Journal of Computer Science

Original Research Paper

A GQM Based Approach towards the Development of Metrics for Software Safety

[1]Kotti Jayasri and [2]Panchumarthy Seetharamaiah

[1]Department of Computer Science and Engineering, GMR Institute of Technology, Rajam, India
[2]Department of CS and SE, Andhra University, Visakhapatnam, India

Article history
Received: 10-01-2015
Revised: 15-03-2015
Accepted: 29-06-2015

Corresponding Author:
Kotti Jayasri
Department of Computer Science and Engineering,
GMR Institute of Technology,
Rajam, AP, India
E-mail: jayasrikotti@gmail.com

Abstract: Software sometimes safety-critical if it resides in a safety-critical computer systems and it causes or contributes to hazards. Therefore, Safety-critical software intensive systems require verification and validation to confirm that they function as per the safety requirements. Software Safety is a combination of many factors. Metrics are commonly used in engineering as measures for the performance of a system on a given attribute. This paper presents a methodology for software safety framework based on Goal-Question-Metric (GQM) Approach. The proposed methodology was applied to a safety-critical Railroad Crossing Control System (RCCS) which is a laboratory prototype. The outcomes of the prototype are satisfactory and observed that safety risks are within the acceptable threshold level.

Keywords: Safety Metrics, Safety Metrics Framework, Hazard Analysis, GQM and RCCS

Introduction

Software has become a dominant part of a promptly growing range of applications and products from all sectors. Systems, in which software interacts with other systems, sensors, devices and with people are called software intensive systems (Navy *et al*., 1999). Software is often used to implement the functionality of safety systems because it is supposed to be design and handle complex functionality. Critical systems are broadly categorized into three categories. They are Safety-critical computer systems, Mission critical systems and Business critical systems. The failure of safety-critical system may cause injury or death to human beings. The failure of mission critical system may result in the failure of some goal-directed activity. The failure of business critical system may result in the failure of the business.

A safety-critical computer system is a system where human safety is reliant on the correct operation of the system. However, the tools and methods used for risk mitigation and risk management are lacking. Software is safety-critical, if it resides in a safety-critical computer system and if it applies at least one of the following:

- Contributes or causes to a hazard
- Controls safety-critical functions
- Practices safety-critical commands
- Mitigates damage if a hazard occurs

Criteria for Software Development Life Cycle (SDLC) are of two types, one is for non-critical software and another one is for safety-critical software. Non-critical software development follows general steps, which are included in Software Development Life Cycle. Due to complexity of the safety-critical software, there are more factors prompting in the development of safety-critical software development (Swarup and Seetharamaiah, 2009). The following are the some of the steps in critical software development process to meet the safety objectives:

- All safety functional and integrity requirements are to be clearly identified before commencing the software design phase. Because significant software modifications can be a major cause of systematic error
- The availability of safety assurance evidence is to be confirmed while considering integrating previously developed software components, in order to minimize the cost and project risk
- The numbers of personnel developing software systems are to be minimized and it is to be ensured that all interfaces are well defined
- The competence of generic safety assurance evidence is to be considered for commercial off-the shelf components in the environment of the safety system

Science Publications

© 2015 Kotti Jayasri and Panchumarthy Seetharamaiah. This open access article is distributed under a Creative Commons Attribution (CC-BY) 3.0 license.

Therefore, better metrics introduced for safety assessment of safety-critical computer systems in terms of satisfying requirements at each of the SDLC phases. One of the important safety processes is Hazard Analysis (HA). HA is the examination of a system for possible to cause harm. Therefore, HA techniques form the core of system safety methodology. System Hazard Analysis (SHA) is the analysis of interface effects and interface integration. Results of other subsystem HA are evaluated to assess the impact on other subsystems and on the total system. Interfaces are of several kinds: software to software, hardware to software and hardware to hardware and all the interfaces in a system of systems.

