Global Journal of Computer Science and Technology
Volume 11 Issue 21 Version 1.0 December 2011
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Safety Critical Systems Analysis

By K. Amarendra, A. Vasudeva Rao
Dadi Institute of Engineering & Technology Visakhapatnam Dt, India

Abstract - A brief overview of the fields that must be considered when designing, implementing safety-critical systems is presented. The notion of safety is most likely to come to mind when we drive a car, fly on an airliner, or take an elevator ride. In each case, we are concerned with the threat of a mishap, which defined 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.

Keywords : *Safety, Design, Implementation, Applications.*

GJCST Classification : *K.6.5*

Strictly as per the compliance and regulations of:

Safety Critical Systems Analysis

K. Amarendra [α], A. Vasudeva Rao [Ω]

Abstract- A brief overview of the fields that must be considered when designing, implementing safety-critical systems is presented. The notion of safety is most likely to come to mind when we drive a car, fly on an airliner, or take an elevator ride. In each case, we are concerned with the threat of a *mishap*, which defined 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.

Keywords: *Safety, Design, Implementation, Applications,*

I. Introduction

A safety critical system is a system where human safety is dependent upon the correct operation of system. Safety is considered not only for software elements but also for hardware, electrical hardware, operators or users etc. If the failure of a system could lead to consequences that are determined to be unacceptable then the system is safety critical.

Safety-critical systems, a term whose customary meaning is systems whose failure might danger human life, lead to substantial economic loss, or cause extensive environmental damage. Many modern systems depend on computers for their correct operation. The future is likely to increase dramatically the number of computer systems that we consider to be safety-critical. The dropping cost of hardware, the improvement in hardware quality, and other technological developments ensure that new applications will be sought in many domains.

Traditional Systems

Traditional areas that have been considered the home of safety-critical systems include medical care, commercial aircraft, nuclear power, and weapons. Failure in these areas can quickly lead to human life being put in danger, loss of equipment, and so on. Computerized equipment is making inroads in procedures such as hip replacement, spinal surgery, and ophthalmic surgery. In all three of these cases, computer controlled robotic devices are replacing the surgeons traditional tools, and providing substantial benefits to patients.

Non Traditional Systems

The scope of the safety-critical system concept is broad, and that breadth has to be taken into account when practitioners and researchers deal with specific systems. Some of the examples of Non Traditional systems are transportation control, banking and financial systems, electricity generation and distribution, telecommunications, and the management of water systems. All of these applications are extensively computerized, and computer failure can and does lead to extensive loss of service with consequent disruption of normal activities.

Separating safety-critical and safety-related systems from systems where safety integrity is unable to be established or maintained is an important aspect of system safety design. When implementing a system safety program it is important to ***suspect all*** components as being unsafe unless assured otherwise and then ***target the few*** areas where safety requirements are allocated. Coupling between components of complex systems can be subtle and interaction with non-safety related systems have led to harmful outcomes in safety related systems.

Traditional Areas

Traditional areas that have been considered the home of safety-critical systems include medical care, commercial aircraft, nuclear power, and weapons. Failure in these areas can quickly lead to human life being put in danger, loss of equipment and so on. Computers are used in medicine far more widely than most people realize. The idea of using a microprocessor to control an insulin pump is quite well known. The fact that a pacemaker is largely a computer is less well known. The extensive use of computers in surgical procedures is almost unknown except by specialists. Computerized equipment is making inroads in procedures such as hip replacement, spinal surgery, and ophthalmic surgery. In all three of these cases, computer controlled robotic devices are replacing the surgeons traditional tools, and providing substantial benefits to patients.

Non Traditional Areas

The scope of the safety-critical system concept is broad, and that breadth has to be taken into account when practitioners and researchers deal with specific systems. A closer examination of the topic reveals that many new types of system have the potential for very high consequences of failure, and these systems should probably be considered safety-critical also. It is obvious that the loss of a commercial aircraft will probably kill people. It is not obvious that loss of a telephone system could kill people.

Author [α] : Associate Professor & Head, Department of Computer Science & Engineering,Dadi Institute of Engineering & Technology, Anakapalle – 531002, Visakhapatnam Dt, India.
E-mail : hodcse@dietakp.com
Author [Ω] : Associate Professor, Department of Computer Science & Engineering,Dadi Institute of Engineering & Technology, Anakapalle – 531002, Visakhapatnam Dt, India .
E-mail : vasudevarao@dietakp.com

Emergency service is an example of a critical infrastructure application. Other examples are transportation control, banking and financial systems, electricity generation and distribution, telecommunica - tions, and the management of water systems. All of these applications are extensively computerized, and computer failure can and does lead to extensive loss of service with consequent disruption of normal activities. In some cases, the disruption can be very serious. Widespread loss of water or electricity supply has obvious implications for health and safety. Similarly, widespread loss of transportation services, such as rail and trucking, would affect food and energy distribution. It is prudent to put the computer systems upon which critical infrastructures depend into the safety-critical category.

