Journal of Computer Science 8 (9): 1564-1575, 2012
ISSN 1549-3636

A Methodological Framework for Software Safety in Safety Critical Computer Systems

[1]Srinivas Acharyulu, P.V. and [2]P. Seetharamaiah
[1]Department of Computer Science and Engineering,
GITAM University, Visakhapatnam, India
[2]Department of Computer Science and Systems Engineering,
Andhra University, Visakhapatnam, India

Abstract: Software safety must deal with the principles of safety management, safety engineering and software engineering for developing safety-critical computer systems, with the target of making the system safe, risk-free and fail-safe in addition to provide a clarified differentaition for assessing and evaluating the risk, with the principles of software risk management. **Problem statement:** Prevailing software quality models, standards were not subsisting in adequately addressing the software safety issues for real-time safety-critical embedded systems. At present no standard framework does exist addressing the safety management and safety engineering priniciples for the development of software safety in safety-critical computer systems. **Approach:** In this study we propose a methodological framework involving safety management practices, safety engineering practices and software development life cycle phases for the development of software safety. In this framework we make use of the safety management practices such as planning, defining priniciples, fixing responsibilities, creteria and targets, risk assessment, design for safety, formulating safety requirements and integrating skills and techniques to address safety issues early with a vision for assurance and so on. In this framework we have also analysed integration of applicability of generic industrial heirarchy and software development heirarchy, with derived cyclical review involving safety professionals generating a nodal point for software safety. **Results:** This framework is applied to safety-critical software based laboratory prototype Railroad Crossing Control System (RCCS) with a limited complexity. The results have shown that all critical operations were safe and risk free. **Conclusion:** The development of software based on the proposed framework for RCCS have shown a clarified and improved safety-critical operations of the overall system peformance.

Key words: Safety-critical systems, software safety, software quality, Rail Road Crossing Control System (RCCS)

INTRODUCTION

Software Safety is considered as most important and discussed in various software standards, specifying the needs for well being of the users, applications, equipment to avoid software failures leading to hazards by involvement of computer systems in real life. Specifically in applications of safety-critical systems, the contributions or attributions of software failures made significant danger to human life, substantial economic loss and extensive damage to environment. As no standard framework does exist which comprehensively address software safety, there is need for proper remedy and requirement of software quality and standards, or for review of the various standards and models in safety-critical computing systems. A safety-critical computer system is such a system which has the potential to cause hazards or allow hazards to occur. A software is said to be safe if it is quite not possible or a seldom instance to produce an output that could cause a catastrophic incident to the system which it controls. Most of the systems that do not have adequate safety design aspects caused loss to the physical property, harm and loss of life (Medikonda and Panchumarthy, 2009). Software Engineering of Safety-critical computer systems needs have a clarified/classified understanding of exact role of software and its interactions with the system.

Software engineering of a safety-critical system requires a clear understanding of the software's role in and

Corresponding Author: Srinivas Acharyulu, P.V., Department of Computer Science and Engineering, GITAM University, Visakhapatnam, India

interactions with, the system (Bofinger *et al*., 2002; RTCA, 1992; MISTD-882C, 1984; Lutz, 2000; Knight, 2002). These systems require the utmost care in their specification, design, implementation, operation and maintenance, as they could lead to injuries or loss of lives and in-turn result in financial loss (Herman, 2000; Schmid, 2002). This type of system is considered in this study.

According to Dunn (2003), dependable, seemingly safe concepts and structures fail while in practice due to three primary reasons. Their originators or users:

- Having an incomplete understanding of what makes a system "Safe"
- Fail to consider the larger systems into which the implemented concept is to be embedded
- Ignore single points of failure that makes the safe concept unsafe when put into practice

There are many well known examples of safety-critical systems' application areas such as automotive, defense, air traffic, air craft controlling, transportation, communications, medical diagnostics, nuclear, thermal and atomic power, instrumentation. Since, the safety is dependant on the correct and perfect desired performance of the software, this paper primarily emphasizes on the software component of safety-critical computer system, while taking into consideraion of the safety management and safety engineering issues specific to particular application system. Since scope of safety does not confine to software element only, but also to consider the safety of whole equipment, software, operators or users and environment, the constributions of safety management and safety engineering towards software safety are analyzed. most of the systems keep their reliability and confidence on software to achieve their ultimate goals. The goal of Software Safety in most of the safety-critical computer systems are real-time control systems and require most attention and care in their specification, planning, design, implementation, validation, evaluation and operational maintenance. A thorough understanding is very much required to eliminate errors in software, otherwise it may lead to or allow hazardous condition that could potentially result in catastrophic accident pertaining to life, un-sustainable injury and damage to equipment and/or environment. In this study, it is considered that such type of safety-critical computer system for application to make fail-safe. Some of the examples of hazards induced by software failures are given for reference as under. The proposed methodology in this study is basically divided into, Software Safety Management, Software Safety Engineering, Software Safety Configuration Management, Software Design and Development, Software Safety Efforts Analysis, Software Safety Testing, Software Implementation, Software Verification applying the safety practices and software developmental life cycle issues. The following are some of the concepts and terms relating to safety found in the literature on the web relevant to the safety-critical computer system.

Terminology: For the purpose of this study, the following are the definitions found available in the literature. "Software Safety Guidebook", NASA Technical Standard, 2004. http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf defines the following (NASASTD-8719.13, 2004).

