

NFRs Model for Nuclear Power Plants

Ehab Shafei
Operation Safety and Human
Factors Department
Nuclear and Radiological
Regulatory Authority
Cairo, Egypt

Hany Sallam
Operation Safety and Human
Factors Department
Nuclear and Radiological
Regulatory Authority
Cairo, Egypt

Mostafa Aref
Computer Science Department
Faculty of computer and
information sciences, Ain
Shams University
Cairo, Egypt

Abstract: Requirements analysis phase plays a vital role in drawing the performance and characteristics of critical software systems. As the requirements were global, detailed and complementary as the system was successfully functioning, free of errors and flaws, and adapted to environment dynamicity. In critical systems, such as Nuclear Power Plants (NPPs), implementing software functional requirements (FRs) is not enough to ensure system safety. Non-functional requirements NFRs implementation beside FRs becomes crucial for ensuring such function. NFRs performs other functions that are essentials for system availability, reliability, and dependability. NFRs should be supportive, not precluding to FRs, and keep system complexity and cost as low as possible. To this end, this paper proposes a model for NFRs which have importance in nuclear field based on safety system classification, and graded approach which assign the quality attributes and constraints to a given system based on its importance to safety. This model helps in enhancing the system overall safety without increasing the system complexity and implementation cost without need.

Keywords: critical software system, requirements analysis, Nuclear Power Plants

1. INTRODUCTION

Safety critical software systems are considered as computer-based systems which of special concern in which failure of the system could lead to significant economic, physical damage to organizations or people injury. Such systems increasingly deployed in many critical systems such as, nuclear power plants, radiotherapy, aircrafts, and many medical devices. These systems rely on the use of safety critical software in controlling and monitoring critical devices. Success of such systems depending on the software requirements analysis. Requirements analysis is considered to be the most important phase in the software development lifecycle. It is widely recognized that, of all phases in software, it considered to be the most crucial task in software engineering. System requirements are divided into FRs, and NFRs. NFRs are often more critical than FRs in the determination of a system's perceived success or failure. According to Kotonya and Sommerville, the NFRs define the overall qualities of the resulting system that are often critical in nature, and sometimes functional requirements may need to be sacrificed to meet these non-functional constraints [1]. Ineffectively dealing with NFRs has led to a series of failures in software development [2], [3], as happened in well-known case of the London Ambulance System [4], where the deactivation of the system right after its deployment was strongly influenced by NFRs noncompliance. Literature [5], [6], [7] has been pointing out the difficulties of dealing with these requirements and showing that errors due to NFRs are the most expensive and difficult to correct.

Literature review also shows that, NFRs are often poorly understood and not considered adequately in software development due to the characteristics of NFRs, and difficulties. Also because there is no consensus about them [8]. Christoph Marhod et al. [9] show that there are many problems related to representing NFRs more than FRs. These problems causing the user non satisfaction and can expensive downtime or even complete failure of the system [10]. Requirements should be complete and express the entire need

and purpose of the system and also should manage all conditions and constraints under which it applies [11].

The paper is organized as follows: section 2 introduces the related work. Section 3 represents the proposed NFRs model for NPPs. The last section concludes the discussion, and explores trends for future research work.

2. RELATED WORK

Critical software requirements analysis don't take into account the requirements imposed as a result of integrating the software with the environment and also other requirements related to the performance and the quality of the software, and the human interaction with the software. The following different kinds of requirements may be incomplete because different component parts of them are missing [12] such as data requirements, interface requirements, quality requirements, and constraints. In 1997, Gilb classified requirements to functions, qualities, costs and constraints [13]. The last three are regarded to NFRs. Qualities denote "How well the function will perform" and "any restrictions on the freedom of requirements or design" relates to the constraints. Gilb's classification emerged due to the presence of unwanted or undesirable requirements or if it's false, unclear, and/or not possible to assess their satisfaction. IEEE Standard "IEEE Std-830-1993" [14] attempting to classify and specify NFRs. Glinz [15] classified NFRs as performance and quality related requirements that could be described using four facets: representation, satisfaction, kind, and role. In ISO 25010 [16] software quality model is defined, which composed of eight attributes. The attributes are reliability, performance, suitability, efficiency, security, portability, maintainability, and compatibility. According to the nature of the application domain, some of these NFRs are prioritized. NFRs such as security and reliability have more importance in safety critical systems than other systems [17]. Each system has a specific nature which requires suitable NFRs to be fulfilled according to its function and environment. The NFRs presented in [18] used in distributed control system in automation domain, the presented NFRs are reusability, modularity, interoperability,

