Real Time Systems Design for Safety Critical Applications

Rinku R. Dhruva
CVR College of Engineering, Department of ECE, Ibrahimpatan, R.R.District, A.P., India
Email: rinku_dhruva@yahoo.com

***Abstract*—This paper studies and suggests few of the famous techniques employed in various applications in different domains, and generalise the guidelines for designing and developing Real Time Systems in safety critical applications.**

I. Introduction

Today's electronic systems infiltrate more and more our daily life. We put our lives in the hand of complex electronic systems. For instance, during a flight with a modern aeroplane, where a severe failure of the electronic flight control system may lead to a catastrophe, we completely rely on the proper functioning of the electronic system.

In the industry, it is an upward trend to replace mechanical and hydraulic control systems by electronic control systems. An example of the automotive industry: in the model year 2001, electronics were accounted for 19% of a mid-sized automobile's costs. It is estimated that in the year 2005, 25% of the total costs of a mid-sized automobile will be accounted for electronic parts, and possibly 50% for luxury models. This includes costs for so-called 'by-wire systems', which will replace traditional mechanical and hydraulic braking and steering systems in cars of the near future (model years 2005–2007).

In a by-wire system, braking or steering relies on the correct behavior of the electronic system. A failure of the electronic system may cause a severe hazard that will endanger human life or the environment. The design, development, production, and maintenance of such a by-wire system is a complex and difficult undertaking, and system failures during operational use have to be prevented by all possible technical means. The difficulties are mainly caused by the complexity of these electronic systems, mass production, and stringent dependability (e.g., safety) requirements imposed by authorities. Among others, a validation of the system's behavior in all stages of the system's life-cycle is a necessary and important technical mean to have confidence that the system under consideration behaves safe in its environment.

A. Real-time system:

A real-time system has to interact with its environment in real-time. The correctness of a real-time system depends not only on the logical result of the computation but also on the time at which the results are produced [1] (see also [2, p. 18–19]). The point in time by which the result must be produced for the temporal behavior of the response to be correct is called deadline.

The paucity of material on safety critical systems has lead to widespread misunderstanding of the various terms used to discuss safety. The most basic term is safety. Safety is defined to be freedom from accidents or losses. An accident is an event in time in which an undesirable consequence occurs, such as death, injury, equipment damage, or financial loss.

A safety-critical system in a system, which may contain electronic, mechanical, and software aspects, that presents an opportunity for accidents to occur. For many people, safety-critical systems are only those that present the opportunity for injury or loss of life, but this omits from consideration other systems which might benefit from the techniques and approaches common in safety analysis. Therefore, it is better to designate a safety critical system to be any system in which the cost of use of a system due to an accident is potentially high. Hazard is the effect of safety failure.

For example, the FDA[2] uses major (irreversible injury or death), moderate (injury), and minor (no injury) levels of concern for device safety. The German standard DIN 19250 identifies 8 categories, along with required safety measures for each category while the more recent IEC 61508 [3] identifies 4 safety integrity levels (SILs): catastrophic, critical, marginal, and negligible, although the text notes that the severity of system-presented hazards is actually a continuum.

The risk of a hazard is defined to be the product of the probability of the occurrence of the hazard and its severity:s.

Riskhazard = probabilityhazard x severityhazard

Being shocked by your car battery is relatively high but when combined with the low severity, the overall risk is low. Similarly, while the consequences of an abrupt release of the kinetic energy of a commercial aircraft are quite severe, its probability is low " again resulting in a low risk. The various standards also identify different risk levels based on both the severity of the hazard and its likelihood of occurrence.

In the process of system design, hazards must be identified and safety measures must be put in place to reduce the risk.

II. Systems Approach

System safety engineering has historically demonstrated the benefits of a "systems" approach to safety risk analysis and mitigation. When a hazard

DOI: 10.32377/cvrjst0111

analysis is conducted on a hardware subsystem as a separate entity, it will produce a set of unique hazards applicable only to that subsystem. However, when that same subsystem is analyzed in the context of its physical, functional, and zonal interfaces with the rest of the "system components," the analysis will likely produce numerous other hazards which were not discovered by the original analysis. Conversely, the results of a system analysis may demonstrate that hazards identified in the subsystem analysis were either reduced or eliminated by other components of the system. Regardless, the identification of critical subsystem interfaces (such as software) with their associated hazards is a vital aspect of safety risk minimization for the total system. When analyzing software that performs, and/or controls, safety-critical functions within a system, a "systems approach" is also required. The success of a software safety program is predicated on it. Today's software is a very critical component of the safety risk potential of systems being developed and fielded. Not only are the internal interfaces of the system important to safety, but so are the external interfaces.

