The Research Framework for Quantitative Safety Assessment for Safety-Critical Computer Systems

Jayasri Kotti1* and Venkataramana Attada2

1Department of CSE, LENDI Institute of Engineering and Technology, Vizianagaram, Jonnada – 535005, Andhra Pradesh, India; jayasrikotti@gmail.com
2Department of CSE, GMRIT, Razam – 532127, Andhra Pradesh, India; mithun.avr@gmail.com

Abstract

Objectives: A safety-critical computer system has to be designed with safety in mind. The purpose of this paper is to identify and assess the safety hazards by using the proposed framework for Safety-Critical Computer Systems (SCCS).

Methods/Statistical Analysis: Computer software quality models like McCall’s and Boehm’s were deficient in addressing the basic safety issues of SCCS. This paper proposes another safety model for software safety by adjusting McCall’s product quality model that particularly distinguishes the conditions comparing to software safety in safety-critical applications. The conditions in the proposed software safety model relate to Hazard Analysis distinguishing of Completeness of safety requirements, Safety-critical Design, Coding, and Testing. Findings: The criteria in the proposed software safety model relate to Hazard Analysis (HA) distinguishing of safety requirements, Completeness of safety requirements, Safety-critical Design, Safety-critical Coding, and Safety-critical Testing. The proposed safety model changes the current quality model by presenting different safety criteria’s and metrics in all phases of Software life cycle to assess the software safety. This model was connected to a safety basic Railroad Crossing Control System (RCCS) which is a laboratory prototype and obtained better results in terms of safety. Application/Improvements: This model was connected to a safety basic Railroad Crossing Control System (RCCS) which is a laboratory prototype and obtained better results in terms of safety.

Keywords: Hazard Analysis (HA), Railroad Crossing Control Systems (RCCS), Safety Metrics, SCCS, Software Safety, Software Development Life Cycle (SDLC)

1. Introduction

Safety is a system property. Creating safety-critical software is turning into an undeniably complex activity. Building software for SCCS is a challenging errand. Software by itself can’t harm anybody. It works in an electronic system and frequently controls other equipment. Computer software is perilous in the event that it can specifically prompt a danger or is utilized to control a risk. A SCCS is a system which has the potential and may bring about mischance either specifically or in a roundabout way.

Failure of SCCS can bring about decrease of life, property harm, environmental harm and financial loss. Software safety is always considered to the entire system including all safety-critical operations. Examples of SCCS are Defence Weapon Delivery Systems, Space Research Applications, Maintenance and utilization of robots, fly by cable frameworks, Air Traffic Control Devices, Trains Signalling Devices, Road Traffic Control Systems, drug radiation treatment machines, Medical Radio Diagnostics, Medical Robots and so on. In the past years, many cases of system malfunction caused by software failures have raised which results in death, harm, or harm to property, or environment.

1.1 Recent Catastrophic Accidents

Loss of Communication between the FAA Air Traffic Control Center, and Airplanes: A software error in a Microsoft Device compounded by individual error was eventually accountable for the 3 hour radio breakdown.
The Research Framework for Quantitative Safety Assessment for Safety-Critical Computer Systems

Accident of Air France Flight 447: The cause was due to incorrect airspeed information that was fed to the aircraft computers which resulted in wrong calculations.

These SCCS require the most extreme consideration in their requirement, plan, execution and maintenance, as they could prompt to injuries or decrease of lives and thurstly bring about money related misfortune. These sorts of systems will be considered in this study. There are numerous techniques and strategies to enhance software safety. Software failure mode and effect analysis (SFMEA) is essential to enhance the safety of the safety-critical software. SFMEA is a successful approach to recognize and mitigate potential issues inside the design of a system. And finally, the analysis yields a quantitative assessment of safety of the software system. Here are some safety-related terms and ideas identifying with software safety found in the literature identifying with SCCS.

2. Safety-Related Terms

**Failure**: According to Leveson, an occasion where a device or subsystem segment does not display the normal performance and ecological setting under which it must be shown ought to be archived in the requirements specification.

**Mishap**: Mishap is a spontaneous occasion or arrangement of occasions bringing about death, harm or loss of hardware or property or harm to nature.

**Hazard**: According to Leveson, A computer system express that may, under definite ecological setting, lead to an incident. Hence hazard is a possibly unsafe situation.

