

Study of Methods for Analyzing and Building Software on Safety Critical Systems

B.Umamaheswararao

*Research Scholar Department of CS&SE,
Andhra University College of
Engineering, Visakhapatnam,
India*

Dr.P. Dr. Panchumarthy Seetha Ramaiah,

*Department of CS&SE,
Andhra University College of
Engineering, Visakhapatnam
India*

ABSTRACT

A safety critical system is a system where human safety is dependent upon the correct operation of the system. The use of formal methods is often advocated as a way of increasing confidence in such systems. This paper examines the usage of these methods, the differences concerning formal methods and methodology for building safer software for the development of safety-critical systems and reviews existing software safety standards, guidelines and other software safety documents and also examines the limitations, practical problems and issues associated with the use of current software safety standards. Some possible future directions are suggested. Safety problems arise with the introduction of computers into safety-critical systems. Building safety-critical software requires special procedures to be used in all phases of the software development process. The Safety-Critical Systems have become more important as computers are increasingly used to monitor and control critical devices and processes in desperate areas like medicine, transportation, energy, manufacturing, etc. Several Design Methods and Metrics have been developed for the safety and security of the Safety-Critical Systems. Methods like FTA, FMEA, FMFEA, FMECA, ETA, MORT, SMORT, etc. are intended to provide a safe and secure Safety-Critical System. These methods are intended to avoid the mishaps in the critical devices. But the occurrence of accidents didn't stop due to the faults in the Safety-Critical Systems and are continuing.

Keywords: Safety-critical systems, Software safety, Design methods.

1. INTRODUCTION

Safety-critical systems are those systems whose failure could result in loss of life, significant property damage, or damage to the environment [12]. Modern electronic systems increasingly make use of embedded computer systems to add functionality, increase flexibility, controllability and performance. However, the increased use of embedded software to control systems brings with

it certain risks. The increased flexibility and complexity can lead to new and different failure modes which cannot be addressed with traditional fault tolerance techniques [1]. This is especially significant in "safety-critical systems".

A safety critical system [9] is a system where human safety is dependent upon the correct operation of the system. An obvious 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. The development of safety critical systems has traditionally been pioneered within the avionics and automotive industries but, as awareness has developed, of how software can impact safety, the scope of safety critical software has expanded into many types of systems such as medical instruments and devices, transport, process control, nuclear and oil and gas facilities.

Safety requirements and failure modes and consequences in medical systems will be quite different from those in avionics systems or automotive systems for instance, and the approaches used to ensure safety may consequently be different too. This paper is aimed at exploring the approaches used to develop embedded software in some of the different safety-critical applications, with a view to establishing any common approaches and identifying opportunities for sharing best practice and development tools and techniques.

1.1 METHODS :

Safety-critical systems, by definition those systems whose failure can cause catastrophic results for people, the environment, and the economy, are becoming increasingly complex both in their functionality and their interactions with the environment. Unfortunately, safety assessments are still largely done manually, a time-consuming and error-prone process. There are methods to analyze the drawbacks of the safety-critical systems like Failure Mode and Effects Analysis (FMEA),

Failure Modes Effects and Criticality Analysis (FMECA), Fault Tree Analysis (FTA), Event Tree Analysis (ETA), Failure Mode Factors and Effects Analysis (FMFEA), etc.

1.2. Software Induced Failures in Real-life

Computers are increasingly being introduced into safety critical systems and, as a consequence, have been involved in accidents. Some well known incidents are the Therac-25 accidents [13], the Ariane 5 explosion. Some of the most widely cited software related accidents in safety critical systems involved a computerized radiation therapy machine called the Therac-25. Between June 1985 and January 1987, six known accidents involved massive overdoses by the Therac-25 – with resultant deaths and serious injuries. They have been described as the worst series of radiation accidents in the 35-year history of medical accelerators. On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. The rocket was on its first voyage, after a decade of development costing \$7 billion. The destroyed rocket and its cargo were valued at \$500 million. A board of enquiry which investigated the causes of explosion found out that the cause of the failure was a software error in the inertial reference system. Specifically a 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,767, the largest integer storable in a 16 bit signed integer, and thus the conversion failed.