Fail-Safe: (1) Ability to sustain a failure and retain the capability to safely terminate or control the operation. (2) A design feature that ensures that the system remains safe or will cause the system to revert to a state which will not cause a mishap.

Failure: The inability of a system or component to perform its required functions within specified performance requirements IEEE Standard 610.12-1990.

Error: (1) Mistake in engineering, requirement specification, or design. (2) Mistake in design, implementation or operation which could cause a failure.

Hazard: The presence of a potential risk situation caused by an unsafe act or condition. A condition or changing set of circumstances that presents a potential for adverse or harmful consequences; or the inherent characteristics of any activity, condition or circumstance which can produce adverse or harmful consequences.

Mishap: An unplanned event or series of events that results in death, injury, occupational illness, or damage to or loss of equipment, property, or damage to the environment; an accident.

Risk: (1) As it applies to safety, exposure to the chance of injury or loss. It is a function of the possible frequency of occurrence of the undesired event, of the potential severity of resulting consequences and of the uncertainties associated with the frequency and severity. (2) A measure of the severity and likelihood of an accident or mishap (3) The probability that a specific threat will exploit a particular vulnerability of the system.

Safe (Safe State): (1) The state of a system defined by having no identified hazards present and no active system processes which could lead to an identified hazard (2) A general term denoting an acceptable level of risk, relative freedom from and low probability of: personal injury; fatality; loss or damage to vehicles,

equipment or facilities; or loss or excessive degradation of the function of critical equipment.

Safety: Freedom from hazardous conditions:
Safety-Critical : Those software operations that, if not performed, performed out-of-sequence, or performed incorrectly could result in improper control functions (or lack of control functions required for proper system operation) that could directly or indirectly cause or allow a hazardous condition to exist

Safety-Critical Computer Software Component (SCCSC): Those computer software components (processes, modules, functions, values or computer program states) whose errors (inadvertent or unauthorized occurrence, failure to occur when required, occurrence out of sequence, occurrence in combination with other functions, or erroneous value) can result in a potential hazard, or loss of predictability or control of a system.

Safety-critical computing system: A computing system containing at least one Safety-Critical Function.

Safety-Critical Computing, Those computer functions in which an error can result in a potential hazard to the user, friendly forces, materiel, third parties or the environment.

Safety-critical software: A Software that (1) Exercises direct command and control over the condition or state of hardware components and, if not performed, performed out-of-sequence, or performed incorrectly could result in improper control functions (or lack of control functions required for proper system operation), which could cause a hazard or allow a hazardous condition to exist. (2) Monitors the state of hardware components; and, if not performed, performed out-of-sequence, or performed incorrectly could provide data that results in erroneous decisions by human operators or companion systems that could cause a hazard or allow a hazardous condition to exist (3) Exercises direct command and control over the condition or state of hardware components; and, if performed inadvertently, out-of-sequence, or if not performed, could, in conjunction with other human, hardware, or environmental failure, cause a hazard or allow a hazardous condition to exist.

Software safety: The application of the disciplines of system safety engineering techniques throughout the software life cycle to ensure that the software takes positive measures to enhance system safety and that errors that could reduce system safety have been eliminated or controlled to an acceptable level of risk.

System safety: Application of engineering and management principles, criteria and techniques to optimize safety and reduce risks within the constraints of operational effectiveness, time and cost throughout all phases of the system life cycle.

Background history of software failures: Computers are introduced in safety-critical systems, as a result, software failure found to have contributions to accidents. The ariane-5 explosion (Leveson and Turner, 1993), Therac-25 Accidents (Leveson and Turner, 1993) are some of the most referred software related accidents. An unmanned Ariane-5 rocket was launched by the European Space Agency, had exploded in short time after its take-off from Kourou, French Guiana in 1996. The rocket was developed after a decade of exercise and cost of $7 Billion dollars, the board of which investigated and found that software error caused failure and the loss was valued at $500 million. Therac -25, a computerized radiation therapy machine lead to six known accidents, involving excess massive doses resulting in deaths and serious injuries. This was the ever worst series of incidents of radiation accidents in the medical history (Leveson and Turner, 1993).

A software design flaw or run-time error within safety-critical functions of a system introduces the potential of a hazardous condition that could result in death, personal injury, loss of the system, or environmental damage (Leveson and Turner, 1993). Other examples, cited by (Alberico *et al*., 1999) are as follows:

- Missile Launch Timing Error Causes Hang-Fire
- Reused Software Causes Flight Controls Shut Down
- Flight Controls Fail at Supersonic Transition
- Incorrect Missile Firing Due to Invalid Setup Sequence
- Operator Choice of Weapon Release Over-Ridden by Software Control

MATERIALS AND METHODS

Overview of safety standards: Table 1 shows camparative analysis of various standards for purview and software safety scope.

The structure of CMMI +SAFE: There are two safety process areas namely, safety management and safety engineering, associated with goals as shown in the Table 2. This is intended primarily as a risk management tool. The measures can be taken to address those strengths and weaknesses identified, such as development or improvement.

The structure of the safety extension is shown in Table 2 and was developed from the structure of the safety model presented in Australian Defence Standard, The Procurement of Computer Based Safety-critical Systems and the structure of CMMI (+SAFE, V1.2, 2007).

