bowen,"Safety Critical Systems, Formal Mehods and Standards", Software Engineering Journal, 1992.

[6] R.D. Tennet, Specifying Software: A Hands-On Introduction, 2002.

[7] IPL Information Processing Ltd, *An Introduction to Software Testing*, Testing Papers.

[8] Ahmed, Syed Usman and Azmi, Muhammad Asim, “A Novel Model Based Testing (MBT) approach for Automatic Test Case Generation”, International Journal of Advanced Research in Computer Science, 4(11), pp 81-83, 2013.

[9] Ahmed, Syed Usman, Sahare, Sneha Anil and Ahmed, Alfia, “Automatic test case generation using collaboration UML diagrams”, World Journal of Science and Technology, Vol-2, pp4-6, 2012.