The rest of this paper is organized as follows.

Section 2 describes Design Methods Safety Critical systems.

Section 3 presents The Computer Based Systems and Mishaps.

Section 4 presents Methodology for modeling software safety in safety-critical computing systems.

Section 5 describes safety issues of Railroad Crossing Control System (RCCS) prototype and the results observed after application of the methodology.

Section 6. **Conclusion**

2. DESIGN METHODS:

2.1. Failure Mode and Effects Analysis (FMEA):

Failure Mode and Effects Analysis (FMEA) is an analytical method of the preventive quality assurance. It serves to find the potential failure of a product/process, to recognize and evaluate its importance and to identify appropriate actions to

prevent the potential failure or to discover it in time. The systematic analysis and removal of weak points leads to the minimization of risks, to the reduction of failure costs and to improved reliability. FMEA is a good means to analyze risks caused by individual failures. The individual risks are weight against each other to recognize priorities. FMEA does not provide a statement on the total failure risk. For the analysis of failure combinations, the fault-tree analysis is more appropriate.

Types of FMEA

- **Process:** analysis of manufacturing and assembly processes
- **Design:** analysis of products prior to production
- **Concept:** analysis of systems or subsystems in the early design concept stages
- **Equipment:** analysis of machinery and equipment design before purchase
- **Service:** analysis of service industry processes before they are released to impact the customer
- **System:** analysis of the management system functions
- **Software:** analysis of the software functions

Areas of Application:

FMEA should be applied:

- Where the criticality of the software or system under consideration must be analyzed, to reduce the risk of failure (e.g. safety critical systems like aircraft flight control systems).
- where mandatory or regulatory requirements apply to remove defects at early stages
- To define special test considerations, operational constraints, design decisions for safety critical systems.

2.2. Failure Modes Effects and Criticality Analysis (FMECA):

Failure Modes Effects and Criticality Analysis (FMECA) is a quality tool which builds on the results of Functional Analysis to identify risks and their consequences. FMECA can be applied to systems, products, manufacturing processes, equipment, plant and even less tangible subjects such as logistic or information flows. It is used to identify the possible ways in which failure can occur for the corresponding causes of failure, the corresponding effects of failure, and the impact on Customer Satisfaction. The objective of FMECA is

to identify the components of products and systems most likely to cause failure, so that these potential failures can then be designed out. FMECA allows the identification early in the product development process of potential problems or safety hazards which are inherent in a product design. The safety and/or reliability of the product can be assessed and modifications initiated at a relatively low cost before they are built into the product.

2.3. Fault Tree Analysis (FTA):

Fault Trees [8] are one of the most widely used methods in system reliability and failure probability analysis. A Fault Tree is a graphical representation of events in a hierarchical, tree-like structure. It is used to determine various combinations of hardware, software, and human error failures that could result in a specified risk or system failure. System failures are often referred to as top events. A deductive analysis using a Fault Tree begins with a general conclusion or hazard, which is displayed at the top of a hierarchical tree. This deductive analysis is the final event in a sequence of events for which the Fault Tree is used to determine if a failure will occur or, alternatively, can be used to stop the failure from occurring. **Main advantage over FMEA/FMECA:** Fault Trees investigate consequences of multiple simultaneous failures or events, which investigate single-point failures.

2.4 Event Tree Analysis (ETA):

It is an inductive failure analysis performed to determine the consequences of single failure for the overall system risk or reliability. Event Tree Analysis uses similar logic and mathematics as Fault Tree Analysis, but the approach is different - FTA uses deductive approach (from system failure to its reasons) and ETA uses the inductive approach (from basic failure to its consequences). An event tree itself is a visual representation of single failure sequences, its influence on other events and on the whole system.