**Safety-critical**: According to John- Those computer software operations, that if not performed, performed out of arrangement, or performed mistakenly could bring about despicable control capacities (or absence of control capacities required for appropriate system operation) that could specifically or by implication cause or permit risky condition to exist.

**Software Safety**: According to Mil-std-882 and IEEE STD 1228, The utilization of the order of system safety engineering procedures for the duration of the life cycle to guarantee that the software takes positive measures to upgrade system safety and that errors that could decrease system safety have been eliminated or controlled to a satisfactory level of risk.

2.1 Safety View Points

The role of software is increasing in industries and applications in safety-critical systems. Engineers can look at safety from dissimilar viewpoints:

Computer device safety, Hardware safety and Computer Software safety. Figure 1 clearly shows these viewpoints from a basis perspective using a Venn diagram. System prosperity is the uttermost level and conceals both device and software product safety issues i.e., device safety concentrates on the overall system’s safety. Equipment safety, as its name implies, focuses on ensuring that hardware devices are safe while software safety concentrates on making sure the software systems are safe. Not only can hardware and software factors affect system safety but also other factors such as environmental conditions and human communications. Consequently and others, system engineers must consider every single vital element while making SCCS.

![Figure 1. Safety Viewpoints.](image)

The computer device state could be because of faulty equipment, software product, or both. A programming hazard is a condition that can permit a device level risk to happen; accordingly, programming hazards are faults in the program that can lead to an accident.

At the point when creating safety-critical program, engineers may remove many faults from the device, but the question remains as to how many faults were safety-related faults.

For instance, a developer may expel maximum all faults from the software product, however in the event that the rest of the errors are safety associated faults, the device is unsafe even though reliable. For software product safety engineers, the handling of safety-related faults is the primary objective. For those systems that
are 100% safety critical any fault in the system will affect safety; therefore, any technique that improves reliability, whether a safety-specific technique or not, will improve safety. Similarly, systems that contain a large percentage of safety-critical features may also benefit from traditional, non-safety-specific techniques; however, for such systems, there is no assurance that the traditional techniques will find any safety-critical faults.

This research, therefore, deals with software systems that are not 100% safety-critical, and is to concentrate on safety-specific goals that target only hazardous software states. According to Raghu Singh every element the comparing criteria is determined. It is contended that determination and utilization of particular, design, implementation and testing techniques in a project ought to be founded on the metrics got from the criteria with a specific end goal to guarantee software safety. All existed software quality models have many restrictions. Finally none of these models suggested safety metrics related to safety critical systems. To conquer these confinements, another safety model is suggested that catches the real issues particularly identified with software safety along with safety metrics. The proposed safety model modifies the existing quality model by introducing various safety criteria's and metrics to evaluate the safety.

3. Research Framework

This paper introduces a research framework for software safety. The research framework is shown in Figure 2. The intention of this framework is to assess the safety status and also it proposes some safety metrics (M1 to M8). Safety-intensive frameworks must agreement with the hazards distinguished by safety analysis model keeping in mind the end goal to make the system fail-safe. A safety case includes an argument as to why the system is believed to be safe to deploy in its proposed operational perspective. The research framework contains two models; these are fault model and safety analysis model. A fault model captures information about the different ways in which the components of the system can malfunction. It defines the behaviour of common failure modes. The fault model also specifies the fault triggers that activate the component failures and their duration. Safety analysis model focus on safety attributes to make sure correct functionality and to detect malfunctions, failures and to develop the mitigation strategies to control hazards. By integrating the safety analysis model with a fault model we can reduce the cost and improve the quality of the safety analysis.

Test data set is given as input to the fault model and observes the outcome of fault model in terms of test response and fault coverage. The output of the fault model is redirected as input to the safety analysis model. Safety analysis model describes the safety status and various safety metrics. This process iterates till observed the expected safety status. The various safety metrics are discussed in the next section.

![Feedback](image)

### Figure 2. Research Framework.

#### 3.1 Fault Model

With a specific end goal to decide the safety measure essential to make sure correct behaviour of critical components over a system, we must first recognize the types of failures that can occur, as well as their causes. The strongest fault model supports the Failure Modes and Effect Analysis (FMEA). Frequently occurred failure modes and their possible causes are brainstormed among the Cross-Functional-Team and are shown in Table 1.

