International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 3, Issue 4, July-August 2014 ISSN 2278-6856

A Systematic Approach for the Design of Safety Critical Systems

Ch.Dinesh[1]

[1]Sr.Asst.Professor,
Department of CSE,
Dadi Institute of Engineering & Technology,NH-5,
Anakapalle-531002,VISAKHAPATNAM

Abstract: ***A brief overview of the fields that must be considered when designing safety-critical systems is presented. The design of safety critical systems has been adopted static techniques to minimize error detection and fault tolerance. This paper specifies basic design approach by identifying the basic components of a safety critical computer system mishap causes and risk factors. Examines the design approach that implements safety and reliability. This paper also deals with some implementation issues***.

Keywords— Safety Critical System, Risk, Safety, Mishap, Risk Mitigation, Fault tolerance .

1. INTRODUCTION

Critical systems are systems in which defects could have a dramatic impact on human life, the environment or significant assets. Such systems are expected to satisfy a variety of specific qualities including reliability, availability, security and safety [1]. A real-time system is safety critical when its incorrect behavior can directly or indirectly lead to a state hazardous to human life.

A safety critical system is a system where human safety is dependent upon the correct operation of the system [6] [10]. However, safety must always be considered with respect to the whole system including software, computer hardware, other electronic and electrical hardware, mechanical hardware and operators or users not just the software element.

Defining Safe:

The notion of safety comes when we drive a car, fly on an airliner, or take an elevator ride [11]. In each case, we are concerned with the threat of a mishap, which the US Department of Defense defines as an unplanned event or series of events that result in death, injury, occupational illness, Damage to or loss of equipment or property, or damage to the environment.

The mishap risk assesses the impact of a mishap in terms of two primary concerns: its potential severity and the probability of its occurrence [13]. For example, an airliner crash would affect an individual more severely than an automobile fender-bender, but it rarely happens. This assessment captures the important principle that systems such as cars, airliners, and nuclear plants are never absolutely safe. It also provides a design principle: Given our current knowledge, we can never eliminate the possibility of a mishap in a safety-critical system; we can only reduce the risk that it will occur [7].

Risk reduction increases the system cost. In some applications such as in nuclear energy, safety dominates the total system cost. When creating a safe system by minimizing cost forces us to compromise to the extent that We expend resources to reduce mishap risk, but only to a level considered generally acceptable.

2. BASIC SAFETY DESIGN APPROACH

Typically, Any computer system—whether it's a fly-by-wire aircraft controller, an industrial robot, a radiation therapy machine, or an automotive antiskid system—contains five primary components [13]:

The application is the physical entity that the system monitors and controls. Sometimes developers refer to an application as a process. Typical applications include an aircraft in flight, a robotic arm, a human patient, and an automobile brake.The sensor converts an application's measured physical property into a corresponding electrical signal for input into the computer. Developers sometimes refer to sensors as field instrumentation. Typical sensors include accelerometers, pressure transducers, and strain gauges.The effector converts an electrical signal from the computer's output to a corresponding physical action that controls an application's function. Developers sometimes call an effect as an actuator or final element. Typical effectors include motors, valves, brake mechanisms and pumps. The operator is the human or humans who monitor and activate the computer system in real time [13]. Typical operators include an airplane pilot, plant operator, and medical technician. The computer consists of the hardware and software that use sensors and effectors to monitor and control the application in real time. The computer comes in many forms, such as a single board controller, programmable logic controller, airborne flight computer, or system on a chip.Many computer systems, such as those used for industrial supervisory control and data acquisition; consist of complex networks built from these basic components.

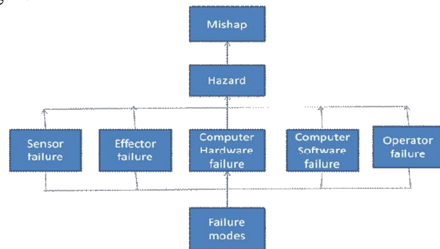
Mishap Causes:

In the basic computer system, developers fully define the application, including all hardware, software and operator functions that are not safety related [9]. Because the basic computer system employs no safety features, it probably will exhibit an unacceptably high level of mishap risk. When this occurs, solving the design problem requires modifying the operator, computer, sensor, and effector components to create a new system that will meet an

acceptable level of mishap risk.The design solution begins with the question, how can this basic computer system fail and precipitate a mishap? The key element connecting a failure in the basic system to a subsequent mishap is the hazard, defined as any real or potential condition that can cause

- injury, illness, or death to personnel;
- damage to or loss of a system, equipment, or property;
- damage to the environment.