2.5. Failure Mode Factors and Effects Analysis (FMFEA):

FMFEA [5] is a safety analysis method used to analyze factors and effects of failure modes of structural components. This method utilizes a combination of FTA and ETA. This method is able to both discover and predict problems using a systematic progression of detailed analysis. In addition, safety (preventing personal injury) and reliability (preventing product failure) can be analyzed simultaneously. FMFEA follows the procedure listed below.

- (1) Select as subject of the investigation a new unit with an important function.

(Predefine "important", e.g., having safety functions, or controlling something else.)

(2) Check basic proper function of that unit.

(3) Define a single failure mode with "failure = loss of function".

3.0 THE COMPUTER BASED SYSTEMS AND MISHAPS

Typically, virtually 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 [14] :

Application: Physical entity the system controls/monitors,

E.g. plant, process

Sensor: Converts application's measured properties to appropriate computer input signals, e.g. accelerometer, transducer

Effectors: Converts electrical signal from computer's output to a corresponding physical action that controls function, e.g. motor, valve, break, and pump.

Operator: Human(s) who monitor and activate the computer system in real-time, e.g. pilot, plant operator, medical technician

Computer: Hardware and software that use sensors and effectors to control the application in real-time, e.g. single board controller, programmable logic controller, flight computers,

Systems on a chip.

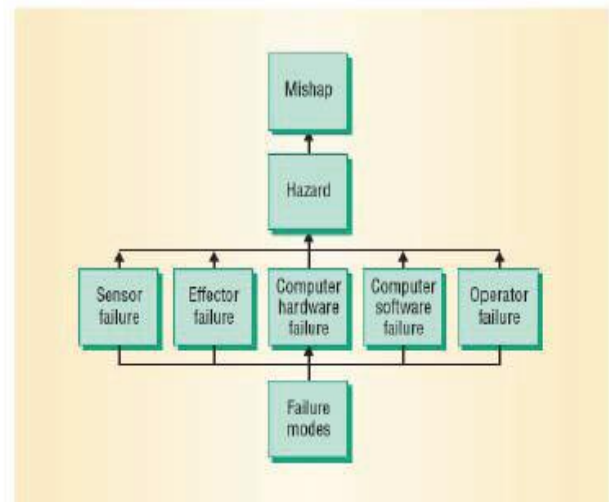


Figure 1. Mishap causes. System designers identify the application's attendant hazards to determine how system-component failures can result in mishaps.

Any of the above five components may fail and cause a mishap as shown in Fig. 1

The main concentration in this work is on Computer Software that too on Safety-Critical Software

3.1 Safety Critical Software

“Any software item identified as a potential hazard cause, contributor, control, or mitigation, whether controlled by hardware, software or human operator, is designated as safetycritical, and subjected to rigorous software quality assurance, analysis, and testing. Safety-critical software is also traced through the software safety analysis process until the final verification.

Thus, safety critical requirements need to be identified as such to insure future changes, as well as verification processes, take them into appropriate consideration.” Software is safety-critical if it resides in a safety critical system and at least one of the following applies:

- Causes or contributes to a hazard.
- Provides control or mitigation for hazards.
- Controls safety-critical functions.
- Processes safety-critical commands or data.
- Detects and reports, or takes corrective action, if the system reaches a specific hazardous state.
- Mitigates damage if a hazard occurs.
- Resides on the same system (processor) as safety-critical software.

3.2 Software Safety Involves:

1. Integrating safety into the software life cycle
2. Analyzing the software, system, and interfaces from beginning to end
3. Documenting safety plans, decisions, processes, and results
4. Tracing software safety requirements through all software phases
5. Reporting and resolving problems and discrepancies
6. Controlling software configuration
7. Evaluating off-the-shelf software

Software Safety Continues during Operations

1. Software safety applies to a system until it is retired
2. Software upgrades, updates, fixes, and other changes
3. User manuals must describe safety-related commands and data.

4.0 METHODOLOGY FOR MODELING SOFTWARE SAFETY IN SAFETY-CRITICAL SYSTEMS

The Ten tasks are:

1. Software safety Planning
2. Safety-Critical Computer System Function Identification and Description
3. Hazard Analysis

4. Software Safety Requirements Analysis,
5. Software Safety Architecture Design analysis
6. Software Safety Detailed Design Analysis
7. Software Safety Code Analysis
8. Software Safety Test Analysis
9. Software Safety Evaluation, and
10. Software Safety Process Review and Documentation.