III. The Hardware Development Life Cycle

A. *Figures and Tables*

The typical hardware development life cycle has been in existence for many years. It is a proven acquisition model which has produced, in most instances, the desired engineering results in the design, development, manufacturing, fabrication, and test activities. It consists of five phases. These are identified as the concept exploration and definition, demonstration and validation, engineering and manufacturing development, production and deployment, and operations and support phases. Each phase of the life cycle ends, and the next phase begins, with a milestone decision point (0, I, II, III, and IV). An assessment of the system design and program status is made at each milestone decision point, and plans are made or reviewed for subsequent phases of the life cycle.

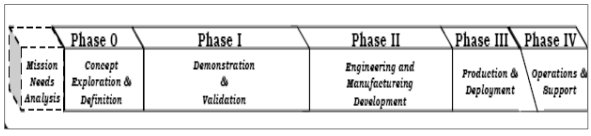

Figure 1. Hardware Development Life Cycle

The one shown in Figure identifies and establishes defined phases for the development life cycle of a system and can be overlaid on a proposed timetable to establish a milestone schedule.

IV. The Software Development Life Cycle

The system safety is critically depend on the software life cycle being used by the development activity. In the past several years, numerous life cycle models have been identified, modified, and used in some capacity on a variety of software development programs. The important issue here is to recognize which ever model is being used, but decide how to correlate and integrate safety activities with the chosen software development model to achieve the desired outcomes and safety goals. Several different models will be presented to introduce examples of the various models.

Figure2 is a graphical representation of the relationship of the software development life cycle to the system/hardware development life cycle. The model is representative of the "Waterfall," or "Grand Design" life cycle. While this model is still being used in numerous projects, other models are more representative of the current software development schemes currently being followed, such as the "Spiral" and "Modified V" software development life cycles. It is important to recognize that the software development life cycle does not correlate exactly with the hardware (system) development life cycle. It "lags" behind the hardware development at the beginning but finishes before the hardware development is completed. It is also important to realize that specific design reviews for hardware usually lag behind those required for software.

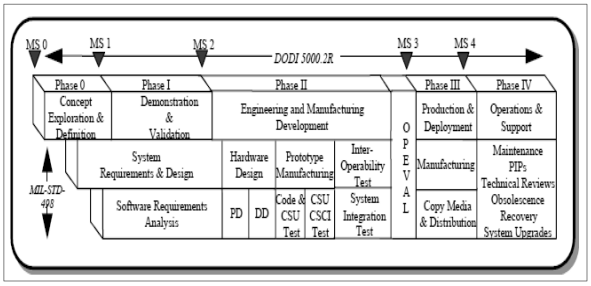

Figure 2. Relation between Software and Hardware Development Life Cycle

A. *Grand Design, Waterfall Life Cycle Model*

The Waterfall software acquisition and development life cycle model is the oldest in terms of use by software developers. Grand Design places emphasis on up-front documentation during early development phases, but does not support modern development practices such as prototyping and automatic code generation. Another limitation to the model is that after a single pass through the model, the system is complete. Therefore, many integration problems are identified much too late in the development process to be corrected without significant cost and schedule impacts. In terms of software safety, interface issues must be identified and rectified as early as possible in the development life cycle to be adequately corrected and verified. Figure 3 is a representation of the Grand Design, or Waterfall, life cycle model. The Waterfall model is not recommended for large, software-intensive, systems. This is due to the limitations stated above and the inability to effectively manage program risks, including safety risk during the software development process. The Grand Design does, however, provide a structured and well-disciplined method for software development.

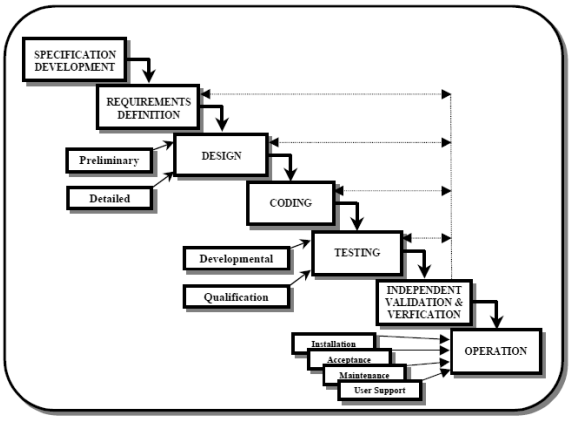

Figure 3. Grand Design, Waterfall LIFE Cycle Model

B. *Modified V Life Cycle Model*

The Modified V software acquisition life cycle model is another example of a defined method for software development. It is depicted in Figure 4. This model is heavily weighted in the ability to design, code, prototype, and test in increments of design maturity. The left side of the figure identifies the specification, design, and coding activities for developing software. It also indicates when the test specification and test design activities can start. For example, the system/acceptance tests can be specified and designed as soon as software requirements are known. The integration tests can be specified and designed as soon as the software design structures are known. And, the unit tests can be specified and designed as soon as the code units are prepared. The right side of the figure identifies when the evaluation activities occur that are involved with the execution and testing of the code at its various stages of evolution.

