A Methodology for Modeling Software Safety in Safety-Critical Computing Systems

S. Phani Kumar1, Dr.P.Seetha Ramaiah2, Dr.V.Khanaa3
1. Department of CSE, Bharath University, Chennai, India.
2. Department of CS&SE, Andhra University college of Engineering, Visakhapatnam, India
3. Department of IT, Bharath University, Chennai, India.

Abstract
The safety aspects of computer-based systems as increasingly important as the use of software escalates because of its convenience and flexibility. Incorrect requirements have been identified as a major cause of software accidents and it appears that current software safety standards do not place a proportionate emphasis upon this causal factor. This paper 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. In this paper, a Methodology is proposed for modeling software safety based on the current software safety standards, their merits and limitations. The tasks in this proposed methodology pertains to System and software hazard analyses, Identification of software safety-critical requirements, safety-constraints based design, software safety implementation and software safety critical testing. This methodology was applied to a laboratory prototype safety-critical Railroad Crossing Control System (RCCS). The results showed that all safety critical operations are safe and risk free and capable of handling the contingency situations.

Key words:
Index Terms: Software Safety – Safety Critical Systems – Safer Software Development – Railroad Crossing Control System(RCCS)

1. Introduction

Safety-critical systems are those systems whose failure could result in loss of life, significant property damage, or damage to the environment [1]. There are many well known examples in application areas such as medical devices, aircraft flight control, weapons, and nuclear systems.

A safety critical system is a system where human safety is dependent upon the correct operation of the system. The emphasis of this paper is on the software element of safety critical systems, which for convenience is often referred to as safety critical software. However, safety must always be considered with respect to the whole system, including software, computer hardware, other electronic and electrical hardware, mechanical hardware, and operators or users, not just the software element. Safety critical software has been traditionally associated with embedded control systems. Many safety-critical systems rely on software to achieve their purposes. The number of such systems increases as additional capabilities are realized in software. Miniaturization and processing improvements have enabled the spread of safety critical systems from nuclear and defense applications to domains as diverse as implantable medical devices, traffic control, smart vehicles, and interactive virtual environments. Future technological advances and consumer markets can be expected to produce more safety-critical applications. With the recent increase in computer controlled critical systems, a clear understanding of the software development process is essential to produce quality software that eliminates software errors that can potentially result in death, injury, loss of equipment or property, or environmental harm.

1.1 Terminology

To begin, for the purposes of this paper, we define the terms **safe**, and **safety** according to definitions found in the literature. The Institute of Electrical and Electronics Engineers (IEEE) defines safe as:

Definition 1: **Safe** is having acceptable risk of the occurrence of a hazard [12]

Definition 2**: Risk** is the combination of the probability of an abnormal event or failure, and the consequence(s) of that event or failure to a system's components, operators, users, or environment [12]

Definition 3: **Hazard** is (a) an intrinsic property or condition that has the potential to cause harm or damage, or (b) an existing or potential condition that can result in a mishap [12].

Definition 4: **Mishap** is an unplanned event or series of events resulting in death, injury, occupational illness, or damage to or loss equipment or property, or damage to the environment [12].

Definition 5: **Safety** is the freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property [4].

1.2 Software Induced Failures in Real-life

Computers are increasingly being introduced into safety critical systems and, as a consequence, have been

Manuscript received July 5, 2009
Manuscript revised July 20, 2009

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 safety aspects of computer based systems.
Section 3 presents Existing software safety documents and Standards.
Section 4 presents Proposed 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 proposed and
the final section concludes the discussion, and explores directions for future research work.

2. 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 [15] :
– 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
– Effector
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 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.

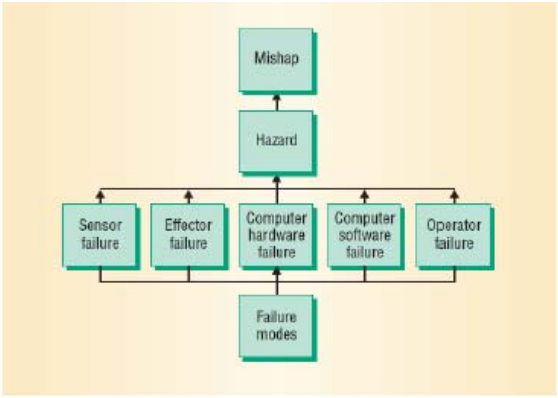

Figure 1. Mishap causes. System designers identify the application's attendant hazards to determine how system-component failures can result in mishaps.

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

2.1 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 safety-critical, 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

2.2 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.