1. Software safety planning

The purpose of software safety planning is to define the approach that will aid in producing software that will satisfy system safety requirements. Planning helps ensure that safety is designed and incorporated in from the beginning of the life cycle. Early hazard identification and risk reduction will typically provide the most effective and lowest cost approach to addressing safety concerns. Software safety plans include a System Safety Program Plan, which describes the software and hardware safety tasks and activities, and the Software Development Plan. A Software Development Plan includes management elements of safe software development (organization and responsibilities, policies and procedures, schedule and tasks, etc.) and engineering elements (hazard analyses, verification approaches, configuration management, quality assurance, etc.). Additional information about software safety planning can be found in [15].

2. Safety-critical computer system function identification

When software is integrated as part of a system to command, control, or monitor safety-critical functions, special measures are required to understand and mitigate safety risks. Therefore, it is

important first to identify those functions that are essential to safe performance or operation.

Identifying these functions helps prioritize the safety effort to focus the resources and activities on the most important safety concerns. Safety critical computer system functions are essentially those software features that are used to monitor, control, or provide data for the safety-critical functions. At this stage top-level, or generic, requirements should be defined. These requirements are in general not tied to a specific hazard but rather are derived from knowledge of the safety-critical functions, design standards, safety standards, mishap reports, experience on similar software, and lessons learned from other programs.

3. Software and computing system hazard analyses

Once the safety-critical computer system functions have been identified, perform analyses to identify the hazards, assess the risks, and identify risk mitigation approaches associated with those functions. In software-intensive systems, mishaps

often occur because of a combination of factors, including component failure and faults, human error, environmental conditions, procedural deficiencies, design inadequacies, and software and computing system errors.

In such systems software often cannot be divorced from the system where it resides. First perform a preliminary analysis that considers software hazards on a system or subsystem level as part of a larger system safety effort. Typical approaches include Preliminary Hazard Analyses and Failure Modes, Effects, and Criticality Analysis. The analysis will result in mitigation

measures to reduce risk and system-level requirements to implement those mitigation measures. In addition to the system or subsystem hazard analysis, perform software-specific hazard analyses. Software-specific hazard analyses identify what can go wrong, what are the potential effects, and what mitigation measures can be used to reduce the risk. Typical software-specific hazard analysis techniques include Software Failure Modes and Effects Analysis and Software Fault Tree Analysis. Software specific hazard analyses should consider multiple error conditions. Some of the error conditions to consider are as follows:

- Calculation or computation errors (incorrect algorithms, calculation overflow, etc.)
- Data errors (out of range data, incorrect inputs, large data rates, etc.)
- Logic errors (improper or unexpected commands, failure to issue a command, etc.)
- Interface errors (incorrect messaging, poor interface layout and design, etc.)
- Environment-related errors (improper use of tools, changes in operating system, etc.)
- Hardware-related errors (unexpected computer shutdown, memory overwriting, etc.)

The software-specific analysis should provide specific mitigation approaches for each potential hazard identified. The recommended order of precedence for eliminating or reducing risk in the use of software and computing systems is the same as that for hardware, as follows:

1. Design for minimum risk
 2. Incorporate safety devices
 3. Provide warning devices
 4. Develop and implement procedures and training
- Mitigation measures can include
- Software fault detection (for example, built-in tests, incremental auditing, etc.)
 - Software fault isolation (for example, isolating safety-critical functions from non-safety-critical functions, etc.)
 - Software fault tolerance (for example, recovery blocks that use multiple software versions of progressively more reliable construction should faults occur, etc.)

- Hardware and software fault recovery (for example, incremental reboots, exception handling, etc.)