Hazard analysis is performed to identify the logical, code, software design and execution, testing, maintenance modules and incidence based reports. Various hazard analysis techniques and methods are applied for the purpose. HA in a less complex safety-critical system prototype is considered for case study to validate the metrics framework. The observations from this analysis are applied to framework for obtaining software safety metrics.

In the literature (Michael *et al*., 2010; Weaver *et al*., 2003; Acharyulu and Seetharamaiah, 2012), the researches on software safety include software safety analysis, safety-critical computer systems, safety metrics and validation metrics framework for software safety analysis (Basili and Weiss, 1984; Cruickshank *et al*., 2009). Because all of the researches just discuss a certain aspects of software safety metrics and issues (Basili and Rombach, 1988). It is difficult to understand the relationship among the researches. Enhancing the performance of software safety is a critical and challenging task (Knight, 2002). Large number of studies have analyzed and addressed various issues related to software safety (Software Safety, 1997; Bhansali, 2005; Axelrod, 2014; Chen *et al*., 2014). This section addresses the issues reported by some of the previous researchers (Cruickshank *et al*., 2009; IEEE, 1994).

Software safety metrics can be used to assess the maturity of hazard analysis processes and its interaction with SDLC (Kumar *et al*., 2010). Some frameworks are designed to analyze whether or not safety metric qualifies as a measure of different perspectives (Misra *et al*., 2012). Software safety measurement is a relatively unexplored area of software engineering.

GQM is a hierarchal framework for defining goals related to products and processes (Cruickshank *et al*., 2009). Agoal is inferred using a set of questions whose answers are associated with objective to safety metrics. The GQM method was adapted in this paper, which was originally developed by V. Basili and D. Weiss. This GQM method will be used throughout this paper to illustrate the various safety steps for hazard analysis.

The rest of this paper is structured as follows:

Section 2 describes proposed methodology for Software Safety. Section 3 describes application of Software Safety Framework to RCCS. Section 4 designates safety issues of RCCS laboratory prototype, the results observed after application of the methodology and the final section concludes the research work.

Proposed Methodology for Software Safety

Software Safety involves incorporating safety into the life cycle of software and analyzing the software, system and interfaces from the starting to the end. Documenting safety plans, decisions, processes, results and tracing software safety requirements through all software phases. Software safety applies to a system until it is retired. Here the authors propose new methodology with three tasks for software safety in safety-critical computing systems. The following are the three tasks:

- Software safety planning
- GQM based safety metrics framework
- Code analysis and performance monitoring

Software Safety Planning

The main advantage of software safety planning is to define the method that will aid in the preparing software that will satisfy system safety requirements. Validation comprises of the steps and processes used to answer the questions like "Are we building the right product?" etc. It means, ‘Are we building a system that meets stake holder’s requirements and expectations?’ On the other hand, verification answers the questions like "Are we building the right product?” Often validation is a last minute reactive process. Validation of software safety requirements process is shown in Fig. 1.

GQM Based Safety Metrics Framework

GQM is a hierarchal framework for defining safety metrics according to organizational objectives. GQM is a hierarchal goal-driven method, which ensures that all metrics are selected for a goal driven purpose mentioned in Fig. 2.

Framework Users

The board audience of the Framework will be the Safety engineering Team, although the hierarchal framework is applicable to other stakeholders. Information gathered from the framework will be used by the safety engineering team to identify potential weakness in the software safety process and to determine plans of action to thoroughly validate the safety requirements. However, the metric data will expose safety aspects of the system to the operators.