resource utilization, reliability, time behavior, analyzability, and installability. We are not aware of any other comprehensive approach to the NFRs classification. Critical software NFRs should be not limited to the existing quality attributes of NFRs but should be extended to include other important NFRs specifically related to the system context, platform, and environment in which system is integrated. These NFRs represent the constraints that should be applied in the system according to its critical level. That the software should execute in a system context without contributing to unacceptable risk.

3. NFRs MODEL

This section provides a model for NFRs which have importance in NPPs based on safety system classification and graded approach which assigns the quality attributes and a set of suitable requirements to a given system based on its importance to safety. This model helps in enhancing the system overall safety without increasing in the system complexity and implementation cost without need. As the system environment is dynamic and has different modes of operation, consequently different requirements for each mode of operation are expected. As human or operator plays an important role in system operation and management, this requires a set of suitable requirements to improve and enhance interface with the system to avoid operator errors. Since the software is vulnerable to cyber-attacks which have severe consequences on system's safety, so security has a special concern and importance. Time is very important NFR, which the system should take action on time as required. Software input data, intermediate data (processed data), and output data can lead to system failure, and consequently lead to accident if they are not accurate, or incorrect. These NFRs may tend to be related to one or more FRs but they aren't FRs. These NFRs are essential for the system to be able to perform its functions safely.

The NFRs model for NPPs composites of two levels of NFRs, quality attributes level, and a set of suitable requirements for system application level as shown in Figure 1. The quality attributes level includes the essential set of NFRs, which are mandatory for such applications such as reliability, robustness, usability, maintainability, testability, and availability. The system application level, which includes NFRs that represent the required requirements and constraints according to the application nature in NPPs, and criticality of the system. These NFRs are data, modes of operation, system integration, security, and time. These NFRs should be considered in the design and implementation phases to ensure the safety of the system. Therefore, the designer of software should be considered and commensurate with the identified NFRs and related constraints.

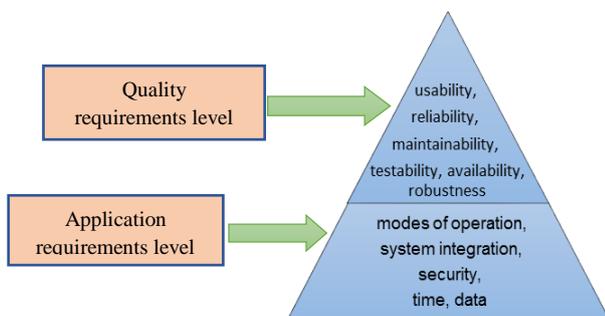


Figure 1 NFRs quality model

Each system in NPPs has a certain degree of criticality according to the importance of safety function to be performed, consequences of failure, period of time for which, the system will be called upon to perform a safety function and the calling frequency of the system to perform the required safety function. Systems criticality has a direct proportional relation with systems severity, which means high critical systems have high severity in case of failure. To design NFRs according to system importance to safety and its criticality, we have to present the systems classification schema in NPPs. Each system in NPPs performing a specific function and accordingly it is classified into one of three classes according to its function in the plant and severity in case of failure. Table 1 shows the relation between system criticality and system safety classification in NPPs.