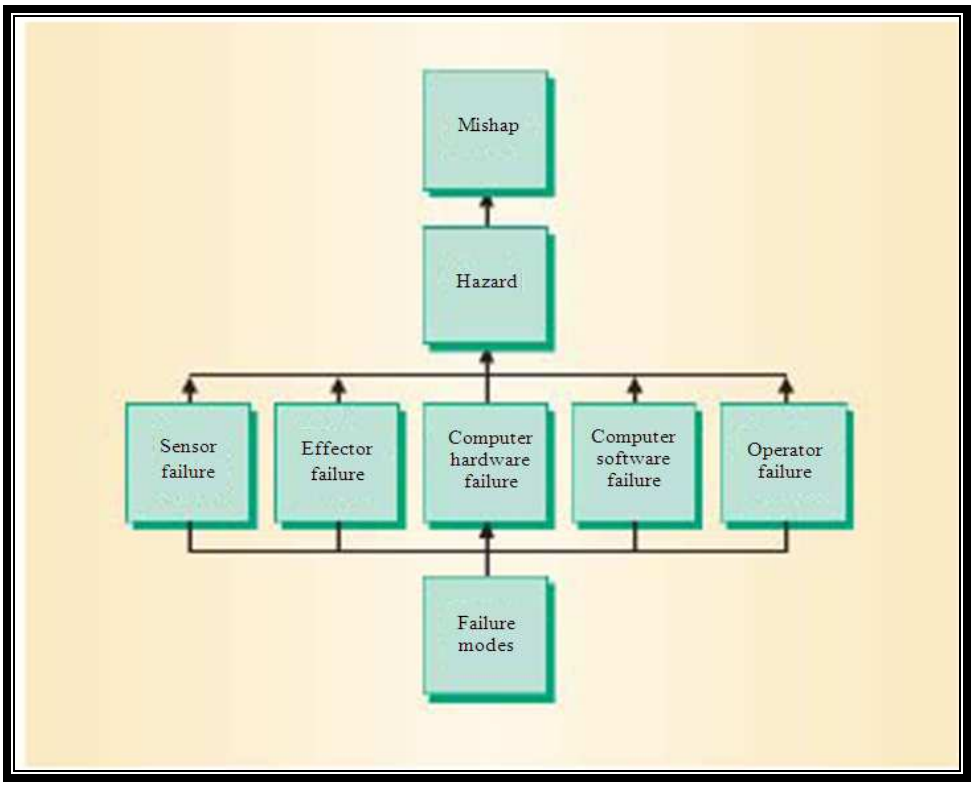

Fig. 1: Mishap causes. System designers identify the application's attendant hazards to determine how system-component failures can result in mishaps (Figure adopted from Dunn, 2003)

Table 1 : A comparative analysis of various standards for software safety scope

Standard description	Purview	Scope for software safety
DO-178B Development of Safety- Related Software in Airborne Industries	Provide guidelines for software for airborne systems	Good for Software Development and System Safety Assessment. No specific software safety tasks are mentioned.
JSSC Joint Software System Committee-JSSC Software System Safety handbook	Gives a software safety process that includes identifying generic and system safety critical software requirements, performing software safety analysis. Verifies whether software is developed according to standards and compliance.	No specific guidance is provided on determining the level of software safety effort required
MIL-STD- 882C U.S. Department of Defense	Intended for System Safety. Provide software hazard risk assessment process	Detailed Software Safety process is not given
NASA-STD-8719.13A National Aeronautics and Space Administration (NASA)	Provide systematic approach to software safety	The standard is applicable to safety critical computer systems.
+SAFE V1.2 A Safety Extension to CMMI-Dev 1.2 (+SAFE, V1.2, 2007) CMMI (Bofinger 2002)	Offer capability maturity model integrated to software	Addresses strengths and weaknesses of software

Table 2: Structure of safety extension

+SAFE – CMMI [SM] Category	Safety Process Areas	Specific Goal
Project Management	Safety Management	1. Develop Safety plans 2. Monitor Safety Incidents 3. Manage Safety Related Suppliers
Engineering	Safety Engineering	1. Identify Hazards, Accidents and Sources of Hazards 2. Analyze Hazards and Perform risk Assessment 3. Develop Safety Requirements 4. Apply Safety Principles and Requirements 5. Support Safety Acceptance

The computer based mishaps and systems: According to WR Dunn (2003), 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:

- The Application is physical entity the system controls/monitors, e.g., plant, process
- The Sensor which converts application's measured properties to appropriate computer input signals, e.g., accelerometer, transducer
- An effecter which converts electrical signal from computer's output to a corresponding physical action that controls function, e.g., motor, valve, break and pump
- Operator is human or humans 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. Any of the above five components may fail and cause a mishap as shown in Fig. 1. The main focus in this study is on Computer Software pertaining to Safety-Critical Software

Safety management process area is helpful in correcting the performance against plan. Each of the Specific Goals include specific practices, such as determining Regulatory, legal and Standards requirements, determine safety criteria, establish organization structure, establish safety plan.

Safety Engineering Process area deals with the activities of safety issues at all stages in the engineering process. The specific goals indicated include specific practices, like identifying hazards, accidents and their sources and their possibility. Specific Goal analyze hazards, is useful in determining possible causes and consequences, their severity and likelihood, help in assessment.