3. Existing software safety documents and Standards

A number of software safety standards and guidelines documents and methods from various organizations for various disciplines exist today. This section provides a brief overview of these standards, guidelines and methods.

National Aeronautics and Space Administration (NASA) : NASA-STD-8719.13A provides the requirements to implement a systematic approach to software safety as an integral part of the overall system safety program[2]. This standard can be applied to software whose failure may cause an hazard and to the software which detects and corrects if the system reaches a specific hazardous state. Safety critical software is identified at system and subsystem levels by analyzing for safety at these levels. The level of system safety effort is determined by its system category and hazard severity level. The NASA Guidebook, NASA –GB-1740.13-96, provides more details of applying this standard [3].

U.S. Department of Defense: MIL-STD-882C is primarily intended for System Safety , so a detailed software safety process is not addressed[4]. However It provides a software hazard risk assessment process and considers the potential hazard severity and degree of control that software exercises over hardware. It does not provide guidance or recommendations on the tasks and levels of analysis to perform for the determined software criticality.

DO-178B – Development of Safety – Related Software in Airborne industries
The purpose of DO-178B is to provide guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements [5]. This standard provides a good description of software development tasks and links the system safety assessment process with the software development process. No specific safety tasks are detailed.

Joint Software System Safety Committee (JSSC): The JSSC Software System Safety Handbook, A Technical and Managerial team approach, provides management and engineering guidelines to achieve a reasonable level of assurance that software will execute within the system context with an acceptable level of safety risk [6]. It gives a software safety process that includes identifying generic and system safety- critical software requirements, performing software safety analysis during each stage of the software lifecycle, verifying that whether software is developed conforming to the standards and developing a software safety assessment. No specific guidance is provided on determining the level of software safety effort required.

International Electrotechnical Commission (IEC) IEC61508 - Development of Safety-Related Systems On Ground : IEC 61508 [7] is intended to enable the development of programmable electronic safety related systems where application sector international standards may not exist, and to facilitate the development of application sector international standards. IEC 61508 defines requirements for the activities to be performed throughout the lifecycle in a similar way as DO178B does. In addition, for each life cycle phase it gives a set of techniques and measures that can be applied depending on the safety integrity level(SIL).

Motor Industry Software Reliability Association (MISRA): MISRA compiles eight detailed reports containing information on specific issues relating to automotive software. The reports are summarized in a single document: Development Guidelines for Vehicle Based Software [8]. It gives software life cycle but does not provide an explicit process for software safety that could be directly implemented.

APT Research, Inc.: APT's 15 Step Process for Definition and Verification of Critical Safety Functions in Software was presented at the 2001

International System Safety Conference [9]. The steps include identifying the system hazards, identifying software safety functional requirements, and tailoring the safety effort to criticality. The method shows the integration of the 15 step process for software system safety into the system safety process and the software lifecycle.

4. Proposed Methodology for modeling software safety in safety-critical computing 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 [10].

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. perform these system-level hazard analysis and risk assessments in a manner similar to that used for systems consisting only of hardware. 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. Note however that because of the difficulties in assigning probabilities to newly developed software, the software-specific hazard analysis does not usually include an assessment of the likelihood of a software fault. 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, but are not limited to, approaches such as the following

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. [11]

Analysis of all software requirements [16] 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; relationship logic for. interdependent limits; 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, 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[18].

b. Identify potentially unsafe states caused by input/output timing, multiple events, out-of-sequence events, failure of events, adverse environments, deadlocking, wrong events, 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 [17]. 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. Weather 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 subsections describe those software safety tasks that are accomplished throughout the life cycle.

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.

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

Crossing gates on a 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 give 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.

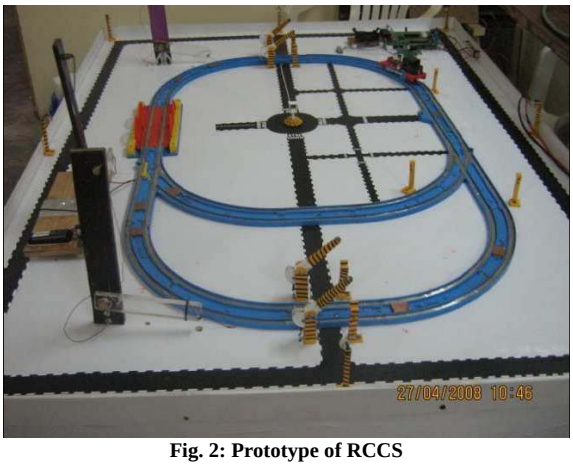

Fig. 2: Prototype of RCCS

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 motorless 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.