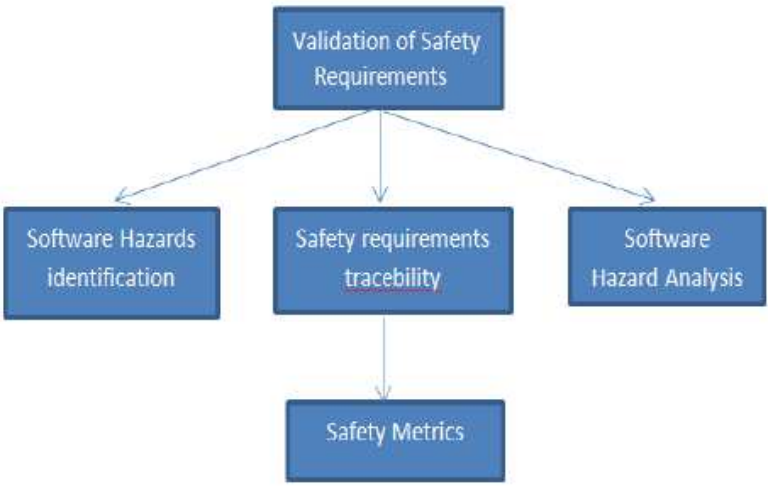

Fig. 1. Validation of software safety requirements

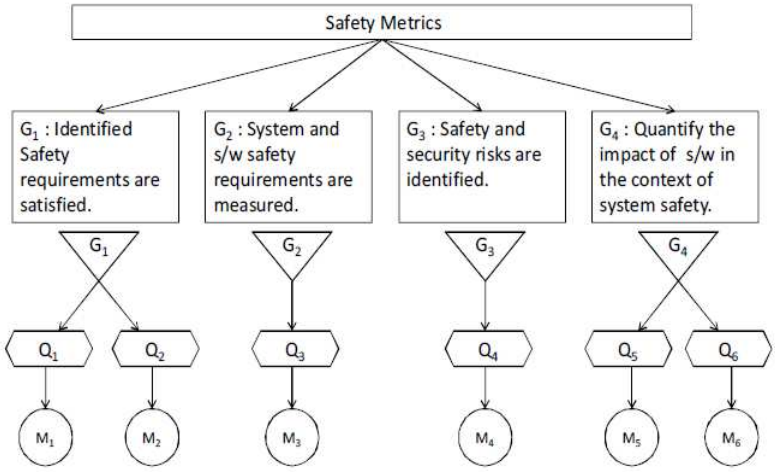

Fig. 2. GQM approach for software safety framework

Framework Goal Structure (FGS): The FGS starts with a Framework Goal (FG). The FG was identified by following the recommendations of Basili and Rombach (1988) for construction of goals in the GQM approach.

Purpose: To measure the quality of software safety process throughout the SDLC in order to aid in validating safety requirements.

Perspective: To inspect the metrics from the safety engineering team's point of view, with an attention on validating safety requirements by proxy in accordance with the proposed model.

Environment: The system has safety-critical elements that will be assured by the safety process. The FG will be used to maintain context and focus of subsequent Goals, Questions and Metrics as they are identified in a hierarchal fashion. Here four goals are identified which are common to the validation of safety requirements sufficiency of any safety-critical computer systems. Framework Goal (FG), Goals (G_1, G_2, G_3, G_4), Questions (Q_1, Q_2,……Q_n) and Metrics(M_1,M_2,…..M_n):

G_1: Identified safety requirements are adequate.

This Goal directly deals with the Hazard identification of the safety requirements and validates the identified requirements. The main purpose of this Goal is to make sure that the sufficient numbers of hazards are identified for a particular system. If the identified hazards are below a threshold boundary then it concludes that the hazard identification is not adequate, hence it leads to invalidation of safety requirements.

G_2: System and Safety requirements are measured. This goal also moderately addresses the Hazard Analysis element of the software safety requirements. The result of HA is that sufficient

requirements are identified to mitigate the software hazards. By measuring the number of safety requirements against a pre-determined model, the sufficiency of hazard analysis can be obtained

G_3: Software hazards are necessarily mitigated

This Goal directly deals with the hazard analysis of the safety requirements to validate the risk mitigation process. The main aim of this goal is to reduce the total number of High Risk hazards in the system. This will aid in increase the confidence of the developer.

