International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 09 | Sep-2016 www.irjet.net p-ISSN: 2395-0072

HOW TO DESIGN AND TEST SAFETY CRITICAL SOFTWARE SYSTEM

Anjaly Joy[1], Nadiya Abdul Kalam[2], Aswathy M[3]

[1]*Anjaly Joy, M TECH Student, Rajadhani Institute of Engineering and Technology, Trivandrum, Kerala, India*
[2]*Nadiya Abdul Kalam, Aswathy M, M TECH Students, Rajadhani Institute of Engineering and Technology, Trivandrum, Kerala, India*
[3]*HOD, Dept. of Computer Science & Engineering, Rajadhani Institute of Engineering and Technology, Kerala, India*

Abstract - *Safety is the foremost need that every human being desire. Nowadays software is associated with almost every field, 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 (SCS).These kinds of risks are handled using safety engineering techniques. Today industries have designed various different standards for the development of these Safety Critical Systems like ISO 9000,IEN 61508,RTCA/DO 178B.The designing of a SCS system identifies the hazards and constraints as early as possible. Basically two approaches are used to design a SCS; they are Formal method based approach which is a mathematical based model and the Prevention and recovery based approach which uses bottom up structure to check the error. Various techniques like Probabilistic Risk Assessment (PRA), Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA) are explained here*

Key Words: **Safety Critical System, Ada, Failure, Malfunction.**

1. INTRODUCTION

A safety-critical software system is a system whose failure or malfunction can severely harm people's lives, environment or equipment. These kinds of risks are managed using techniques of safety engineering. Safety-critical systems are widely used in various different fields such as medicine, nuclear engineering, transport, aviation, aerospace, civil engineering, industrial simulation, process control, military devices, telecommunications, infrastructures, etc. Safety-critical systems consist of hardware equipment and software equipment and both of them have to be secure in order to ensure that the whole system is fully secure. The main aim is to provide a brief overview of safety-critical software systems and describe the main techniques or approaches used to design and test these kinds of systems. For this, consider the broader notion of testing which comprises all the development cycle of a software product without limiting the scope of testing only to the testing of code. The first section focuses on the basic standards used and applied in different fields for the development of safety-critical systems. The next section focuses on the programming features and languages recommended, then will go on to describe different approaches on designing safety-critical software systems. Two main approaches will be considered. Finally, it will then outline the main techniques used to test these kinds of particular systems and also examples of tools used to test real systems as well as companies or institutions using the techniques mentioned will be provided. There are basically three approaches to achieving reliability in a safety critical system:

1. Testing.
2. Formal Specification and Verification
3. Automatic Program Synthesis

1.1 Testing

Testing is the process of identifying defects, where a defect is any variance between actual and expected results. Testing in Safety Critical System is the process of executing a software system to determine whether it matches its specification and executes in its intended environment.

1.2 Formal Specification and Verification

Verification is the act of proving the correctness of intended algorithms. Verification is the process of determining that a system or module meets its specification. Formal methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired. This formal description can be used to guide further development activities (see following sections); additionally, it can be used to verify that the requirements for the system being developed have been completely and accurately specified. The motivation for proving the correctness of a system is not the obvious need for re-assurance of the correctness of the system, but a desire to understand the system better. Consequently, some proofs of correctness are produced in the style of mathematical proof: handwritten (or typeset) using natural language, using a level of informality common to such proofs. A "good" proof is one which is readable and understandable by other human readers.

1.3 Automatic Program Synthesis

Program synthesis is a special form of automatic programming that is most often paired with a technique for formal verification. The goal is to construct automatically

a program that provably satisfies a given high-level specification. In contrast to other automatic programming techniques, the specifications are usually non-algorithmic statements of an appropriate logical calculus.

2. STANDARDS OF SAFETY CRITICAL SYSTEM

Industries have designed various different standards for the development of these safety critical systems. Various standards like ISO 9000, IEC 61508 , RTCA/DO 178B are used in the development of Safety Critical System.

2.1 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. ISO 9000 requires the organization to "say what it does, do what it says, and be able to demonstrate it".
The main standards are:

Table -1: ISO Sub Standards

ISO Standard	DOMINE
ISO9002	Model for quality assurance in design, development, production, installation, servicing.
ISO9002	Model for quality assurance in production, Installation and servicing.
ISO9002	Model for quality assurance in final inspection and test.