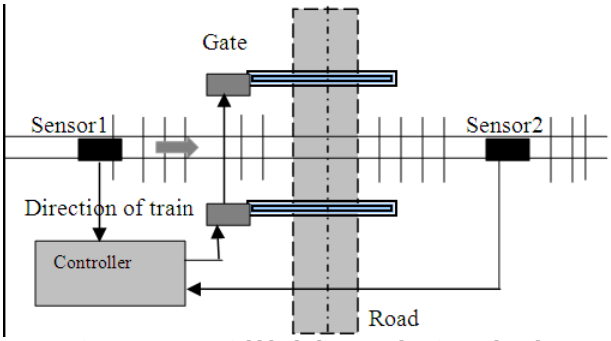

Fig. 3: RCCS partial block diagram showing railroad crossing intersection

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 a 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

This study discussed different software safety standards, their merits, limitations and problems relevant to software safety. A new methodology for software safety is proposed. A set of tasks that form the basis of software safety is presented. The proposed model is applied to a laboratory prototype of a software-based Railroad Crossing Control System (RCCS) that includes safety-critical operations and observed satisfactory results. Using the experimental results of the proposed model with railroad crossing control system, work can be extended to address issues of development cost and development time in implementing this model to achieve software safety metrics. Rigorous work is needed to meet the complete requirements of software safety aspects that leads to standardization of model with safety metrics.

References

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

[2] NASA. Software Safety: NASA Technical Standard NASA-STD-8719.13A. September 1987

[3] NASA Guidebook for Safety Critical Software NASA –GB-1740.13-96

[4] Department of Defense. System Safety Program Requirements MIL-STD-882C.1984

[5] Software considerations in airborne systems and equipment certification. DO178B 1992

[6] D. Alberico, J. Bozarth, M. Brown, et. Al. JSSC Software System Safety Handbook; A Technical and Managerial Team Approach December 1999.

[7] IEC, International Standard, Functional Safety of Electrical / Electronic / Programmable Electronic Safety- Related Systems – IEC 61508 -3; Part 3 Software Requirements. 1998

[8] MISRA. Development Guidelines for Vehicle Based Software. November 1994.

[9] H.D. Kuettner, Jr. and P.R. Owen, "Definition and Verification of Critical Safety Functions in Software", Proceedings of the International System Safety Conference (ISSC) 2001. System Safety Society, Unionville,Virginia 2001, pp.337-346.

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

[11] 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

[12] IEEE100, "The Authoritative Dictionary of IEEE Standard Terms", IEEE Press, 2000.

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

[14]] James Gleick. The New York Times Magazine 1st December 1996

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

[16] 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=1062455.1062635

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

[18] Holzmann, G.J., 2006 The power of ten: Rules for developing safety critical code. IEEE Computer, 39: 95-99. DOI: 10.1109/MC.2006.212

S Phani Kumar
Department of Computer Science & Engineering, Bharath University, Chennai, Tamilnadu – INDIA. S. Phani kumar obtained his Bachelors degree in Computer Science and Engineering from Rural Engineering College, Bhalki, which is affiliated to Visweswaraiah Technological University, Karnataka – INDIA in 2002 and his Masters degree in Software Engineering from Bharath University, Chennai, INDIA in 2006. He has presented two International Conference papers in addition to two papers at National Conferences in India He is presently pursuing the doctoral degree in Computer Science and Engineering from Bharath university, Chennai – INDIA. His research interests are Safety Critical Systems, Safety Software, Software Engineering and Real Time Systems.

Dr. Panchumarthy Seetha Ramaiah
Professor, Dept. of Computer Science and Systems Engineering, Andhra University College of Engineering, Visakhapatnam-530 003, Andhra Pradesh – INDIA. Dr. Panchumarthy Seetha Ramaiah obtained his PhD in Computer Science from Andhra university in 1990. He is presently working as a professor of Computer Science in the department of Computer Science and Systems Engineering, A.U. College of Engineering, Visakhapatnam – INDIA. He is the Principal Investigator for several Defence R&D projects and Department of Science and Technology projects of the Government of India in the areas of Embedded Systems and robotics. He has published seven journal papers, and presented Fifteen International Conference papers in addition to twenty one papers at National Conferences in India

Dr. V.Khanaa
Dean (Information Systems) Bharath University Chennai – INDIA. Dr. V.Khanaa is presently working as Dean – Information Systems in Bharath University, Chennai – INDIA and his research interests are Computer Networks, Digital Image Processing, Real Time Systems and Safety Software.