G_4: Quantify the importance of software with respect to system safety

This Goal is useful to software requirements and quality assurance. This goal can be achieved by classifying causes of a hazard in the system. The following are the questions related to this goal.

Questions and Metrics of GQM Framework

The above mentioned goals can be achieved by answering a number of questions as given below. Every goal and question will be assessed by relevant metrics as shown in the framework. The relevant questions and metrics of GQM are described as follows:

Q_1: How many Software safety requirements are identified?
Q_2: How many system safety requirements are identified?
Q_3: Whether the number of safety requirements identified is sufficient or not?
Q_4: Are all the hazards are having mitigation plan?
Q5: What percentage of the hazards is software related?
Q_6: What percentage of the hazards is caused by hardware?

M1: Percentage of Hazards for Software Safety (PHSS): M1 (PHSS) is an indicator of the adequacy of hazard identification. By comparing the total number of safety hazards identified against historical data and system safety Hazards, indicate the validity of the software safety requirements through identified hazards.

$$PHSS = \frac{Total\ software\ safety\ hazards}{Total\ system\ safety\ hazarda} \times 100 \quad (1)$$

The model for inferring MI requires an Estimated PHSS (EPHSS), which is based on previously developed similar systems. If [PHSS-EPHSS] <σ, it indicates that a necessary number of Safety hazards have been identified in hazard identification process, where EPHSS is the average of the PHSS for all other similar systems and σ is the standard deviation of the PHSS.

M_2: Percentage of Hazards for System Safety: M_2 (PHSyS) is pointer of how sufficient hazard analysis has been performed and hence the validity of the derived safety requirements. It is a like in format to MI:

$$PHSyS = \frac{Total\ system\ safety\ hazards}{Total\ software\ safety\ hazarda} \times 100 \quad (2)$$

The model for PHSyS requires an Estimated PHSys (EPHSys) based on previously developed systems. If [PHSyS-EPHSyS] <σit indicates that a reasonable number of safety requirements have been identified where the EPHSyS is the average of the PHSyS for all systems in the family, (in line with other systems) and σ is the standard deviation of the PHSyS.

M_3: Percentage of Software Safety Requirements: M_3 (PSSR) is an indicator of how necessary hazard analysis process has been performed and hence the validity of the derived safety requirements:

$$M_3 = \frac{\#SH_{HR-SR}}{\#SH_{HR}} \times 100 \quad (3)$$

where, #SHHR-SR is the number of High Risk software hazard with associated software safety requirements and #SHHR is the total number of high-risk software hazards While development of the system progresses, it is expected that this safety metrics will approach and reach cent percentage.

M_4: Percentage of High Risk Software Hazards with Safety Requirements: M_4 (PSH_{HR}) is an indicator of high-risk software hazards have resulted in applicable safety requirements through hazard analysis. This indicates the sufficiency of safety requirements:

$$M_4 = \frac{Total\ no.of\ software\ safety\ requirements}{Total\ safety\ requirements} \times 100 \quad (4)$$

While development of the system progress, it is anticipated that this metric will approach and reach cent percentage.

M_5: Percentage of Failures with Software Hazards: M_5 (PFSH) is an indicator of Failures with Software Hazards.

$$M_5 = \frac{Total\ failures\ with\ software\ hazards}{Total\ system\ failures} \times 100 \quad (5)$$

M_6: Percentage of Failures with Hardware Hazards: M_6 (PFHH) is an indicator of failures with hardware hazards:

$$M_6 = \frac{Total\ no.of\ failures\ with\ hardtware\ hazards}{Total\ no.of\ system\ failures} \times 100 \quad (6)$$

Code Analysis and Performance Monitoring

The task of software safety code analysis and performance monitoring of the system begins in the software implementation and unit testing phase. Inputs into this task include the system hazard analyses outputs, safety requirements. The software safety code analysis will examine the software requirements specification and test procedures.