ISO 9001 focuses on product conformity to international standards throughout the product lifecycle. Emphasis is put on the design element and performance factors. This standard is the most stringent of the ISO 9000 series.

ISO 9002 does not include a design or R&D component. It is focused on the production, installation and servicing of products.

ISO 9003 covers product inspection and testing of ready-made components .For the development of software ISO 9003 is important because it guide the application of ISO 9001 to the development, supply and maintenance of software.

2.2 IEC 61508

IEC 61508 is a standard developed by the IEC (International Electro technical Commission), a worldwide organization consisting of IEC National Committees in more then 60 countries of the world. IEC prepares and publishes international standards for all electrical, electronic and related technologies. These server as a basis for national standardization and as reference when drafting international contracts.

IEC 61508, titled "Functional safety of electrical/electronic/programmable electronic safety-related (E/E/PE) systems" is a generic standard and consists of 7 parts:

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

The standard aims of IEC 61508 is:

A. Release the potential of E/E/PE technology to improve both safety and economic performance

B. Enable technological developments to take place within an overall safety Framework.

C. Provide a technically sound, system based approach, with sufficient flexibility for the future.

D. Provide a risk-based approach for determining the required performance of safety-related systems.

B. Provide a generic standard that can be used directly by industry but can also help with developing sector standards (e.g. chemical plants, medical, or railway) or product standards (e.g. drive-by-wire).

F. Provide a means for users and regulators to gain confidence when using computer-based technology.

G. Provide requirements based on common underlying principles to facilitate by improved efficiencies in the supply chain for suppliers of subsystems and components to various sectors and also by improvements in communication and requirements.

2.3. BS EN 50128 (Railway Industry)

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". The standards were adopted to define a process for the specification and demonstration of dependability requirements for the railway industry and to promote a common understanding and approach to the management of dependability. EN 50128 provide Railway Authorities and the railway support industry,
Throughout the European Community, with a process which will enable the implementation of a consistent approach to the management of Reliability, Availability, Maintainability and Safety. EN 50128 introduces software integrity levels (SILs), where each level associated with a degree of risk when using the software system.

Table -2 SIL Levels

SIL Level	RISK
0	Non safety related
1	Low
2	Medium
3	Very High

2.4. RTCA/DO 178B

The Requirements and Technical Concepts for Aviation (RTCA) DO 178B is officially a "guidance document" but widely accepted as an international standard. It was first issued in the United States in the 1980s and relates to civil aircraft. The last
Released version is a major revision from 1992.The standard is compatible with the European Organization for Civil Aviation Electronics (EUROCAE) standard ED-12B. The intent of DO-178B is to describe the objectives of software life-cycle processes, describe the process activities, and to describe the evidence of compliance required at different software levels. The software levels are chosen by determining the severity of failure conditions on the aircraft and its occupants (DO-178B).

The verification objectives for DO-178B are set in place to detect and report errors that may have been introduced during the software development processes. Software verification objectives are satisfied through a combination of reviews and analyses, the development of test cases and procedures, and the subsequent execution of those test procedures. Reviews and analysis provide an assessment of the accuracy, completeness, and verifiability of the software requirements, software architecture, and source code. Most software code is written in a high level language such as C, C++ or Ada, and the coverage achieved by any given test is usually measured against high-level source code (also referred to as Structural Coverage).
The development of test cases may provide further assessment of the internal consistency and completeness of the requirements. The execution of the test procedures provides a demonstration of compliance with the requirements. Software test cases should be based primarily on the software requirements and developed to reveal potential errors. Software coverage analysis is used to determine which requirements were not tested. This is supported by the structural coverage analysis objectives required by DO-178B that are intended to determine what software structures (e.g. statements or decisions) were not exercised as a result of these verification activities. This, in turn, reveals requirements that may have been in error, tests that were lacking adequate coverage for these structures, or dead code. Structural coverage analysis is performed to the degree required by the criticality of the software. Structural coverage analysis may be performed on the source code; unless the software level is A and the compiler generates object code that is not directly traceable to source code statements. Additional verification should then be performed on the object code to establish the correctness of such generated code sequences.