Software is safety-critical if it performs any of the following (NASASTD-8719.13, 2004):

- Controls hazardous or safety-critical hardware or software
- Monitors safety-critical hardware or software as part of a hazard control
- Provides information upon which a safety-related decision is made
- Performs analysis that impacts automatic or manual hazardous operations
- Verifies hardware or software hazard controls
- Can prevent safety-critical hardware or software from functioning properly

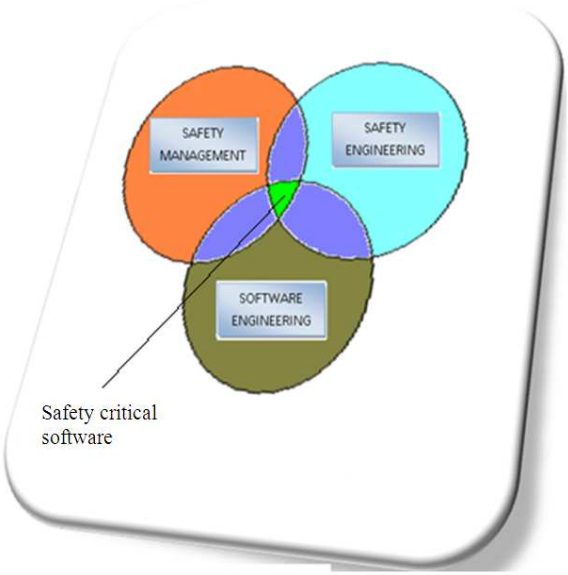

Fig. 2: Safety Critical Software shown as Cut-set

Interactions of safety critical computing systems: A Safety Critical Software is a composite of basically three areas, namely, Software Engineering, Safety Engineering and Safety Management, their interaction are shown in the Fig. 2.

The diagram shown above shows cut-set among these areas, the green colored field is the safety critical software. These interactions are described below:

- Safety Management process area, include the responsibilities of applying the defined safety principles, criteria and sets safety targets to achieve and establish safety planning to meet specified requirements, their implementation and assessment, while monitoring the safety incidents. This is a continuous process throughout and is intended to address early in the development phase. The prime goal is to develop safety plans, monitor safety incidents and coordinating with software engineering and safety engineering areas for safety critical computer systems development
- Safety Engineering process area deal with the safety configuration management, decision analysis, process quality assurance, safety requirements development and management, providing technical solutions, validation and verification of the processes
- Software Engineering concern about software design and development in coordination with safety management and safety engineering, deals with software architecture, data management and

structures, determining the simplicity of specifications and of verification and validation tasks. And all those recommended by guidelines, such as complexity avoidance, safety, modifiable structure, traceability, predictability of responsiveness, consistency and completeness, verifiability and testability and many other in accordance with software architecture analysis

- Goals common between Safety Management and Safety Engineering, are all those parameters such as decision analysis, safety requirements development and management, providing technical solution, verification
- The tasks common to Safety Engineering and Software Engineering are configuration management, process quality assurance, validation
- Common tasks between Software Engineering and Safety Management are the activities of training, project planning, monitoring and control, risk assessment and analysis, development of risk mitigating measures, providing alternate systems and development of safety case

Organization and software team structure: The Safety, Environment Management Group (EMG) exist in any industrial organization, but are not integrated. At the bottom line, executives develop software for management information system. The software development mainly considers the development of software for processing raw data into information. Unless integration of safety into the organization structure, development of software for safety critical systems, would be difficult, assurance become questionable. The integration phase of safety group, need to be considered foremost, in order to classify and focus on the issues of safety with reference to development of software for safety critical systems. The following Fig. 3 shows independent domains of an organization and software development organization in V model towards development of software for an organization. But complete information of the other is lacking on both sides particularly for safety aspects of software such as organization does not have complete knowledge of software and software developing team does not have complete knowledge of the safety and infrastructure of the other.

In this context and in order to bring a common of objective and understanding of the software requirement with safety for development of safety critical computing system, with complete knowledge of both sides, a group to act in between which need to compile the requirements from both side and act in co-ordination between them.

Upon integration a new structure with software and safety divisions with integration, primarily makes the documentation of requirements from safety, environmental, quality, software, configuration, process feedback, after preliminary hazard analysis, by verification and validation, with complete knowledge for designing software development and improvement phases.