II. Safety Boundaries

Functional Safety Boundaries

a) The need for having boundaries

Taking the extreme position, very few systems are fully independent in their operation and to be completely assured of the absence of interaction or common cause failure between the safety-related and other systems would take an inordinate amount of time and effort. This could cause the opposite effect to delay introducing the safety benefits of the deployment of a safety-related system. At some point a determination must be made that all possible influences are controlled or risks sufficiently known so the safety analysis can be bounded.

b) Objectives of functional safety boundaries

Minimise the interfaces across the safety boundary to direct focus on the safety separation implemented in these;

Minimise likelihood of common-cause failures across the boundary;

Exclude non safety related functions where these are volatile or subject to undefined or non-safety related controls;

Allow a Safety Integrity Level (SIL) to be achieved within the boundary.

c) Identifying safety functions

A useful method to establish the functional safety boundary between systems or subsystems is to undertake a Fault Tree Analysis (FTA) of the contributing factors to failure of the system, which may lead to hazardous events identified in the preliminary hazard analysis. The first attempt at a boundary would be around the systems that are implicated in the FTA. This FTA needs to be extensive and complete from all initiating situations to the system failure that is a casual factor for the hazardous event. Then flowing down the tree, mark off those functions that are related to systems that should be excluded due to:

- The possibility of common-cause failure;
- High levels of complexity and non-deterministic Failure rate; or
- Components that may not always be present or enabled.

d) The problem with software

At a system level, this process looks reasonably straightforward but the problem comes with setting boundaries with distributed software architectures. In this situation it is very difficult to identify boundaries that don't involve the possibility of common-cause failures.

Common-cause failures and dependencies extending over the distributed communication networks must also be considered and the functional safety boundary set accordingly. These may include:

- Global variables accessed by network
- Security attack and security blocking issues
- Affects of network lock-up on functional safety

The separation requirements over the functional safety boundary must take these failures into account.

III. Disciplines

The criterion used is that these disciplines are at the heart of the safety-critical electronic and information technology components of modern vehicles. Safety-critical systems have many requirements that stem from several engineering disciplines. The main disciplines having a direct bearing on designing safety critical systems are: domain engineering, embedded systems engineering, protocol and network engineering, safety engineering, reliability engineering, real-time systems engineering, and systems engineering.Currently, several design and implementation options are available to a researcher, developer, or designer. In terms of protocols, one can choose among CAN, TTCAN, Switched Ethernet, TTP/C, Flex Ray and others. Because of cost, flexibility, the intended application theoretical, advances implementation technology, and other issues, it is not straightforward to decide what protocol or network technology is the best.

a. *Domain Engineering :* Safety-critical systems exist in a certain application context. Certainly the details of safety-critical aerospace systems are different from those of the space shuttle, process control, or automotive. It is important that it can be used to tune or optimize certain mechanisms (e.g., communications, fault tolerance, fail status, etc)
b. *Embedded System Engineering : S*afety-critical systems are embedded systems such as micro-controllers; real-time operating systems, memory configurations, and I/O are relevant.
c. *Protocol and Network Engineering :* Protocols and networking are at the heart of distributed safety-

critical systems. The degree of flexibility offered by the protocol is in order to experiment with and provide higher layer protocols (HLP). Still another issue is the application of inter-networking (using bridges, switches, and routers) at the vehicle level.

d. *Safety Engineering :* system (availability) reliability deals with the problem of ensuring that a system performs a required task or mission (at) for a specified time. *System safety* is concerned with ensuring that a *mishap* does not occur in the process. Usually, there are some failures exits like *benign failures and catastrophic failures.*
e. *Reliability Engineering :* It deals with the available operation of a system even under the failure of system components. The primary mechanism is the use of redundant components to design fault tolerant systems. There are two schemes to handle the replacement of failed components. They are *static and dynamic redundancy.*
f. *Real time Engineering :* Techniques for ensuring that a system meet timeliness requirements are important for safety-critical applications. A distinction is made between hard real-time and soft real-time systems. Safety-critical systems certainly belong to the category of hard real-time systems. To see if a system meets real-time requirements, schedulability analysis is used and this methodology is well known for single-processor or multiprocessor operating systems.
g. *Systems Engineering :* System engineering emphasizes formal processes that start with a system's requirements and specification, and includes an iterative design, test, and verification cycle.

IV. Applications

Safety critical systems are whose failure results in loss of life, property damage or damage to the environment. There are many well known examples in application areas such as medical devices, aircraft flight control weapons and nuclear systems.