4. Software Safety Requirements Analysis:

A Software Safety Requirements Analysis (SSRA) shall be performed and documented. The system-level PHA and the system conceptual design shall be used as input to the SSRA. The SSRA shall examine system level software requirements, interface control documents, and the ongoing software requirements specification development to:

- a. Identify software requirements that are safety critical.
- b. Ensure the correctness and completeness of the decomposition of the high level safety requirements.
- c. Provide safety-related recommendations for the design and testing process. [16]

Analysis of all software requirements [17] shall be performed in order to identify additional hazards that the system analysis did not include and to identify areas where system or interface requirements were not correctly assigned to the software. Identified hazards shall then be addressed by adding or changing the interfaces, system requirements, and/or software requirements. The SSRA shall consider such specific requirements as specific limit ranges; out-of-sequence event protection requirements (e.g., "if-then" statements); timers; voting logic; hazardous command processing requirements; Fault Detection, Isolation, and Recovery (FDIR); and switch over logic for failure tolerance. Output of the SSRA shall be used as input to follow-on software safety analyses. The SSRA shall be presented at the Software Requirements Review (SRR)/Software Specification Review (SSR) and system-level safety reviews. The results of the SSRA shall be provided to the ongoing system safety analysis activity.

5. Software safety Architecture Design Analysis:

This begins in the System and Software Architecture Design phase. Inputs into this task may include the system architecture design, the system hazard analysis outputs like PHA, safety concept etc., the safety-related design and testing recommendations from the software safety requirement analysis task, the software architecture design, the software safety requirements, and software criticality and tailoring guidelines. Software components and functions are identified in the software architecture design phase. The software components and functions that implement the software safety requirements or that affect the output of the software safety requirements are identified as safety critical. The correctness and

completeness of the software architecture design as it is related to the software safety requirements and the safety-related design recommendations is analyzed to help ensure that the design satisfies the software safety requirements. Safety-related recommendations for the detailed design and test procedures are provided, and test coverage of software safety requirements is verified.

6. Software safety detailed design analysis:

This begins in the software detailed design analysis phase. Inputs into this task include the system hazard analyses, the system and software detailed designs, the software safety requirements, software architecture design analysis output, and safety related detailed design recommendations. The identified safety critical components and functions that implement the software safety requirements are refined to the unit level software components and functions. The system and software detailed designs are analyzed to ensure that the software detailed design satisfies the software safety requirements. Subsystem interfaces may be analyzed to detect the interface problems which may lead to hazards. Test coverage of software safety requirements is verified, and safety-related recommendations for the software implementation are provided. The software safety detailed design analysis continues during a portion of implementation and unit testing also. The outputs from this task may include the identified safety-critical unit level software components and functions, the identified subsystem interfacing hazards, and safety-related software implementation and test coverage recommendations.

7. Software Safety Code Analysis:

This task begins in the software implementation and unit testing phase. Inputs into this task may include the system hazard analyses outputs, software safety requirements, software detailed design, software safety detailed design analysis output, safety related software implementation recommendations, software implementation and tailoring recommendations. The Software safety code analysis shall examine the software requirements specification, test procedures, and the ongoing code development to:

- a. Ensure the correctness and completeness of the code as related to the software safety requirements, detailed design, and safety related coding recommendations.
- b. Identify potentially unsafe states caused by input/output timing, multiple events, out-of-sequence events, failure of events, adverse environments, deadlocking, wrong vents, inappropriate magnitude, improper polarity, and hardware failure sensitivities, etc.

- c. Ensure test coverage of software safety requirements
- d. Update safety-related information for inclusion in the User's Guide and other appropriate documentation.
- e. Ensure proper comments are used in safety critical component implementation

8. Software Safety Testing and Test Analysis Software safety Test Planning:

This begins in the software architecture design phase and continues through the software integration and acceptance testing phase. During this task, appropriate software safety tests that address all identified potential hazards related to or affected by the software are incorporated into the software safety test plan.