The fault model takes possible failures as Test data set and prepares the test response, fault coverage with the help of fault model by using applicability matrix. Here, some of the results are observed from the laboratory prototype RCCS. The behavioural model for the train procedure of a Railroad Crossing Control System (RCCS) in Finite State Machine (FSM) design that characterized in Figure 3 is adopted from. The model indicates that gates are to be shut and cautioning lights are to be turned on when a train approaches, and that they are to stay in the same state until the train takes off. At the point when the train leaves, the doors are opened and the lights exchanged off. Gates stay open and lights are off while no train approaches.
The model contains 4 states, S = {S0, S1, S2, S3} where the initial and final states are S0. There are 6 transitions X = {x1, x2, x3, x4, x5, x6}. Assuming edge coverage is required, the test tracks in Table 2 fulfil this requirement.

Table 2. Test Tracks through FSM example

| Test Tracks | States in FSM |
|-------------|---------------|
| T1          | S0,S1,S2,S3,S0|
| T2          | S0,S1,S2,S3,S1,S2,S3,S0|
| T3          | S0,S1,S3,S1,S2,S3,S0|
| T4          | S0,S1,S3,S0|

3.2 Failure Modes and Applicability Matrix for RCCS

Failure modes and Applicability matrix is derived from Safety Analysis Model for obtaining safety status and safety metrics. Table 3 shows four possible failures modes that could occur in the system and the applicability matrix, indicating that not all failures are applicable in all states.

Table 3. Failure Modes and Applicability Matrix

| F  | Description                                      | F/S | S0 | S1 | S2 | S3 |
|----|--------------------------------------------------|-----|----|----|----|----|
| F1 | sensor fails (to detect approaching or leaving train) | F1  | 1  | 1  | 1  | 1  |
| F2 | warning lights fail                              | F2  | 1  | 1  | 1  | 0  |
| F3 | gate stuck open                                  | F3  | 0  | 1  | 1  | 0  |
| F4 | controller fails                                 | F4  | 1  | 1  | 1  | 1  |

3.3 Safety Analysis

Safety analysis is a vital procedure to manage risks for avoidance or moderation, and requires expertise, knowledge, practice and appropriate application. Computers and software are used for such systems called safety-critical systems. Software safety analysis is an essential part of hazards analysis for distinguishing proof and evacuation of software mistakes bringing about failures of SCCS. However, the tools for risk or safety assessment, risk mitigation and risk management are not up to a satisfactory level. Hence, better safety metrics are introduced for safety evaluation of SCCS in terms of satisfying needs at each of the SDLC phases. Safety metrics are commonly used in engineering as measures of the performance of a device for a given attribute. Any failure that may occur during any stage of the software safety life cycle is called systematic failure. In order to identify the reasons for such failures, a through safety analysis is a must. The
observations from safety analysis through various safety and hazard analysis methods would reveal the factors and criteria that could contribute to hazards. Metrics can be derived after consolidation of these factors and criteria.

Safety analysis is carried out on the laboratory prototype, to identify risk associated in the various stages throughout the life cycle. Hazard analysis in a less compound SCCS model is measured for case study to validate the metrics framework. The remarks from this study are applied to framework for obtaining software safety metrics.

### 3.4 Safety Model

Safety status is characterized as a system property that is worried with failures that can bring about hazards to individuals or systems. For SCCS it is regularly necessary to perform different safety related analyses as a component of the software development lifecycle. A system performing safety-critical undertakings might be respected by sets of correlative states: safe and Unsafe. The safety model can further be explained by isolating safety-critical systems into two types, inherently dangerous and potentially dangerous. The first type comprises of systems that were designed to harm. The second type of systems was intended to do no harm.

The safety model is used for enhancing quality assurance techniques for accomplishing safety objectives in computer controlled systems. The proposed model locations safety metrics in every phase of SDLC for any SCCS. The Safety Model as shown in figure 4 and it is mainly focused on safety metrics (M1 to M8) described below. The proposed model contains three major safety issues. These are safety factors, safety criteria and safety metrics (S-FCM).