Hazard examples include loss of flight control, nuclear core cooling, or the presence of toxic material or natural gas. All such hazards reside in the application. Thus, system design focuses first on the application component of the system to identify its hazards. Then designers must have their attention to the operator, sensor, computer, and effector components.To determine how these components can fail and cause a mishap, the designers perform a failure-modes analysis to discover all possible failure sources in each component. These include random hardware failures, manufacturing defects, programming faults, environmental stresses, design errors, and maintenance mistakes. These analyses provide information for use in establishing a connection between all possible component failure modes and mishaps, as Figure 1shows. With this analytical background in place ,actual design can begin.

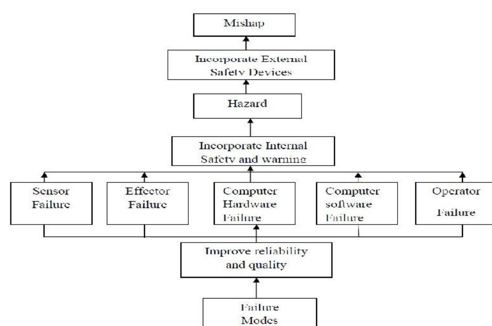


Mishap Risk Mitigation Measures

For any given the system having a high risk of mishap, design attention turns to modifying it to mitigate this risk. We can do this in three ways:

1) Improve component reliability and quality
2) Incorporate internal safety and warning devices
3) Incorporate external safety devices

Figure 2 shows how and where applying these mishap-risk-mitigation measures can alleviate the computer system mishap causes shown in Figure 1.



Improving reliability and quality involves two measures: improving component reliability and exercising quality measures. Reliability improvement seeks to reduce the probability of component failure, which in turn will Reduce mishap probability [8]. A widely used and effective approach for improving reliability employs redundant hardware and software components. Redesign can remove component reliability problems. Other sources of component failure such as procedural deficiencies , personnel error are difficult to find.

Although reliability and quality measures can reduce mishap risk, they normally will not lower it to an acceptable level because component failures will still occur. If the project requires additional risk mitigation steps ,internal safety devices used as defense. Even after designers have taken these measures, system failure still continues, resulting in mishaps. Finally external safety devices are used as last line of defense against these residual failures [6]. External safety devices range from simple physical containment through computer based safety instrumented systems. To achieve effective mishap risk mitigation ,developers usually apply all three of these mitigation measures to create a layered approach to system protection.In addition ,risk mitigation efforts must be distributed evenly across the system ‘s sensor, effector, computer and operator components because single failure in any of the part of the system can make the aggregate mishap risk totally unacceptable.

3. EVALUATING SAFETY CRITICAL COMPUTER SYSTEMS

The design of any safety critical system must be as simple as possible ,taking no unnecessary risks.Software point of view ,this usually involves minimizing the use of interrupts and minimizing the use of concurrency within the software.Ideally,a safety critical system requiring a high integrity level would have no interrupts and only one task.However,this is not achievable in practice.There are two distinct philosophies for the specification and design of safety critical systems[2].

•To specify and design a "perfect" system, which cannot go wrong because there are no faults in it, and to prove that there are no faults in it.

•To aim for the first philosophy, but to accept that mistakes may have been made, and to include error detection and recovery capabilities to prevent errors from actually causing a hazard to safety [10].

The first of these approaches can work well for small systems, which are sufficiently compact for formal mathematical methods to be used in the specification and design, and for formal mathematical proof of design correctness to be established.

The second philosophy, of accepting that no matter how careful we are in developing a system, that it could still contain errors, is the approach more generally adopted. This philosophy can be applied at a number of levels:

•Within a routine, to check that inputs are valid, to trap errors within the routine, and to ensure that outputs are safe.

•Within the software, to check that system inputs are valid, to trap errors within the Software, and to ensure that system outputs are safe [14].

•Within the system, as independent verification that the rest of the system is behaving correctly, and to prevent it from causing the system to become unsafe. The safety enforcing part is usually referred to as an interlock or protection subsystem.After designers have applied measures to mitigate mishap risk to a basic system, they must determine if the modified system design meets an acceptable level of mishap risk [13]. In order to determine this three analytical techniques are used.Failure modes and effects analysis(FMEA),the designer looks at each component of the system,considers how that component can fail,then determines the effects each failure would have on the system.Fault Tree Analysis(FTA) reverses this process by starting with an identified and known mishap and downward to identify all the components that can cause a mishap and all the safety devices that can mitigate it.Risk analysis(RA), In contrast to FMEA and FTA ,which are both qualitative methods ,risk analysis(RA) is quantitative measure that produces probabilities of mishap. If the risk calculation yields an acceptable result, the design is ready for additional validation steps such as risk assessment,testing and field trials to assure that the system,when implemented ,will be safe.