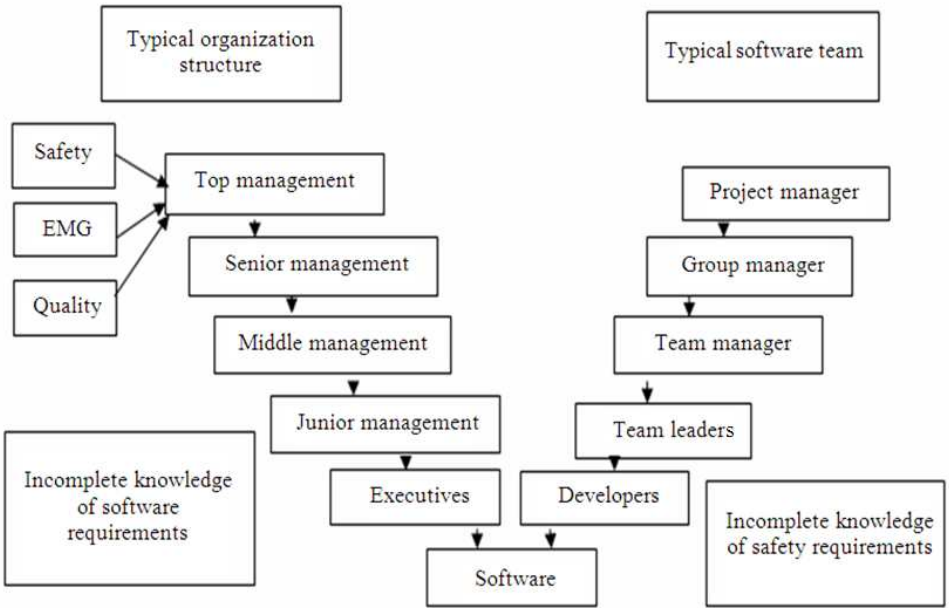

Fig. 3: Organization and software teams in V model towards software safety

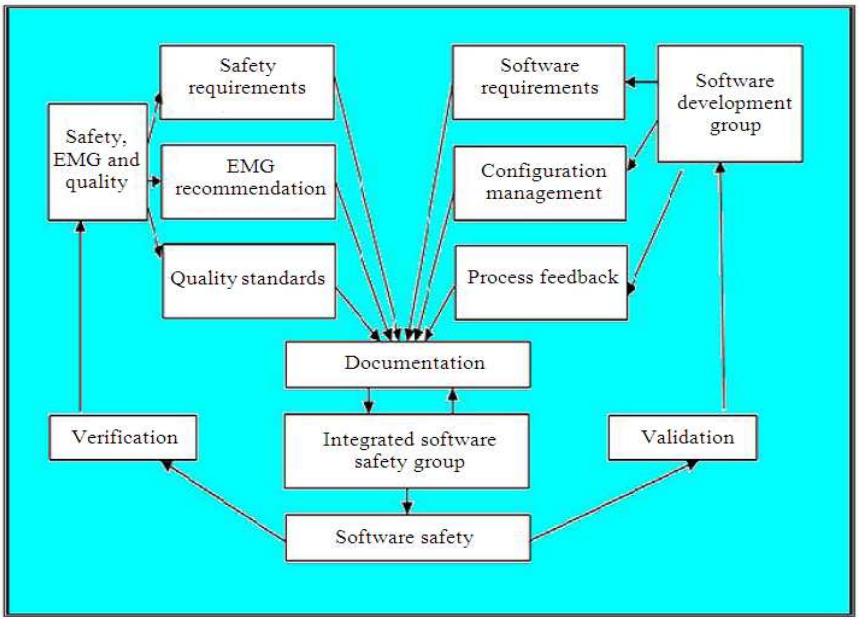

Fig. 4: Integrated software safety life cycle

Integrating safety and software towards software safety: The entire software development is oriented around ISSG, which has documentation as the main base, comprising the validated software requirements, configuration requirements, process feedback from software development group, safety requirements, standards, recommendations, quality standards from the organization. The advantage is being the availability of the complete requirements, which helps in developing software with safety. The software development with safety management and safety engineering processes functions are closely be taken into consideration.

With the above, the completeness of requirements criteria gets fulfilled, in the initial stage, as well helps in improvement of the process, as shown in the Fig. 4, from firsts trial run of the software test till retirement and provide the executing data and results for documentation and subsequent execution.

Generic safety life cycle processes: The generic safety life cycle process comprise, identification, planning, critical system identification, risks and hazards analysis for the subject system, formulate safety requirements pertaining to particular system or component, identifying significant risks and hazard, assessments, impact, evaluation, designing control procedures for avoidance or mitigation, documentation, standards compliance, else to review from the appropriate phase until complied, along with periodical reviews and audits for assessment or revision. The entire process is diagrammatically shown in the Fig. 5.

This generic safety life cycle deals only with the machinery and system as a whole. But required emphasis is not focused for the software involvement in operating the machinery or equipment. The part of safety instrumentation and system control depends on the software development with safety concerns for isolation, avoidance, reduction, use of alternate safety devices, when abnormal functioning of such system happens, out of the control, for safety critical computer system.

In the generic safety life cycle, the detailed requirements, associated risks and hazards, mitigation measures are thoroughly documented forms a complete understanding, guidance for effective safetymanagement and safety engineering. The entire process is shown in the Fig. 5.

Software development life cycle: The software development life cycle comprises, software process identification, planning, system analysis, feasibility, system design, software code design and development, code testing, verification and validation, implementation and compliance with the requirements, else refers to appropriate phase for review in the initial and continuous stages and periodical reviews and audits whenever required while looping back to any phase.