Software safety testing and Test analysis: These tasks begin in the software implementation and unit testing phase. Inputs into the software safety testing task include the system and software safety test plans and procedures. Inputs into the software safety test analysis task include the software safety requirements, system safety program plan, software safety program plan, System and Software safety test plans and procedures and safety test results. The test results shall be analyzed to verify that all safety requirements have been satisfied. The analysis shall also verify that all identified hazards have been eliminated or controlled to an acceptable level of risk [18]. The results of the test safety analysis shall be provided to the ongoing system safety analysis activity.

9. Software Safety Evaluation

The purpose of the Software Safety Evaluation Phase is to evaluate all System and software safety analyses and test results and generate a Safety Certification Letter or Safety Analysis Report (SAR). The Safety Certification Letter provides a safety recommendation on whether or not to certify the computer program and hardware component undergoing Safety Analysis. A SAR report also provides a safety recommendation along with a summary of the findings normally found in the Final Report. Whether a Certification Letter or SAR report is provided depends on customer requirements.

10. Software Safety Process Review and Documentation

This phase allows time for final documentation. This phase also provides for review of the process and lessons learned. The lessons learned are used for Software Safety Process/Technology Improvement.

4.1 Phase Independent Tasks

The following software safety tasks are accomplished throughout the life cycle.

1. Safety Requirements Traceability
2. Discrepancy Reporting and Tracking
3. Software Change Control
4. Safety Program Reviews

1. Safety Requirements Traceability

A system shall be used to trace the flow down of the software safety requirements to design, Implementation, and test. The tracing system shall also map the relationships between software safety requirements and system hazard reports.

2. Discrepancy Reporting and Tracking

A system shall be used for closed-loop tracking of safety related discrepancies, problems, and failures in base lined software products. All discrepancy reports shall be reviewed for safety impacts, with the safety activity's concurrence on safety-related discrepancy report closures.

3. Software Change Control

All changes, modifications, and patches made to the safety critical component requirements, design, code, systems, equipment, test plans, procedures, or criteria shall be evaluated to determine the effect of the proposed change on system/subsystem safety.

4. Safety Program Reviews

Safety program reviews shall be conducted to ensure that implementation of safety controls of hazards are adequate. The software safety activity shall support the system safety review process.

Application of safety model to Railroad Crossing Control System (RCCS):

Crossing gates on full-size railroads are controlled by a complex control system that causes the gates to be lowered to prevent access to the crossing shortly before a train arrives and to be raised to allow access to resume after the train has departed. This requires the detection of approaching trains or the manual actuation of the crossing gates by an operator. RCCS is a prototype safety-critical railroad crossing control system of limited complexity. Figure 2 shows the laboratory prototype of RCCS consisting of several components listed below.

5.1 Components of RCCS

RCCS consists of the following main components: Train, Railway track, Sensors, Gates, Controller with a digital I/O card, Signals and a muscle-wire operated track change lever. A brief description of each component is given below.

Train: The train is powered by a power supply relay. When the power is initially switched on, the train begins movement along the track when the metallic wheels of the train receive power. The train comes to a halt at the position where the power to the tracks is switched off. When a train approaches the gate crossing region, the train is detected by the sensor positioned near the gate

crossing area. The sensor sends this information to the controller component. When a train completely passes the crossing section, it is detected by the sensor which is positioned after the gate crossing area. This information is sent to the controller.

Sensors: These are used to detect the location of the train on the tracks. Altogether RCCS employs nine sensors. Two pair of sensors detect the train position before and after the gates. A set of three sensors relate to track change where the track splits into two directions. A pair of sensors gives the train position with reference to the platform, which is the starting point of the train movement. Information from each of the sensors is passed to controller.



Controller: The controller synchronizes the train activities with the gate. When the controller receives a message from sensor1, it sends a command to lower the gates. When it receives a message from sensor2, it sends a command to raise the gates. An IBM compatible PC is used as a controller for RCCS. RCCS software that controls the overall operation of the system is stored in the memory of the controller PC. A user interface is provided to operate the selections of the controller PC. A 48-line digital I/O (DIO) add-on card is plugged into an available slot in the controller PC for monitoring and controlling sensors and gate actuators. The DIO card receives the inputs from each of the nine sensors of RCCS. The eight output signals sent from DIO card control the following: the power supply to the train track, power supply to the two gate assemblies, power supply to muscle-wire based mechanism to change the track lever and four signal lights.