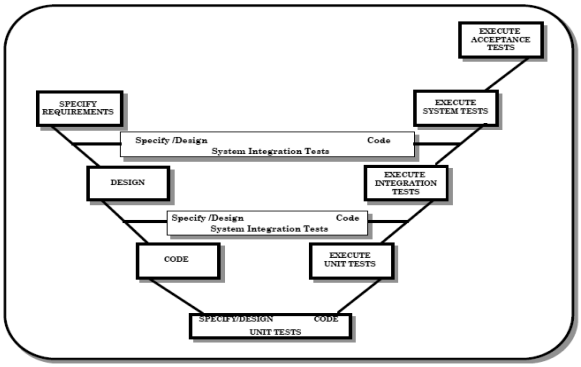

Figure 4. V Life Cycle Model

C. *Spiral Life cycle Model*

The Spiral acquisition life cycle model provides a risk-reduction approach to the software development process. In the Spiral model, Figure 5, the radial distance is a measure of effort expended, while the angular distance represents progress made. It combines features of the Waterfall and the incremental prototype approaches to software development. Spiral development emphasizes evaluation of alternatives and risk assessment. These are addressed more thoroughly than with other strategies. A review at the end of each phase ensures commitment to the next phase or identifies the need to rework a phase if necessary. The advantages of Spiral development are its emphasis on procedures, such as risk analysis, and its adaptability to different development approaches.

This model represents a "demonstration based" process that employs a top-down incremental approach that results in an early and continuous design and implementation validation. Advantages of this approach are that it is built from the top down, it supports partial implementation; the structure is automated, real and evolved; and that each level of development can be demonstrated. Each build and subsequent demonstration validates the process and the structure to the previous build. Hence, Spiral Life-cycle model is more appropriate for safety-critical systems design.

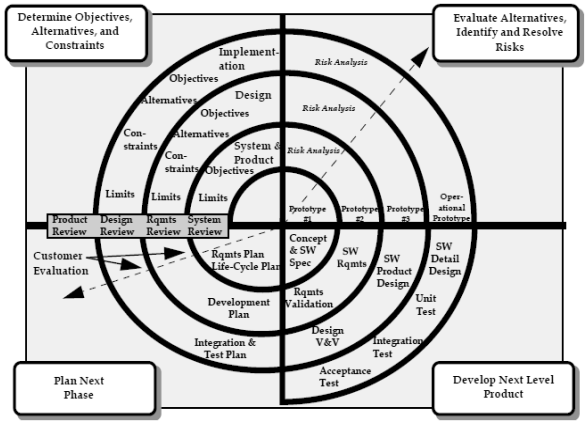

Figure 5. Spiral Life Cycle Model

V. INTEGRATION

The life cycle process of system development was instituted so managers would not be forced to make snap decisions. A structured life cycle, complete with controls, audits, reviews, and key decision points, provides a basis for sound decision making based on knowledge, experience, and training. It is a logical flow of events representing an orderly progression from a "user need" to finalize activation, deployment, and support.

The elements contributing to a credible and successful software safety engineering program will include the following:

- A defined and established system safety engineering process,
- A structured and disciplined software development process,
- An established hardware and software systems engineering process,
- An established hardware/software configuration control process, and
- An integrated SSS Team responsible for the identification, implementation, and verification of

safety-specific requirements in the design and code of the software.

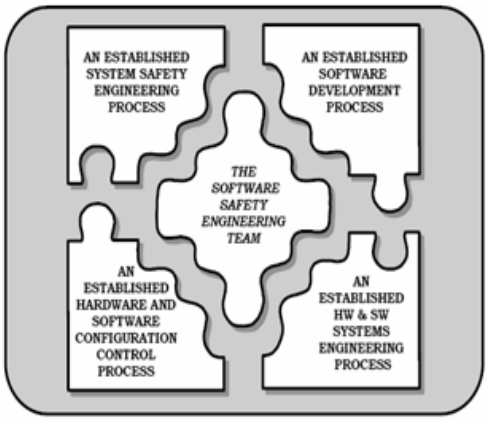

Figure 6. Software Safety Engineering Program

VI. Redundancy for Fault-tolerant Design

Fault tolerant design, also known as fail-safe design, is a design that enables a system to continue operation, possibly at a reduced level (also known as graceful degradation), rather than failing completely, when some part of the system fails. The term is most commonly used to describe computer-based systems designed to continue more or less fully operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software.