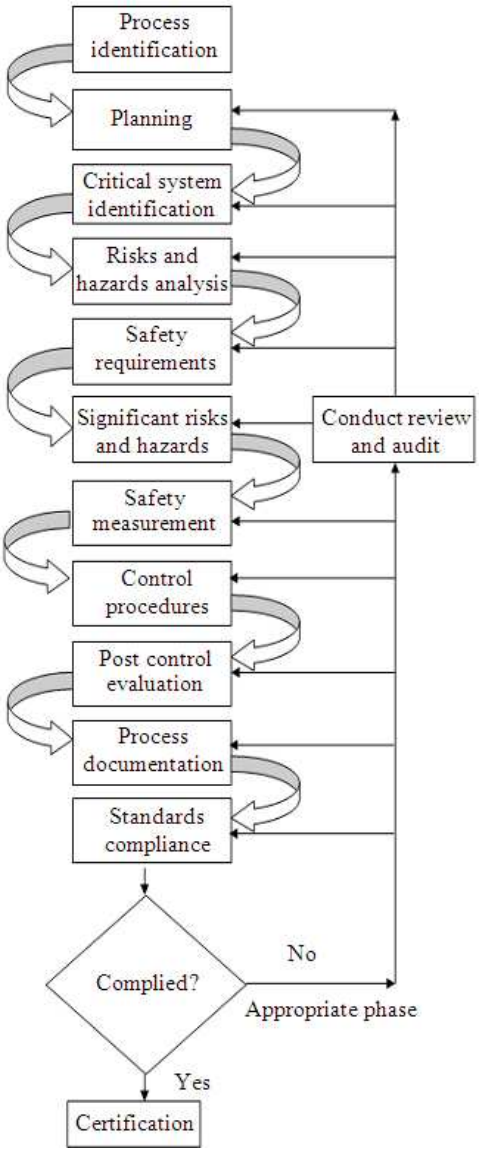

Fig. 5: Generic safety process structure

But configuration of versions released with modified design and development to suit to the current requirements is missing, with respect to Commercially off The Shelf software (COTS) unless they are notified for revision. The generic software development life cycle is shown in Fig. 6.

When software is considered for safety critical systems, the aspects of safety are untouched, or if, limited to minimum. Detailed safety analysis of software in a categorical phase development is necessary, to classify functions into safety management, safety engineering and software engineering, design and analysis, while amplifying the various software safety issues during the entire development.

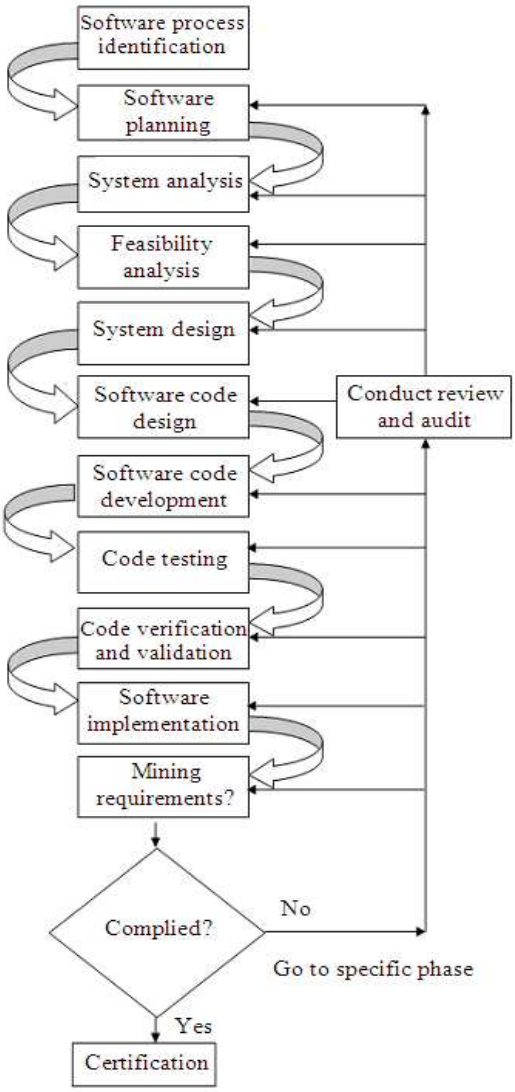

Fig. 6: Software development process

Proposed methodological framework for modeling software safety: A new methodology is proposed as follows containing modular functional based design and tasks, each consists of various sub processes as narrated below in each task and is diagrammatically shown in the Fig. 7:

- Software safety management
- Software safety engineering
- Software safety configuration management
- Software safety design and development
- Software safety efforts analysisSoftware safety testing
- Software safety implementation
- Software safety verification
- Software safety case design, development and analysis