4. DESIGN PATTERNS FOR SAFETY AND RELIABILITY

A pattern is a generalized solution to a common problem. A pattern is instantiated and customized for the particular problem at hand, but provides a means for capturing design knowledge by capturing best practices from experienced designers [4]. There are several design patterns that affect both safety and reliability.

- Homogeneous Redundancy Pattern
- Diverse Redundancy Pattern
- Monitor-Actuator Pattern
- Safety executive Pattern

Software Fault Tolerance:Fault tolerance is intended to handle faults when they occur in an executing system. Since software faults are expected to exist, they need to be managed, i.e. avoided, removed, evaded, or tolerated.Fault Prevention: There are two means of fault prevention: avoiding their introduction during production and removing them before deployment. In both cases faults are dealt with prior to execution Fault avoidance�wis a design activity that attempts to prevent faults from being introduced into the deployed system [3]. Fault removal‘ is an implementation activity focused upon testing.Fault Tolerance:In contrast to fault avoidance, fault tolerance schemes consider faults inevitable and deal with them after deployment.There are two types of fault tolerances.:Static and Dynamic.Fault Detection:Errors may be detected by both implicit and explicit checks. Implicit checks are produced by the underlying virtual machine including both hardware checks (e.g., divide by zero) and software checks (e.g., null pointer dereference) implemented by the compiler and the run-time support environment. Explicit checks are those by which a program checks its own dynamic behaviour at run-time.Fault Location:It is implantation specific.The location of the risk has to be identified by applying fault location mechanisms.Fault Containment: An assessment of the propagation of the fault and an action to contain that propagation are necessary because there is latency between the occurrence of an error and its detection.Error Recovery:Error recovery is the central component of dynamic fault tolerance strategies – it must transform a faulty system into one with a valid, perhaps degraded, state [Burns01]. Two approaches to error recovery have been identified: backward error recovery and forward error recovery.Backward error recovery: Backward error recovery mechanisms attempt to simulate the reversal of time to a point at which the system state was error-free. They do so by saving state when it is assumed to be valid and then restoring that state as necessary. The act of saving state is called _taking a checkpoint.forward error recovery: Forward error recovery mechanisms attempt to make selective changes to an erroneous state in order to move to a new error-free state.

5. IMPLEMENTATION

Some programming language features prone to problems than others. This is because of number of reasons [1][8]. Those are

1)Programmers do errors while using the feature.
2)Poor compilation or poor implementation.
3)Programs written may be difficult to analyse and test.

Few programming language features that cause problems:

1)Usage of pointers :It is very difficult to use the pointers in programming language .Inorder to use pointers ,the developers need great understanding of memory address and management.Programs which use pointers can be difficult to understand or analyze.

2)Memory Management: The memory allocation and deallocation is related to pointers.every programmer allocate memory but sometimes they forget to deallocate .Compilers and operating systems frequently fail to fully recover deallocated memory. The result is errors which are dependent on execution time, with a system mysteriously failing after a period of continuous operation.

3)Multiple Entry and Exits: More number of exit and entry points to loops, blocks, procedures and functions, is really just a variation of unstructured programming. However, controlled use of more than one exit can simplify code and reduces the risk.

4)Type of Data : where the type of data in a variable changes, or the structure of a record changes, is difficult to analyze, and can easily confuse a programmer leading to programming errors.

5)Declaration & Initialisation: A simple spelling mistake can result in software which compiles, but does not execute correctly. In the worst case individual units may appear to execute correctly, with the error only being detectable at a system level. Declaration must be perfect.

6)Parameter Passing: passing one procedure or function as a parameter to another procedure or function, is difficult to analyze and test thoroughly.

7) Recursion:Recursion is calling a function itself.It is difficult to analyze and test thoroughly. Recursion can also lead to unpredictable real time behaviour.

8) Concurrecy and Interrupts:These features are supported directly by some programming languages only.Use of concurrency and interrupts is some what produce ambiguity.

The use of such programming language features in safety critical software is discouraged.Most modern programming languages encourage the use of block structure and modular programming, such that programmers take good structure for granted. Well structured software is easier to analyze and test, and consequently less likely to contain errors.The features of few programming languages which can be used to increase reliability are:

1) Perfect data usage:The data is only used and assigned where it is of a compatible type.
2) Constraint checking: Ensure that arrays bounds are not Violated, that data does not overflow, that zero division Does not occur
3) Parameter checking:To ensure that parameters passed to or from procedures and functions are of the correct type, are passed in the right direction (in or out) and contain valid data.