**M1:** Percentage of Safety Requirements Traceable (PSRT) is an indicator of the adequacy of safety requirements recognizable proof. i.e., no. of Safety Requirements Traced divided by Total number of Safety Requirements.

\[
\text{PSRT} = \left(\frac{\text{NSRT}}{\text{TNSR}}\right) \times 100
\]

**M2:** Percentage of Incomplete requirements (PIR) is an indicator of the incomplete requirements. i.e., Incomplete Requirements divided by total no. of safety requirements.

\[
\text{PIR} = \left(\frac{\text{NIR}}{\text{TSR}}\right) \times 100
\]

**M3:** Percentage of Hazards Analyzed (PHA) is an indicator of the hazards identified and analyzed i.e., \(m_1 = \frac{HA}{HI}\) and \(m_2 = \frac{HE}{AH}\). Here \(m_1\) hazards analyzed and \(m_2\) is hazards eliminated.

\[
\text{PHA} = \text{Average} \left( m_1 \text{ and } m_2 \right)
\]

**M4:** Percentage of Hazards Eliminated (PHE) is a metric for find the number of eliminated hazards. This is important metric to know the quality of software. i.e., Eliminated Hazards divided by Identified Hazards.

\[
\text{PHE} = \left(\frac{\text{EH}}{\text{IH}}\right) \times 100
\]

**M5:** Percentage of Software Fault Tolerance (PSFT) is a pointer of all fault tolerant components. It is gotten by looking at the number of fault tolerant components with total number of safety-critical components.

\[
\text{PSFT} = \left(\frac{\text{FTC}}{\text{TNSCC}}\right) \times 100
\]

**M6:** Percentage of Safety-Critical Errors (PSCE) is a metric for discovering number of safety-critical errors in the software. By looking at the number of safety-critical errors with total number of errors will obtain the value of M6.

\[
\text{PSCE} = \left(\frac{\text{NSCE}}{\text{TNE}}\right) \times 100
\]

**M7:** Percentage of Defect Removal Efficiency (PDRE) is a metric, which indicates the defect removal efficiency. i.e. the proportion of total number of defects resolved and total defects identified.

\[
\text{PDRE} = \left(\frac{\text{TDR}}{\text{TDI}}\right) \times 100
\]

**M8:** Percentage of Hazard Based Test Cases (PHBTC) is an indicator of number of hazard based test cases performed in the testing phase. It is the ratio between hazard based test cases and total test cases.

\[
\text{PHBTC} = \left(\frac{\text{HBTC}}{\text{TTC}}\right) \times 100.
\]

The proposed model depends on the McCall's model that particularly intended for software safety in SCCS. In the proposed model software safety is considered as an exceptional safety factor. The criteria in the proposed
The Research Framework for Quantitative Safety Assessment for Safety-Critical Computer Systems

model are Hazard Analysis, Identification of safety requirements, and completeness of requirements, safety-critical design, safety-critical coding and safety-critical testing. The proposed model with research framework is then connected on lab model called RCCS to validate its efficacy.

Figure 4. Safety Model.

The main objective of quality assurance procedures is to establish a feedback mechanism for the managerial purposes. This objective is achieved by making both the development process and the software products (i.e., software safety in our case) visible. The safety model is for designing computer controlled SCCS. The safety model can be used for implementing quality assurance procedures and enables visibility of the main safety aspects of a system through the entire life cycle. The benefits of the model are system safety integrity, simplicity and completeness. The safety integrity is a direct result of the application of safety standards and system hazard analysis.

4. Application of Safety Model to RCCS

Accidents are inclined to happen in the unmanned railroad intersections. With a specific end goal to keep the event of mishaps a RCCS is proposed. In this arrangement the methodology of the train is detected in advance and likewise the closing and opening of the railroad crossing doors is impelled. The lab model of RCCS is appeared in Figure 5 and comprises of a few sections as recorded beneath. Also, the Partial functional diagram of RCCS is appeared in figure 6.

4.1 Components of RCCS

RCCS comprises of the accompanying primary parts: Train, Railway track, Sensors, Gates, Controller with an advanced I/O card, Signals, and a muscle-wire worked track change lever. Depiction of every part of the lab model is shown underneath.

Train: The actuating force is given by the power supply transfer, to the wheels of the train, which starts the movement of the train along the track. With a specific end goal to stop the movement of the train, the actuating force is cut-off. At the point when the train approaches the gate crossing range, a sensor distinguishes the methodology of the train and sends this data to the controller segment. 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 send the signal to the controller.