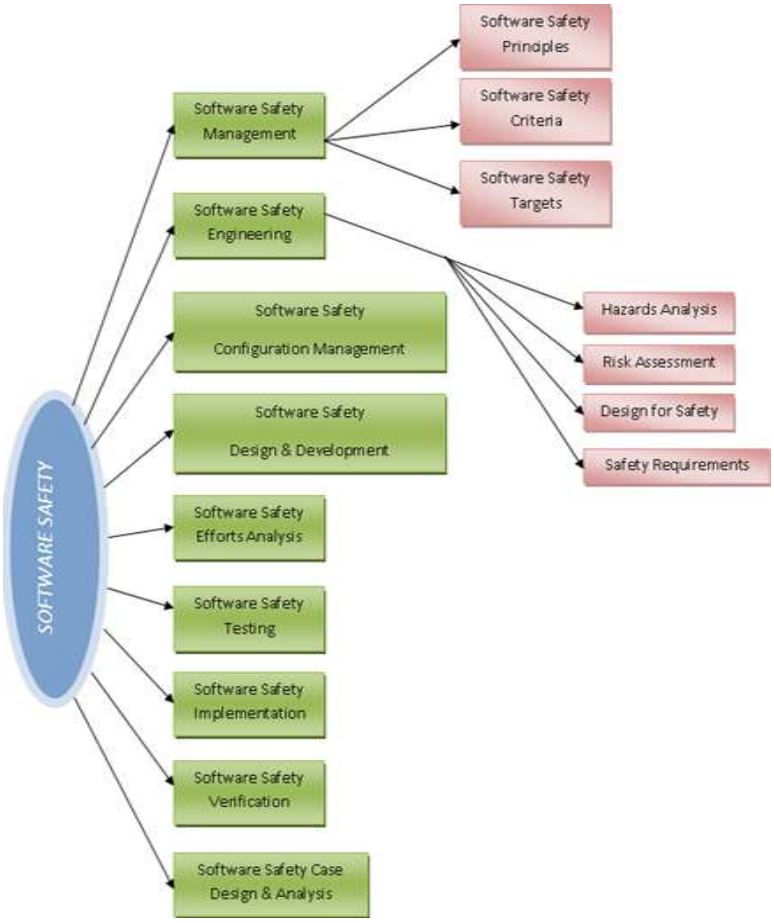

Fig. 7: Software safety creteria, sub-creteria

Software safety management: The scope covers planning, hardware may be analogous or digital, software or computer operator or use of the other involved technologies, software safety principles, criteria and target, in overview of the entire system, fixing the responsibilities, roles of the performing engineering professionals, their competence, safety activity schedules including those reviews, evaluation, installation, maintenance and software retirement.

Software safety engineering: The purview include hazards analysis, risk assessment, design for safety, formulating safety requirements, can be categorized into management principles, technical and practices, in accordance with Software Safety Management, the areas of scope of demonstrated safety critical system, addressing safety issues early, visualizing the assurance, integrating skills and techniques.

Software safety configuration management: This phase include configuration plan, establishment and subsequent management, recording of engineering changes such as modification and amendments, to provide assurance such as static and dynamic analysis results, detailed verification and validation plans.

Software safety design and development: This phase include those functions, such as software safety planning comprising of scope, software safety requirements analysis, architecture analysis, detailed design, implementation, testing and integration.

Software safety efforts analysis: This phase consisting of scoping the software safety effort, to determine system risk index and inherited software risk, determining the volume & complexity of the software with hazardous aspects of the system and how to meet or overcome the various levels. And tailor the effort to meet requirements by the degree of controls, complexity and timing criticality.

Software safety testing: The phase is intended to focus safety testing by identifying the program weaknesses

during abnormal or unexpected conditions that may cause or tend to cause software failures, beyond the boundary of operational performance limitations so that these can be removed.

Software safety implementation: Is that, the safety requirements that passed downstream to coding level for effective and efficient software controls of safety hazards to materialize or realized. The errors that are exclusive for safety-related safety-critical parts of software code, but other parts of software code which comply with the safety control, need to be considered equally. Implementation of the developed software after incorporating safety features, unit wise segregated safety-critical test.

Software safety verification: Software safety verification phase should provide the correctness arguments which demonstrate about how the component level and system level safety requirements are satisfied. The verification phase is applicable to required phases to the entire development phases wherever deemed fit. No single technique is enough for providing complete assurance, combination of techniques need to be applied. Moreover, manual inspections, walk-through and audit may be carried out.

Software safety case design, development and analysis: A software safety case to be designed and analyzed consisting of various phases, particular to a safety-critical system and safety-critical software. The case is iterative until acceptance is achieved, should go through hazards analysis, architecture and design assurances.

Application to rail road crossing control system: A limited Complex Rail Road Crossing Control System (RCCS) as shown in Fig. 8, a laboratory prototype, a Safety Critical Computer System application that includes operations of opening and closing of the gates to allow and prevent traffic on the road while a train passes through the crossing. The operation of the gate is to close before a train arrives and allow access to pass through and resume after the departure of the train. This basic function requires a detection of approaching train, or manual operation of the crossing gates by a human operator. The Fig. 8 shows laboratory prototype of Rail Road Crossing Control System comprising of several components described as follows. The main components are train, railway track, sensors, gates, controller with digital I/O card, signals and muscle wire operated track change lever. These are briefly demonstrated as under. The partial block diagram of RCCS is shown in Fig. 9.

Fig. 8: Laboratory prototype of rail road crossing control system

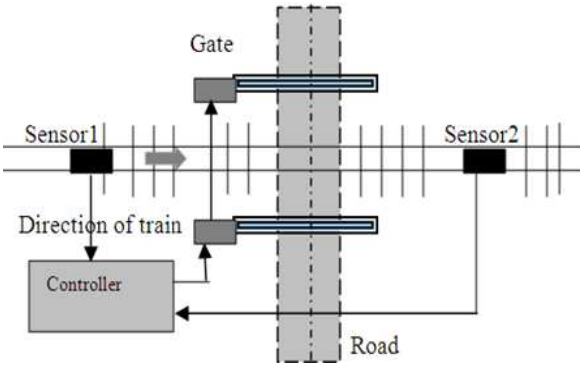

Fig. 9: Partial functional block diagram of RCCS

The Locomotive runs with the electric power, while on initial switching, the locomotive begins to move along the track or rail when the metallic wheels of the train or locomotive receive power. The train stops at the same position if the power supply to the rails or track is switched off. When the train approaches the gate crossing zone, the object train is identified or sensed by the sensor installed near the gate crossing area. The captured information by the sensor will be sent to the controller equipment. Similarly, the captured information after the departure of the train or locomotive after completely passing through another sensor installed on the other side of the gate.