3 .SAFETY CRITICAL SYSTEM PROGRAMMING PLATFORM

The design of safety critical 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 are:

i. Dynamic allocation / de-location of memory.

ii. Use of pointers

iii. Use of unstructured programming constructs like

iv. Go to.

v. Multiple entry and exit points in a loop, procedure

vi. Functions.

vii. Recursion

viii. Procedural parameters

On the other hand, other programming features which provide reliability and are less likely to lead to errors are:

- Strong typing.
- Runtime constraint checking.
- Parameter checking.

Among the programming languages with the good features required mentioned above, Ada stands out as one of the most reliable and secure, however, as none programming language is perfect, the common approach is to use a small subset of it in order to avoid the risky features and make the most of the reliable ones. The Ada subset most commonly used for safety-critical software systems is called SPARK.

The Ada language defines many run time checks that are required to raise exceptions if they fail. As any Ada programmer knows, these checks and resulting exceptions are enormously valuable in finding errors in the early stage of testing, rather than later on in the development process. The issue of whether such checks should be enabled in the final product is an interesting one. On the one hand, it would prefer to demonstrate that a program is free of any possibility of run time errors. On the other hand, it provides an extra safety belt in case of a problem sneaking through our careful procedures. But for sure such run time checking is invaluable during the testing process. Ada is an example of a language that is designed with this criterion in mind. Programmers learning Ada for the first time often comment that it is hard work to get the compiler to accept a program, but when it does, the program is far more likely to run correctly the first time. That characteristic may be a bit annoying for rushing out programs rapidly where reliability is not paramount, but for safety-critical programming it is just what we want. Ada achieves this partly by implementing a much more comprehensive type system, in which for example there are multiple integer types, and the compiler can check at compile time that you are not doing something that makes no sense like adding a length to a time.

The languages like C, and Java and C++ are not suitable for writing safety-critical software. The extended functionality of modern programming languages does make it easier to write code in the first place, but we have to worry about demonstrating that the resulting code is error-free; so that the programs should be written in a very well understood subset of the chosen language, which avoids unnecessarily complex semantics. For instance, in Ada, we most likely avoid using the full power of Ada tasking. In C, we exclude some of the C library routines which are unlikely to be available in a safety-critical environment. For C++ we avoid the complex use of templates. For Java, we avoid the use of dynamic features that allow the program to modify itself while it is running.

4. DESIGNING SAFETY CRITICAL SYSTEMS

The basic idea of designing safety-critical software systems is to identify hazards as early as possible in the development life-cycle and try to reduce them as much as possible to an acceptable level.
Mainly two approaches they are:

Formal method based approach.
Prevention and recovery based approach.

The first approach considered is to formally prove that the system does not contain errors by construction by means of formal methods. Formal methods are mathematical techniques and tools used to specify, design and verify software systems. 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
.

4.1 Formal method based approach.

Formal method based approach 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. Formal methods are mathematical techniques and tools used to specify, design and verify software systems. Specifications are written as well-formed statements using logic mathematical language and formal proves are logic deductions using rules of inference. Mathematical proofs can also be faulty. So whereas verification might reduce the program-testing load, it cannot eliminate it. It is also almost impossible to formally prove everything used to develop the system such as the compiler, the operating system in which the system will ultimately operate and in general every underlying program used to build the target critical system. That makes necessary the use of specialized tools to help with formal specifications and proves. There are already some tools for that but they are not completely satisfactory up to date and it remains as a developing task. For small systems, where formal specifications and proves are easier to deal with, the approach can be very successful. The technique used to overcome problems with large scale systems is to try to separate the critical functionality of the system from the other non-critical parts. This way of using components with different safety integrity levels works well providing it is proved that the non-critical components cannot affect the high integrity ones or the whole system.

In Specifying software a hands-on introduction by R.D. Tennent, gives an example of formal methods applied to a real case with successful results. 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 2000 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.
Formal based approach consist of 3 levels:

LEVEL 0:

Formal specification may be undertaken and then a program developed from this informally this is the most cost-effective option in many cases. Proofs may be undertaken to confirm property by assuming that the result is true.