Gates: RCCS has two sets of gates on either side of the track layout. The gate receives signals from the controller component. When it receives lower, it moves down. When the gate receives raise, it moves up. The gates are operated by means of a muscle wire based mechanism. Muscle wire (Nitinol) is a nickel titanium alloy which contracts when current flows through it, for achieving motor less motion for gate movement and track

change.

Signals: Railroad signals are provided to indicate to train operators whether the track is clear or occupied, or if certain precautionary measures should be taken while using the track, such as maintaining a reduced speed. RCCS contains three train signals, erected beside the track. One signal is at the platform to signal a halt at the platform. The other two signals are placed just before the point of convergence of the inner track and outer track, Which lead to the platform. A signal head consists of one or more signal faces that can include solid red and green lights.

5.2 Results and Discussion

Normal operation of RCCS: When RCCS is first switched on, the controller does a preliminary check of the normal working status of all the subsystems involved the driver circuitry, the sensors, the gate assemblies and the train signals. If all the components are found to be in normal working condition, it executes the code related to normal operation. Figure 3 shows the partial block diagram of RCCS corresponding to the rail-road intersection. If the train passes Sensor1 positioned prior to gate, a signal is sent to the controller indicating the approaching train. The controller then sends a signal to the gates assembly, causing the gate arms on either side of the road to close. When the train finally has passed Sensor2, which is positioned just beyond the gate crossing section, a corresponding signal is sent to the controller, which in turn triggers both the gate arms to open simultaneously. If RCCS detects any abnormal situation or state during its normal mode of operation, perhaps due to an unexpected lightning strike or rainstorm that disrupts the circuitry of the gate assemblies, it executes the code relating to emergency situation causing the signal erected near the gates, to flash a red light continuously. This is an indicator to the public that the gate assembly is not in working condition and that they need to take necessary precaution in crossing the intersection. All the tasks of the methodology were applied to RCCS. First, the system level hazard analysis was done to identify possible hazardous failure conditions at the system level. The potential hazards identified are: Failure of Controller, Failure of Sensors, Failure of Driver Circuitry, Failure of Gate 1 and Gate 2, Failure of Train Signal, Failure of muscle-wire operated Track Change Lever in changing from outer to inner track. Next, the identified hazards were classified according to their severity. A hazard belongs to one of four levels-catastrophic, critical, marginal and negligible. For example, the failure of the controller may lead to both gates being permanently open, causing accidents, can be considered a catastrophic or severe hazard. Failure of the sensor that detects that the train has passed the gate crossing section, with the effect of the

gates being permanently closed will not cause an accident but will violate the utility property of the gates, until the problem is rectified. Failure of the sensor that detects the approaching train can cause an accident as the controller will not close the gates keeping them open, which can lead to accidents as the road users are unaware of the approaching train. This is a catastrophic or severe hazard

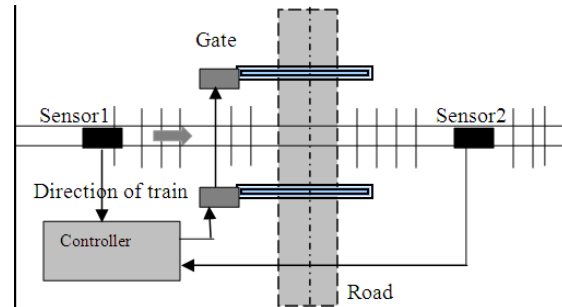


Fig. 3: RCCS partial block diagram showing railroad crossing intersection

Second, completeness of requirements is verified to check any missing or ambiguous specifications. This was done by peer review and manual checking rather than applying any formal methods.