The Sensors are used to identify and detect the location of the locomotive on the rails or track. Total of nine numbers of sensors are deployed in this. Two pairs of sensors are located on either side of the crossing gate, a set of three sensors are positioned (track changing purpose) at the division of track or rail one before and the other two on each track after split. Lastly another pair of sensors are commissioned with reference to the platform

to identify the position, the starting point of the train movement. The captured information from each will be transmitted to the controller.

The Controller synchronizes the train activities with the gate. When the controller receives the signal from the sensor 1, controller issues command to close the gates. And open gates command is issued when the message from sensor 2 is received. An IBM compatible PC is used as controller for RCCS. The software that controls the overall operations 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 I/O (DIO) add-on card is plugged into an available slot in the controller PC for monitoring and controlling the sensors and gate actuators. The DIO card receives the signals from all the nine sensors, the eight output signals sent from the DIO card control the power supply to the train track, power supply to the two gate assemblies, power supply to the muscle wired mechanism to change the track lever and the four signal lights.

The Gates RCCS has two sets of gates on either side of the track layout. The gate receives signal from the controller component. When a close signal is received from the controller the gate is closed and when an open signal is received, the gate will be opened. The open and close operations of the gate are carried out with muscle wired mechanism. Muscle wire is nickel titanium alloy which contract and expand according to current flow which achieves motor less motion for gate movement and track change.

The Signals for both rail and road are provided to indicate the train operators about the clearance of the track whether occupied or not, or any pre-cautionary measures need to be taken while using the track, such that to maintain reduced speed. RCCS contains three signals, erected beside the track. One is at the platform to indicate halt at the platform, the other two signals placed just before the convergence of inner and outer track which lead to platform. A signal head contains signal faces that include standard color indicators to stop, proceed with caution or proceed.

RESULTS AND DISCUSSION

Normal operations of RCCS: A primary check is being done by the controller when the system is switched on, for finding out any functional abnormalities for all systems and sub-systems involved, if not, proceed to execute the software defined for normal operations, continues to executes, with the timing schedules, else, alternative procedures are activated for any abnormal situation arises out of control due to situations beyond control like lightning, storm, signal lights begin to blink red light indicating failure, with implied suggestion to take necessary precautions. All of the points discussed above are thoroughly followed to implement, by collecting information from all concerned.

CONCLUSION

This study discussed about the various phases during the development of software for safety critical systems. Two safety process areas namely, safety management and safety engineering are applied for the development of safety critical system, Rail Road Crossing Control System (RCCS). All phases are discussed in detail, for forming the basis of software safety through and safety case development was suggested. This proposed method of integrating safety management, safety engineering and software engineering, their interactions among were discussed and is applied to laboratory prototype of four road junction traffic control system, as that includes safety critical operations and observed meaningful, improved results and found safer. Further work is in progress to apply this integrated approach to software based safety critical systems in other areas of industrial applications like power generating stations. This can be further extended to address the issues in the developmental costs and time in implementation for software safety. Rigorous work is needed to meet the complete set of software safety requirements leading to the standardization of the framework.

REFERENCES

Alberico, D., J. Bozarth and M. Brown, December 1999. JSSC Software system safety handbook: A technical and managerial team approach.

Bofinger, M., N. Robinson and P. Lindsay, 2002. Experience with extending CMMI[SM] for safety related applications. University of Queensland, Australia.

Dunn, W.R., 2003. Designing safety-critical computer systems. IEEE Comput. Soc., 36: 40-46. DOI: 10.1109/MC.2003.1244533

Herman, D.S., 2000. Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors. Wiley-IEEE Computer Society Press, ISBN-10: 978-0-7695-0299-1, pp: 520.

Knight, J.C., 2002. Safety Critical Systems: Challenges and directions. Proceedings of the 24th International Conference on Software Engineering (ICSE), May 25-25, IEEE Xplore Press, Orlando, FL, USA, pp: 547-550.

Leveson, N.G. and C.S. Turner, 1993. An investigation of the Therac-25 accidents. IEEE Comput., 26: 18-41. DOI: 10.1109/MC.1993.274940

Lutz, R.R., 2000. Software Engineering for Safety: A roadmap. Proceedings of the Conference on The Future of Software Engineering, Jun. 04-11, Limerick, Ireland, pp: 213-226. DOI: 10.1145/336512.336556

Medikonda, B.S. and S.R. Panchumarthy, 2009. A framework for software safety in safety-critical systems. 34: 1-9. ACM SIGSOFT Software Eng. Notes, DOI: 10.1145/1507195.1507207

MISTD-882C, 1984. System Safety Program Requirements.Department of Defense.

NASASTD-8719.13, 2004. NASA software safety guidebook: NASA technical standard. Department of Defense.

RTCA, 1992. Software considerations in airborne systems and equipment certification. ARINC, Annapolis, Maryland.

+SAFE, V1.2, 2007. A safety extension to CMMI-DEV, V1.2. Defence materiel Organisation, Australian. Department of Defence.

Schmid, D.C., 2002. Middleware for real-time and embedded systems. Comm. ACM-Adaptive Middleware, 45: 43-48. DOI: 10.1145/508448.508472