LEVEL 1:

Formal development and formal verification may be used to produce a program in a more formal manner. For example, proofs of properties or refinement from the specification to a program may be undertaken. Formal methods used to develop process by using Rules and Design calculus. This may be most appropriate in high-integrity systems involving safety or security.

LEVEL 2:

Theorem provers may be used to undertake fully formal machine-checked proofs. This can be very expensive .In this level maximum number of error is eliminated.(e.g., in critical parts of microprocessor design).

4.2. 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.

The Prevention and recovery based 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. Redundancy techniques systems mainly used in aircrafts, where some parts of the control system may be replicated. 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.

5. TESTING SAFETY CRITICAL SYSTEM

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. 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.

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.

Some well-known techniques used to generate test cases to test these kinds of systems are white box and black box testing and reviews. However, they are taken to a further level of detail than with typical systems. For instance, according to IPL, reviews become more formal including techniques such as detailed walkthroughs of even the lowest level of design and also the scope of reviews is extended to include safety criteria. If formal mathematical methods have been used during the specification and design, then formal mathematical proof is a verification activity indeed. To give a real example, Hewlett-Packard generates test cases using white box and black box techniques to test their patient monitors of the HP Omni Care Family.

Complex static analysis techniques with control and data flow analysis as well as checking that the source code is consistent with a formal mathematical specification are also used. Tools such as SPARK Examiner are available for that. Dynamic analysis testing and dynamic coverage analysis are also performed using known techniques such as equivalence partitioning, boundary value analysis and structural testing. IPL has developed tools such as Ada test and Cantata to give support for dynamic testing and dynamic coverage to the levels of functionality required by standards for safety-critical software.
The two important factors that distinguish legacy software testing techniques from safety critical testing techniques are:

Degree

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.

Organizational structure:

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

5.1 Techniques for testing and verifying Safety Critical System

Some of the specific techniques from safety engineering to test and verify safety-critical software system are explained below.

5.2.1 Probabilistic Risk Assessments (PRA).

Probabilistic Risk Assessment is a systematic methodology to evaluate risk associated with a complex engineering technologies entity (such as an airliner or a nuclear power plants). The steps involved in PRA for testing Safety Critical System are:

1. Perform a primary hazard analysis to find out the predefined hazard on Safety Critical System.

2. The severity of each impact is calculated. The severity levels can be classified as

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. The assessment of risk is calculated by combining both impact and probability of occurrence in matrix.

For this evaluation we use different risk criteria like risk-cost trade-offs, risk benefit of technological options, etc. Risks that fall into the unacceptable category (e.g.: high severity and high probability), that is to say, are unacceptable, must be mitigated by some means such as safeguards, redundancy, prevention and recovery mechanisms, etc., to reduce the level of safety risk. Probabilistic risk assessment also uses tools such as cause and effect diagrams. For instance, HP applies these techniques to their patient monitors naming it as risk and hazard analysis and they consider it to be a grey box method.

5.2.2 Failure Modes and Effect Analysis (FMEA).

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. Failure modes, 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.

5.2.3 Fault Tree Analysis (FTA).

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 labelled with actual numbers about failure probabilities. The probability of the top level event can be determined using mathematical techniques.
FTA can be used to:

a) Understand the logic leading to the top event / undesired state.

b) Show compliance with the (input) system safety / reliability requirements.

c) Prioritize the contributors leading to the top event – Creating the Critical Equipment/Parts/Events lists for different importance measures.

d) Monitor and control the safety performance of the complex system.

e) Minimize and optimize resources.

f) Assist in designing a system. The FTA can be used as a design tool that helps to create (output / lower level) requirements.

g) Function as a diagnostic tool to identify and correct causes of the top event.

6. CONCLUSIONS

A basic overview of safety-critical software systems has been given and some standards to cope with the development of Safety Critical System are also named. Programming features and languages related to these kinds of systems have also been mentioned. Then, the two main approaches like Formal

method based approach and Prevention and recovery based approach; used when designing safety-critical software were explained. Finally, some techniques used to test safety-critical software have been described, 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

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] IPL Information Processing Ltd, an Introduction to Software Testing, Testing Papers.

[6] 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.