Application of Software Safety Framework to Railroad Crossing Control System (RCCS)

Accidents are prone to occur in the unmanned railway crossings. In order to prvent the occurrence of accidents a RCCS is proposed. In this system the approach of the train is sensed before hand and accordingly the closing and opening of the railway crossing gates is actuated. This operation is a fool-proofing system which eliminates the errors prone to manual intervention. The laboratory prototype of RCCS is shown in Fig. 3 and consists of several parts as listed below.

Components of RCCS

RCCS consists of the following main parts: Train, Railway track, Gates, Sensors, Controller with a digitalI/O card, Signals and a muscle-wire operated track change lever. Description of each part of the laboratory prototype is indicated below.

Train: The actuating power is given by the power supply relay, to the wheels of the train, which initiates the movement of the train along the track. In order to stop the movement of the train, the actuating power is cut-off. When the train approaches the gate crossing area, a sensor detects the approach of the train and sends this information to the controller component. The sensor keeps sending the signal to the controller till the train completely overtakes the gate crossing area.

Sensors: RCCS uses nine sensors in totality. The sensors perform the job such as detection of the presence of the train on the crossing area and finally sends the signal to the controller.

Controller: Controller controls the activities of lowering and raising the gates with respect to the presence and absence of the train respectively. Sensor no.1 is responsible for lowering the gates and sensor no.2 for raising the gates. This activity is done by the controller which is actuated by the signal from the sensors. AnIBM compatible PC is nominated as the controller for RCCS. The DIO card receives the inputs from each of the ninesensors 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.

Fig. 3. Prototype of RCCS

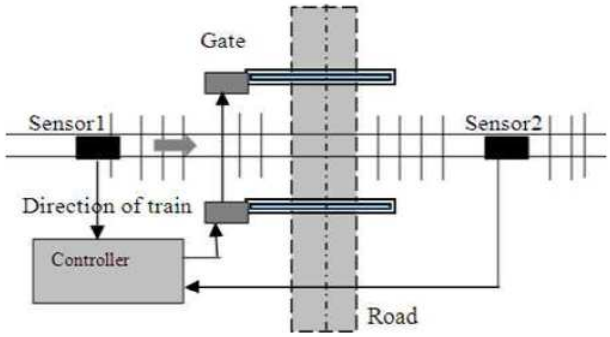

Fig. 4. Partial functional block diagram of RCCS

Gates: RCCS has two sets of gates on either side of the track layout, which is operated by a muscle wire based mechanism. Controller sends the signal to gate. When the signal value is lower, gate moves down and when the signal value is higher the gate is raised.

Signals: RCCS contains three train signals, placed beside the track. Signals give an indication to the train operators that whether the track is clear or occupied, or if certain precautionary measures were taken or not while using the track, such as maintaining a reduced speed. A signal post consists of solid red and green lights.

Experimental Results and Analysis

When RCCS is switched on, the controller preliminarily checks of the normal working status of all subsystems involved in 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. All the tasks of the methodology were applied to RCCS.

Firstly, software safety planning is used to define the system safety requirements and completeness of requirements is verified and applied some functions for safe performance (or) operation. In the Software Safety planning the system level, software hazard analysis was used 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 Gates, Failure of Signals, Failure of muscle wire operated Track Change Lever in changing from outer to inner track. All requirements that directly or indirectly which lead to incorrect operation of the gates are considered as safety-critical.

With system hazard analysis, the existing hazards will be found and fixed.

Secondly, GQM based safety metrics framework is proposed. This framework identifies a number of goals, related questions and safety metrics to achieve those goals. The Safety Metrics Framework (SMF) provides early warnings of the invalidity of Software Safety requirements. The results indicated that a sufficient number of software safety requirements are being developed and safety risks are within the acceptable threshold level. This increases the confidence that the safety requirements that are indeed valid.