Redundancy is the duplication of critical components of a system with the intention of increasing reliability of the system, usually in the case of a backup or fail-safe.

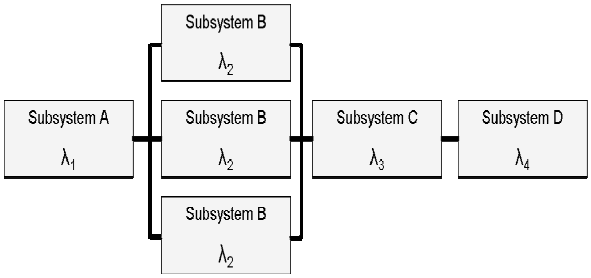

Figure 7. Redundancy in System

A. Replication

Replication is the process of sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility [4]. It could be data replication if the same data is stored on multiple storage devices, or computation replication if the same computing task is executed many times. A computational task is typically replicated in space, i.e. executed on separate devices, or it could be replicated in time, if it is executed repeatedly on a single device.

A lockstep fault-tolerant machine uses replicated elements operating in parallel. At any time, all the replications of each element should be in the same state. The same inputs are provided to each replication, and the same outputs are expected. The outputs of the replications are compared using a voting circuit. A machine with two replications of each element is termed dual modular redundant (DMR). The voting circuit can then only detect a mismatch and recovery relies on other methods.

In many safety-critical systems, some parts of the control system are triplicated, which is formally termed triple modular redundancy (TMR). A machine with three replications of each element is termed triple modular redundant (TMR). The voting circuit can determine which replication is in error when a two-to-one vote is observed. In this case, the voting circuit can output the correct result, and discard the erroneous version. After this, the internal state of the erroneous replication is assumed to be different from that of the other two, and the voting circuit can switch to a DMR mode. This model can be applied to any larger number of replications.

VII. Testing, Verification and Validation

Systems are called safety-critical if their malfunction represents a severe threat to human lives or to the environment. Following Laprie's terminology[3], dependability is the capability of a system to deliver the specified application services during its period of operation. Laprie identified four attributes which characterise the dependability of a system:

(1) A safe system cannot assume states that are regarded as "catastrophic" from the point of view of the application. This means that the system will only perform transitions into states satisfying the specified invariants, perform calculations that are correct with respect to the specification and output data fulfilling the desired integrity constraints. Safety does not guarantee that a desired calculation and the corresponding output will always be produced. This aspect is covered by the following two attributes:
(2) Reliability is a characteristic specifying the probability that a system will deliver its service for a given period of time.
(3) Availability is a measure reflecting the probability that the system will be available at a certain point in time.
(4) Finally, Security reflects the capability of the system to protect the application against damage arising from accidental or malicious human interaction.

Design, execution and evaluation of tests for safety-critical systems require considerable effort and skill and consume a large part of today's development costs. Due to the growing complexity of control systems it has to be expected that their trustworthy test will become unmanageable in the future if only conventional techniques requiring a high degree of human interaction during the test process are applied. For these reasons methods and tools helping to automate the test process gather wide interest both in industry and research communities.

CONCLUSIONS

This paper tried to touch the fundamental points to be taken care while designing a safety-critical real time system. With the advent growth in technology and tools in future the designers can eliminate most of the hazards at design stage itself, and improve the system reliability and safety, if understand these concepts.

REFERENCES

[1] John A. Stankovic. Misconceptions About Real-time Computing: A Serious Problem for Next Generation Systems. IEEE Computer, 21(10):10–19, October 1988.

[2] Jerey J. P. Tsai, Yaodong Bi, Steve J. H. Yang, and Ross A. W.Smith. Distributed Real-Time Systems: Monitoring, Visualiza-tion, Debugging, and Analysis. John Wiley & Sons, New York et al., 1996.

[3] J. C. Laprie et al. Dependability: Basic Concepts and Terminology. Springer-Verlag, 1992. and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] David Alberico et al., Joint Software System Safety Committee - SOFTWARE SYSTEM SAFETY HANDBOOK, Joint Services Computer Resources Management Group, December 1999.