There are no commonly available programming languages which provide all of the good language features. The solution is to use a language subset, where a language with as many good features as possible is chosen, and the bad features are simply not used [12]. Use of a subset requires discipline on behalf of the programmers and ideally a subset checking tool to catch the occasional mistake. An advantage of a subset approach is that the bounds of the subset can be flexible, to allow the use of some features in a limited and controlled way.Ada is the preferred language for the implementation of safety critical software because it can be used effectively within the above constraints.

6. SAFER DESIGN

Even with a safe design,it is possible to increase or decrease device safety.Safety of the device depends on how the software is written [3][4] .The issues related to coding for safety are language selecton and usage of safe coding styles.Languages that provide strong compile time and runtime checking are considered safer.Some of the issues for safer design are:

1) Language choice
2) Compile time checking
3) Runtime checking
4) Exceptions Vs Error codes
5) Use of safe and Lanuage subsets

Some of the guidelines and rules for preparing safety critical code proposed by Gary Holtz are [5]

- Restrict to simple control flow constructs.
- Give all loops a fixed upper-bound.
- Do not use dynamic memory allocation after initialization • Limit functions to no more than 60 lines of text.
- Use minimally two assertions per function on average.
- Declare data objects at the smallest possible level of scope.
- Check the return value of non-void functions, and check the validity of function parameters.
- Limit the use of the preprocessor to file inclusion and simple macros.
- Limit the use of pointers. Use no more than one level of dereferencing.
- Compile with all warnings enabled, and use source code analyzers.

7. CONCLUSION

Designing safety–critical systems is a complex thing involving several fields. This paper describes about how to engineer safe mechanical systems than safe computing systems. In safety critical systems the importance is on using a safety process rather than specifying techniques for ensuring safety and reliability. This paper will give an analysis of safety critical system means about design, implementation etc. Although safety critical systems have been in use for many years, the development of safety critical software is still a relatively new and immature subject.New techniques and methodologies for safety critical software are a popular research topic with universities, and are now becoming available to industry. Tools supporting the development of safety critical software are now available, making the implementation of safety critical standards a practical prospect.

REFERENCES

[1] N.Leveson, Safeware: System Safety and Computers, Addison Wesley, 1995.

[2] L.Pullum, Software Fault Tolerance: Techniques and Implementation, Artech House, 2001

[3] W.R. Dunn, *Practical Design of Safety- Critical Computer Systems*, Reliability Press, 2002.

[4] Kopetz, H.,*Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

[5] Conmy, P., Nicholson, M., Purwantoro, Y.,M. And McDermid, J. (2002) *Safety Analysis and Certification of* Open *Distributed Systems*.

[6] J. A. McDermid, The cost of COTS, *IEE Colloquium – COTS and Safety critical systems* London,1998.

[7] Tindell, K., "Analysis of Hard Real-Time communications",Real-Time Systems,vol 9,pp,147-171,1995.

[8] Jesty, P.H., Hobley, K.M., Evans, R., and Kendall, I., "Safety Analysis of Vehicle-Based Systems," proceedings of the 8th Safety-critical Systems Symposium, 2000.

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

[10] John C. Knight. "Safety Critical Systems: Challenges and Directions" *Proceedings of the 24th International*

Conference on Software Engineering (ICSE), Orlando, Florida, 2002.

[11] Raghu Singh. "A Systematic Approach to Software Safety". *Proceedings of Sixth Asia Pacific Software Engineering Conference* (APSEC), Takamatsu, Japan ,1999.

[12] N. G. Leveson "Software Safety: Why, what, and how". *ACM Computing Surveys*, 18(2):125-163, June 1986.

[13] The University of York. Safety critical systems engineering, system safety engineering: Modular MSc, diploma, certificate, short courses1999. The University of York, Heslington, U.K.; www.cs.york.ac.uk/MSc/SCSE.

[14]The Hazards Forum. Safety-related systems: Guidance for engineers. The Hazards Forum (1995). London, U.K.; www.iee.org.uk/PAB/SCS/hazpub.htm.

Chandrasekharam Dinesh is a Sr. Assistant Professor in the Department of Computer Science & Engineering, Dadi Institute of Engineering and Technology (Affiliated to JNTUK), Anakapalle, Andhra Pradesh, India. He Completed M.Tech in Computer Science . His main research interests are Software engineering, Operating System, Distributed Systems.