Finally, run-time performance of the RCCS was monitored for problems relating to omissions (Exceptions), deadlocks, memory related issues.

The results in applying the proposed methodology in developing the safety-critical laboratory prototype RCCS clearly demonstrate that the system is risk free and fail safe when compared to a methodology which does not take hazards as well as associated risks into consideration. The goals and corresponding metrics are shown in Table 1. The metrics and the corresponding risk impact levels are shown in the Fig. 5.

Table 1. GQM framework Results for RCCS

Goals	Metrics	Evaluation	Risk impact factor
Identified safety requirements are satisfied	M1 (PHSS)	$\frac{TSSH}{TSySH}\times 100$	2.80
	M2 (PHSyS)	$\frac{TSySH}{TSSH}\times 100$	1.79
System and S/w safety requirements are measured	M3 (PSSR)	$\frac{\#SSR}{\#SR}\times 100$	3.20
Safety and Security risks are identified	M4 (PSHhr)	$\frac{SHhr}{TSHhr}\times 100$	2.90
Quantify the impact of oS/w in the context of system safety	M5 (PFSH)	$\frac{TSHF}{TSyF}\times 100$	4.20
	M6 (PFHH)	$\frac{THHF}{TSyF}\times 100$	1.80

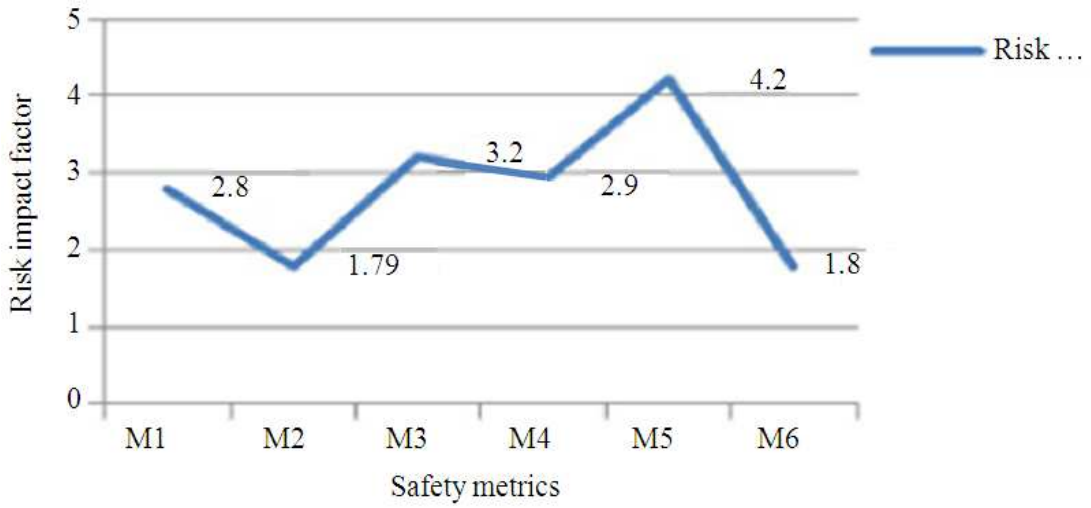

Fig. 5. RCCS metrics results

Conclusion

This paper addresses the key hazards that are required to mitigate for any safety-critical system. The paper has presented a new methodology for software safety called safety metrics framework based on GQM approach. The proposed methodology was applied on the laboratory prototype RCCS, which is a safety-critical system. The goals were evaluated based on the values given by the prototype. Six metrics were evaluated and based on these metrics we estimated the risk factor. The results of the prototype are satisfactory and indicated that safety risks are within the acceptable threshold level. The suggested methodology forms the basis of software safety. The framework provides early warnings of the invalidity of software safety requirements. The proposed methodology is applied to a laboratory RCCS prototype, which includes safety-critical operations. This RCCS prototype is tested and yielded satisfactory results. This paper also found that most of the risks are software related when compared with hardware risks. Therefore, the authors conclude that for any safety-critical system like RCCS identification and mitigation of software hazards should be given high priority.