Figure 5. RCCS Laboratory Prototype.

Figure 6. Partial Functional Block Diagram of RCCS.
**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. An IBM PC is assigned as the controller for RCCS. The DIO card gets the inputs from each of the nine sensors of RCCS. The eight output signals sent from DIO card control the following: The electric force supply to the train track, power supply to the two gate congregations, power supply to muscle wire based instrument to change the track lever and four signal lights.

**Gates**: RCCS has two arrangements of gates on either side of the track design, which is worked by a muscle wire based system. 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, set adjacent to the track. Signals give a sign to the train administrators that whether the track is clear or possessed, or if certain careful steps were taken or not while utilizing the track, for example, keeping up a diminished velocity. A sign post comprises of solid red and green lights.

The research framework with proposed safety model is applied on RCCS and found satisfactory results. The framework was applied on RCCS with respect to various attributes and criteria of safety development lifecycle phases. The safety metrics with Risk Assessment Factor is shown in Table 4. Two core metrics of requirements phase is percentage of incomplete requirements (M1) and percentage of requirements traceability (M2). The threshold value of risk assessment factor (RAF) for M1 is <= 0.2, i.e. incomplete safety metrics should be less than 20%. The threshold value for M2 is >=0.8. The two metrics of design phase are PHA (M3) and PsyHE (M4). Threshold value for both M3 is >= 0.8 and for M4 is >= 0.7 shown in figure 7. In RCCS the hazard elimination rate is around 73%, the remaining hazards are need to mitigate. The two metrics of coding phase is SFT (M5) and SCE (M6). Threshold values for both M5 and M6 are >=0.85. Coming to the testing the two metrics are FCM (M7) and HbTC (M8). Threshold values for both M7 and M8 are >=0.85. Therefore for the safety-critical RCCS, all the safety metrics are within the acceptable level.

![Figure 7. Quantitative Assessment of Safety Metrics for RCCS.](image)

### 5. Conclusion

This paper discusses the criteria identified with software safety. Another safety model for software safety is proposed. An arrangement safety criteria that shape the premise of software safety is displayed. The safety model can be utilized for executing safety parts of a system through the whole life cycle.

First and Second metrics characterizes early notices of the invalidity of Software Safety requirements. The results

| Safety Life cycle Phase | Criteria                                | Safety Metrics | Risk Assessment Factor (RAF) |
|-------------------------|-----------------------------------------|----------------|-------------------------------|
| Requirements            | Requirements traceability and Completeness | M1             | 81% (i.e. 0.81)              |
|                         |                                         | M2             | 15% (i.e. 0.15)              |
| Design                  | Safety-critical design and Hazard Analysis | M3             | 77.57% (i.e. 0.78)           |
|                         |                                         | M4             | 60% (i.e. 0.6)               |
| Coding                  | Safety-critical coding                  | M5             | 84.8% (i.e. 0.84)            |
|                         |                                         | M6             | 86.9% (i.e. 0.87)            |
| Testing                 | Safety-critical testing                 | M7             | 89.2% (i.e. 0.89)            |
|                         |                                         | M8             | 92.5% (i.e. 0.92)            |
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. Third and fourth metrics are utilized as a part of safety-critical design and fifth metric for safety- critical coding and sixth and seventh metrics are utilized in safety-critical testing. The outcomes in apply the safety model in building up the safety-critical RCCS unmistakably demonstrate that the system is safe, hazard free and fail-safe when contrasted with different models, which does not go out on a limb into thought. The benefits of the model are system safety integrity, simplicity and completeness. The safety integrity is a direct result of the application of safety standards and system hazard analysis. The simplicity is achieved by the general structure of SCCS. The completeness is a result of whole system safety. The proposed model is connected to a software based RCCS which is a lab model, that incorporates safety-critical operations and watched agreeable results. Thorough work is expected to meet the complete necessities of software safety perspectives that lead to standardization of model with safety metrics.