Example of a safety critical system is an aircraft fly by wire control system, where the pilot inputs commands to the control computer using a joystick, and the computer manipulates the actual aircraft controls. The lives of hundreds of passengers are totally dependent upon the continued correct operation of such a system.

Moving down to earth, railway signalling systems must enable controllers to direct trains, while preventing trains from colliding. Like an aircraft fly by wire, lives are dependent upon the correct operation of the system. However, there is always the option of stopping all trains if the integrity of the system becomes suspect. You can't just stop an aircraft while the fly by wire system is fixed!

Software in medical systems may be directly responsible for human life, such as metering safe amounts of X-rays. Software may also be involved in providing humans with information, such as information which a doctor uses to decide on medication. Both types of system can impact the safety of the patient.

Big civil engineering structures are designed on computers and tested using mathematical models. An error in the software could conceivably result in a bridge collapsing. Aircraft, trains, ships and cars are also designed and modelled using computers.

Even something a simple as traffic lights can be viewed as safety critical. An error giving green lights to both directions at a cross road could result in a car accident. Within cars, software involved in functions such as engine management, anti-lock brakes, traction control, and a host of other functions, could potentially fail in a way which increases the likelihood of a road accident.

V. Challenges

In one way or another, many people in the software business are working on safety-critical systems technology. Many more systems than one might expect have to be viewed as safety-critical and the number is increasing all the time. So what are the major challenges that we face?

In some cases, what amount to completely new technologies is required? The number of interacting safety-critical systems present in a single application will force the sharing of resources between systems. This will eliminate a major architectural element that gives confidence in correct operation—physical separation. Knowing that the failure of one system cannot affect another greatly facilitates current analysis techniques. This will be lost as multiple functions are hosted on a single platform to simplify construction and to reduce power and weight requirements. Techniques that provide high levels of assurance of non-interference will be required.

Breakdowns in the interplay between software engineering and systems engineering remains a significant cause of failures. It is essential that comprehensive approaches to total system modelling be developed so that properties of entire systems can be analysed. Such approaches must accommodate software properly and provide high fidelity models of critical software characteristics. They must also deal with the issue of assured non-interference.

Defective software specifications are implicated in many serious failures, and it is clear that we have difficulty stating exactly what software is required to do. There are many aspects of specification that are not supported by any current technique, and, even where specification techniques do exist, there remains a lack of integration to permit whole specification analysis.

VI. Designing

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.

- 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 is 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.

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;
- 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. Designing safety-critical systems is a complex endeavour particularly if extensive use of advanced electronics and information technology is used. The increased use of microcontrollers in modern automotive systems has brought many benefits such as the merging of chassis control systems for active safety with passive-safety systems. Unfortunately, it has also brought the potential for catastrophic failures. Thus, the widespread application of microcontrollers requires extreme care in order to produce a dependable system. Dependability involves reliability, safety, availability, and security but in this paper we are only concerned with safety, reliability, and availability.

The area of system safety is well-established and procedures exist to identify and analyse electromechanical hazards along with techniques to eliminate or limit hazards in a final product. Unfortunately, much more is known about how to engineer safe mechanical systems than safe computing systems, particularly when software is a major component of the engineered system. With the increased used of software in safety-critical components of complex systems, governments agencies and other institutions are increasingly including requirements for software hazard analysis and verification of software safety.

Security : It has become clear that security attacks against information systems are a large and growing problem. Attacks against both public and private networks can have devastating effects. The Internet is being used increasingly to provide communication service to business, and security attacks against the Internet are a troubling problem for network users.

Although Internet attacks are important, private networks are a bigger concern. Money is moved locally and around the World on private networks owned by financial institutions. Transportation systems are monitored and controlled using mostly private networks. A successful attack against certain private networks could permit funds or valuable information such as credit card numbers to be stolen, transportation to be disrupted, and so on. The potential for loss is considerable, and, although no physical damage would be involved in security failures, the consequences of failure are such that many systems that only carry information should be regarded as safety-critical.

VII. Implementation

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

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

Few programming language features that cause problems:

1) *Usage of pointers :* It is very difficult to use the pointers in programming language .In order 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 de-allocation is related to pointers. every programmer allocate memory but sometimes they forget to de-allocate .Compilers and operating systems frequently fail to fully recover de-allocated 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 & Initialization :* 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 behavior.
8) *Concurrency 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 : T*o 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. 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. The most popular Ada subset for safety critical software is the SPARKAda subset.

SPARKAda is a subset of the Ada Programming Language that restricts several features of Ada such as unrestricted tasking. SPARKAda includes a built-in toolset called the "Examimator" which tests the entire source code for conditional and unconditional data flow errors which in theory would deem the source code exception free. The disadvantage to SPARKAda is that is closed and proprietary which increases the cost of implementation. Since it is a closed format, outside community support is restricted and there is a higher risk of implementation with only one vendor to rely on for technical support and language updates.