Acknowledgment

The authors wish to thank anonymous reviewers for their valuable, detailed comments that improve both the content and representation of this study.

Author's Contributions

Kotti Jayasri: All the work belongs to me is my PhD work.

Panchumarthy Seetha Ramaiah: My supervisor he support me a lot.

Ethics

This research paper is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Axelrod, C.W., 2014. Reducing software assurance risks for security-critical and safety-critical systems. Proceedings of the IEEE Long Island Systems, Applications and Technology Conference, May 2-2, IEEE Xplore Press, Farmingdale, pp: 1-6. DOI: 10.1109/LISAT.2014.6845212

Basili, V.R. and H.D. Rombach, 1988. Towards improvement-oriented software environments. IEEE Trans. Software Eng., 14: 758-778

Basili, V.R. and D.M. Weiss, 1984. A methodology for collecting valid software engineering data. IEEE Trans. Software Eng., I0: 728-738.

Bhansali, P.V., 2005. Software safety: Current status and future direction. ACM SIGSOFT Software Eng. Notes, 30: 3-3. DOI: 10.1145/1039174.1039193

Chen, L., L. Huang, C. Li, L. Wu and W. Luo, 2014, Design and safety analysis for system architecture: A breeze/ADL-based approach. Proceedings of the IEEE 38th Annual Computer Software and Applications Conference, Jul. 21-25, IEEE Xplore Press, Vasteras, pp: 261-266. DOI: 10.1109/COMPSAC.2014.35

Cruickshank, K.J., J.B. Michael and M.T. Shing, 2009. A validation metrics framework for safety-critical software-intensive systems. Proceedings of the IEEE International Conference on System of Systems Engineering, May 30-Jun. 3, IEEE Xplore Press, Albuquerque, pp: 1-8.

IEEE, 1994. IEEE Standard for Software Safety Plans. 1st Edn., IEEE, New York, ISBN-10: 155937425X, pp: 17.

Knight, J.C., 2002. Safety critical systems: Challenges and directions. Proceedings of the 24th International Conference on Software Engineering Orlando, Florida, pp: 547-550.

Kumar, S.P, P.S. Ramaiah and V. Khanaa, 2010. A methodology for building safer software based critical computing systems. Proceedings of the IEEE 2nd International Advance Computing Conference, Feb. 19-20, IEEE Xplore Press, Patiala, pp: 422-429. DOI: 10.1109/IADCC.2010.5422901

Michael, J.B., M.T. Shing, K.J. Cruickshank and P.J. Redmond, 2010. Hazard analysis and validation metrics framework for system of systems software safety. IEEE Syst. J., 4: 186-197. DOI: 10.1109/JSYST.2010.2050159

Misra, S., I. Akman and R. Colomo-Palacios, 2012. Framework for evaluation and validation of software complexity measures. IET Software, 6: 323-334. DOI: 10.1049/iet-sen.2011.0206

Navy, U.S., U.S. Army and U.S. Air Force, 1999. Joint services computer resource management group joint software system safety committee, software system safety handbook: A technical and managerial team approach.

Software Safety, 1997. NASA technical standard. Software Safety.

Acharyulu, P.V.S. and P. Seetharamaiah, 2012. A methodological framework for software safety in safety critical computer systems. J. Comput. Sci., 8: 1564-1575. DOI: 10.3844/jcssp.2012.1564.1575

Swarup, M.B. and P. Seetharamaiah, 2009. A software safety model for safety critical applications. Int. J. Software Eng. Applications, 3: 21-32.

Weaver, R., J. Fenn and T. Kelly, 2003. A pragmatic approach to reasoning about the assurance of safety arguments. Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03), Australian, pp: 57-57.