6. References

1. Raghu Singh. A Systematic Approach to Software Safety. IEEE 1999.
2. Acharyulu PVS, Seetharamaiah P. A Framework for Safety Automation of Safety-Critical Systems Operations. Safety Science. 77 (2015): 133–142. Crossref
3. NASA-STD-8719.13. Software Safety. Washington, USA: Jul 2004.
4. Air France Flight 447 (Crossref).
5. Schmid D. Adaptive Middleware: Middleware for Re- al-Time and Embedded Systems. Commun. ACM. 2002; 45:43–8. Crossref
6. Herman DS. Software Safety and Reliability Basics. Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors. Wiley-IEEE Computer Society Press; 2000. p. 520.
7. Haifeng Li, Xuyang Xie, Chang Liu. The Model-Based and Bidirectional Software Failure Mode and Effect Analysis Method. International Conference on Reliability, Maintainability and Safety (ICRMS). IEEE 2014.
8. Jacob J. Stadler, Neal J. Seidl. Software Failure Modes and Effects Analysis. IEEE. 2013.
9. Sinda Rebello. Software System Reliability and Safety Assessment: An Extended FMEA Approach. International Journal Reliability and Safety. 2010; 4(4):366–80. Crossref
10. Leveson NG, Turner C. An investigation of the Therac-25 accidents. IEEE Computer. 1993; 18–41.
11. IEEE 100. The Authoritative Dictionary of Standard Terms. IEEE Press, 2000.
12. Leveson NG. Software Safety – Why, What and How. ACM Comput. Surv. 1986; 18 (2):125–163. Crossref
13. John C. Knight. Safety Critical Systems: Challenges and Directions. Proceedings of the 24th International Conference on SE Orlando, Florida. 2002.p. 547 – 50.
14. System Safety Program Requirement. Mil-std-882. 1984.
15. IEEE STD 1228-1994. IEEE Standard for Software Safety Plans. 1994.
16. Huang Xizi. Software Reliability, Safety, Quality Assurance, Beijing, China: Publishing House of Electronics Industry; 2002.
17. Weaver R, Fenn J, Kelly T. 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). 2003; 33: 57.
18. Raheja D. A Different Approach to Design Review. In: Reliability Review, American Society for Quality assurance. 1982.
19. Raghu Singh, Software Quality: A Tutorial. 1998.
20. Bhansali PV. Software Safety: Current Status and Future Directions, ACM SIGSOFT Software Engineering Notes. 2005 January; 30(1):1. Crossref Crossref2 Crossref3
21. Kristian J. Cruickshank and James Bret Michael, Man-Tak-Shing. A Validation Metrics Framework for Safety-Critical Software-Intensive Systems 2010.
22. James Bret Michael, Man-TakShing, Kristian John Cruickshank, and Patrick James Redmond. Hazard Analysis and Validation Metrics Framework for System of Systems Software Safety, IEEE Systems Journal. 2010; 4(2).
23. Anneliese Andrews, SalwaElakeili, and Salah Boukhris. Fail-safe test generation in safety critical systems. IEEE 15th International Symposium on High-Assurance Systems Engineering (HASE). 2014.p. 49–56.
24. Salwa Elakeili. Fail-Safe Test Generation Of Safety Critical Systems. Dissertation Book. University of Denver; 2015.
25. Srinivas Acharyulu PV, Seetharamaiah P. A Measures and Metrics Framework for Software Safety. ACM SIGSOFT Software Engineering Notes. 2015; 40(1):1–8. Crossref
26. McCall J, Richards P, Walters G. Factors in Software Quality (3 Vols). NTIS AD-A049.1977.p.176.
27. Benswarup Medikonda,Seetharamaiah P. An Approach to Modeling software safety in Safety-critical systems. Journal of computer Science. 2009; 5(4):311–22.
28. Jayasri Kotti, Seetharamaiah P. A GQM Based Approach towards the Development of Metrics for Software Safety. Journal of Computer Science. 2015; 11(6): Crossref.
29. Jayasri Kotti, Seetharamaiah P. An Experimental Safety Analysis using SFMEA for a Small Embedded Computer Control System. International Journal of Innovations in Engineering and Technology (IIJET). 2016; 7(3): 342–51.
30. Jayasri Kotti, Seetharamaiah P. The Quantitative Safety Assessment and Evaluation for Safety-Critical Computer Systems. ACM SIGSOFT Software Engineering Notes., ACM New York; NY, USA: 2016; 41(1): Crossref.