Third, all the safety-critical and non-safety critical Requirements were identified. All requirements that directly or indirectly lead to incorrect operation of the gates are considered safety-critical. Fourth, a design that enforced the safety constraints was chosen for RCCS. The objective of the design was to eliminate or mitigate the hazards identified in the Preliminary system-level hazard analysis. Another Objective was to avoid the possibility of single point failure. This was achieved by using an additional redundant controller that takes over control of the system should the main controller fail unexpectedly. Implementation was done in Cyclone programming language which is a dialect of C language which includes several safety features not found in C. Fifth, run-time performance was monitored for problems relating to exceptions, deadlocks, memory related issues like buffer overruns. Lastly, safety critical testing of RCCS was done by separating the code into two risk groups. Group one includes hazards that are catastrophic or critical. Group two includes hazards that are marginal or negligible. More testing effort was spent on those code sections dealing with hazards related to group one. The preliminary results in applying the safety methodology in developing the safety-critical RCCS clearly demonstrate that the system is safe, risk-free and fail-safe when compared to a development methodology that does not take hazards and associated risks into consideration.

6. CONCLUSION

In the above sections the detailed description of the methods and metrics to check the safety critical systems are being analyzed and comparison of them is done. But there are drawbacks for the methods and metrics that are needed for the development of the correct safe safety-critical system. So, this paper paves the path for the development of the perfect method or the metric that helps for the development of a safety-critical system that does not have any bugs.

REFERENCES

[1] Metrics and Models in software Quality Engineering by Stephen H. Kan Second Edition, Pearson Education India, 01-Sep-2003.

[2] http://en.wikipedia.org/wiki/Life-critical_system.

[3] http://en.wikipedia.org/wiki/Nuclear_and_radiation_accidents.

[4] Recent Catastrophic Accidents: Investigating How Software Was Responsible- W.Eric Wong, Vidroha Debroy, Adithya Surampudi & HyeonJeong Kim Michael F. Siok, 2010 Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement.

[5] Hiroshi Wada, "Safety analysis methods and applications at the design stage of new product development"—Introducing the FMFEA and S-H Matrix Method.

[6] <http://www.scribd.com/doc/49760660/44/RELIABILITY-METRICS>

[7] <http://www.fault-tree.net/papers/ericson-fta-history.pdf>

[8] http://en.wikipedia.org/wiki/Fault_tree_analysis

[9] "SAFETY ISSUES OF COMPUTER FAILURE", Dr. Sami M. Halawani, 30th March 2005.

[10] Jones, James V., *Integrated Logistics Support Handbook*, McGraw-Hill Professional, 3rd edition (June 8, 2006).

[11] "SYSTEMSPECIFICATION", <http://www.cs.umd.edu/~mvz/cmssc435-s09/pdf/slides6.pdf>.

[12]. John C. Knight, "Safety Critical Systems: Challenges and Directions", Proceedings of the 24th International Conference on Software Engineering Orlando, Florida, 2002, pp. 547 – 550

[13] N.G.Leveson and C.S.Turner.An investigation of the Therac-25 accidents. IEEE Computer, 26(7):18-41, March 1987

[14] William R. Dunn, "Practical Design of Safety Critical Computer Systems", Reliability Press, 2002.

[15] IEEE STD 1228-1994, IEEE Standard for Software Safety Plans, 1994.

[16] P.Seetharamaiah and M.Ben Swarup "Towards a methodology for building safe software based systems", Proceedings of the CONQUEST 2008, 11th International Conference on Quality Engineering in Software Technology, Potsdam 2008

[17] Firesmith, D.G., 2005. Engineering safety-related requirements for software-intensive systems. Proceeding of the 27th International Conference on Software Engineering, May 15-21, St. Louis, Missouri, USA., pp: 720-721. <http://portal.acm.org/citation.cfm?id=1062455>. 106 2635

[18] Anderson, P., 2008. Detecting bugs in safety critical code. Dr. Dobbs J., February. <http://www.ddj.com/development-tools/206104422>

[19]. P.Seetharamaiah and M.Ben Swarup "Towards a methodology for building safe software based systems", Proceedings of the CONQUEST 2008, 11th International Conference on Quality Engineering in Software Technology, Potsdam 2008.

IJERT