VIII. Testing & Verification

Safety critical testing : Testing of safety-critical systems follows two important strategies which are systematic rigorous testing and static analysis. While there is no substitute for rigorous testing at many levels: Unit, regression, functionality and integration testing, testing effectiveness depends on the quality of the test cases used. The best test suites are those that have good code coverage. Statement coverage and condition Coverage are the most commonly used metrics. Full Condition coverage is considered essential for safety-critical code, such as flight control software. Achieving full coverage can be exceedingly time-consuming and expensive.

Safety critical software functions provide the source of requirements to be tested. Testing shall be performed to verify correct incorporation of software safety requirements. Testing must show that hazards have been eliminated or controlled to an acceptable level of risk. Additional hazardous states identified during testing shall undergo complete analysis prior to software delivery or use. Software safety testing of Safety-Critical Computer Software Components (SCCSC) shall be included in the integration and integration and acceptance tests. Acceptance testing shall verify correct operation of the SCCSCs in conjunction with system hardware and operators[36]. It shall verify correct operation during stress conditions and in the presence of system faults. It is important to tailor the safety-critical testing effort to emphasize the parts of the software that need additional analysis and testing. The greatest effort must be placed on the hazards posing the highest risk. We consider it adequate to divide the software into two risk groups for test purposes.

Verification is the most important and most expensive group of activities in the development of safety critical systems, with verification activities being associated with each stage of the development lifecycle. An added complication is that *independent verification* is usually required. The means by which this is achieved depends upon the integrity level. Independent

verification can vary from independent witnessing of tests, participation at reviews and audit of the developer's verification, to fully independent execution of all verification activities. Independent verification is an addition to verification conducted by developers, not a substitute for it.

According to ISO 9001 activity, reviews will be conducted as a part of verification. Reviews become more formal, including techniques such as detailed walkthroughs of even the lowest level of design. The scope of reviews is extended to include safety criteria.

IX. Conclusion

The choice of a language can have a significant impact on the success or failure of a safety-critical system. The language can impact the ease of validation, the number of defects, and many important parts of the development process. Few languages are inherently "safe" as well as having good tool support, good documentation and wide usage.

A general-purpose language, which is made "safe" by use of a subset and good tool support, is the best choice for a safety-critical system. Modelling languages show excellent promise as implementation languages for all types of software development, not just safety critical.

Safety critical software is a complex subject. This paper will give an analysis of safety critical system means about design, implementation, verification, Applications 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 References Referencias

1. 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.
2. Alan C. Tribble et al. "Software Safety Analysis of a Flight Guidance System", *Proceedings of the 21st Digital Avionics Systems Conference* (DASC'02), Irvine, California, Oct. 27-31, 2002.
3. Debra S. Herman, "*Software Safety and Reliability Basics*", (ch.2), Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors Wiley-IEEE Computer Society Press, 2000.
4. Dale M. Gray. *Frontier Status Report* #203, 19 May 2000, www.asi.org
5. John C. Knight. "Safety Critical Systems: Challenges and Directions" *Proceedings of the 24th International Conference on Software Engineering* (ICSE), Orlando, Florida, 2002.
6. N.Leveson, Safeware: *System Safety and Computers*, Addison Wesley, 1995.
7. L.Pullum, *Software Fault Tolerance: Techniques and Implementation*, Artech House, 2001.
8. W.R. Dunn, *Practical Design of Safety-Critical Computer Systems*, Reliability Press, 2002.
9. Kopetz, H.,*Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.
10. Conmy, P., Nicholson, M., Purwantoro, Y.,M., and McDermid, J. (2002) *Safety Analysis and Certification of Open Distributed Systems*.
11. J. A. McDermid, The cost of COTS, *IEE Colloquium - COTS and Safety critical systems* London,1998.
12. IEC 61508 *Functional Safety of electrical / electronic / programmable electronic safety-related systems* Geneva: International Electrotechnical Commission, 1998.
13. Tindell, K., "*Analysis of Hard Real-Time communications*", Real-Time Systems,vol 9,pp,147-171,1995.
14. 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.
15. Raghu Singh. "A Systematic Approach to Software Safety". *Proceedings of Sixth Asia Pacific Software Engineering Conference* (APSEC), Takamatsu, Japan, 1999.
16. N. G. Leveson "Software Safety: Why, what, and how". *ACM Computing Surveys*, 18(2):125-163, June 1986.
17. The University of York, *Safety critical systems engineering, system safety engineering,* Modular MSc, diploma, certificate, short courses1999.
18. The University of York, Heslington, U.K.; www.cs.york.ac.uk/MSc/SCSE.
19. *The Hazards Forum, Safety-related systems: Guidance for engineers,* The Hazards Forum (1995). London,U.K.;www.iee.org.uk/PAB/SCS/hazpub.htm.