Table 1 Npp systems safety classification

Safety class	System criticality (severity)
Class 1 (safety systems)	High
Class 2 (Safety related)	Medium
Class 3 (Non safety)	Low

The three safety classes are:

- Safety class 1:** contains safety systems which perform safety function such as reactor protection system, and whose failure would lead to consequences of high severity;
- Safety class 2:** contains safety related systems which perform safety related functions such as safety related monitoring and alarm system (fire alarm system, seismic information and control system). These systems do not impact safety directly, but may cause the NPP trips. In case of its failure would lead to consequences of medium severity;
- Safety class 3:** contains independent systems such as information processing and monitoring system for non-safety systems that do not impact NPP safety or trips (radiation monitoring system). The failure of these systems would lead to consequences of low severity.

For each safety class, there is a set of quality NFRs and a set of suitable application requirements which are designed according to system criticality. Table 2 illustrates the system classification in NPPs and associated suitable NFRs. Also the application constraints which have to be applied in the software design and implementation, and should be commensurate with the class of the target system, which can be key elements of meeting safety of the system.

In safety systems (safety class 1), based on the fact that these systems perform safety function such as RPS, which brings the reactor to a safe state when the safety setting value is reached by shutdown the reactor. All presented NFRs should be exist in this system. That the system should be: reliable to be trusted, available all the time because it performs safety

function. The system also should be maintainable in case of failure occurred, testable at any time to check its operability, robust to have the ability to cope with errors during execution, and usable by the operator in easy way. While, safety critical system performs its function in reactor operation mode, it also continues to monitor the reactor variables after the reactor shutdown to inform the operator about the reactor state and values of the variables, so the requirement of modes of operation should be well addressed in the design, and implementation phases. While the safety critical system is an embedded system and operates in a dynamic environment, so the integration and compatibility between its components (software, computer system hardware, sensors, actuators, network, and operators) is very important and should be taken into consideration during the software failure analysis, design, implementation, and testing. As any digital system, critical software system is vulnerable to cyber-attacks, where such software system processes sensitive data and executes safety function, software security becomes an essential and crucial NFR. Software security is concerned about preventing unauthorized access to the running programs, and related data used since such access could result in a system malfunctioning due to intentional change to system settings and data values. For these reasons, software security should be addressed in the design, implementation, and test phases of the software. As this system performs a safety functions, so its safety decision should be taken in the required time, the time is very crucial constraint for successful safe operation. The critical software system function depends on input and output data, so the data constraints should be considered in different software development phases to ensure the safety and reliability of the system.

In safety, related systems such as a fire alarm control and information system, this system is safety related, and does not impact safety directly but may cause NPP trip. It has a medium importance to safety, so not all NFRs have to be considered in the software development lifecycle, such as modes of operation, and some of these NFRs may be considered partially i.e. not all constraints should be considered such as security, time, and data. While these systems operate in all operation states not specific for certain one, so NFR operation modes are not considered. Also, these systems do not deal with sensitive data and the probability to be attacked is medium, so security can be assured by any commercial security programs or security hardware. Not all constraints of time and data addressed in this system.

In non-safety systems, such as radiation information and monitoring system. These systems do not affect the NPP safety, and if these systems failed, there are alternative methods (detection devices) which can perform the same function. Not all NFRs addressed in the development of these systems such as availability, maintainability, robustness, modes of operation, system integration, and security, and some of them addressed partially such as time, and data. For example, security can be assured through physical security or even by system password because the system does not have sensitive data and not vulnerable to cyber-attacks. While radiation values can be monitored by another system, so the data requirements and constraints are important to be addressed, they are addressed partially. Time also is partially addressed to monitor the radiation value at the required time.

Table 2 NPP safety systems classification and associated NFRs

Safety class NFRs	Class 1	Class 2	Class 3
Reliability	x	x	x
Availability	x	x	
Maintainability	x	x	
Robustness	x	x	
Testability	x	x	x
Usability	x	x	x
Modes of operation	x		
System integration	x	x	
Security	x	partially	
Data	x	partially	partially
Time	x	partially	partially

In this paper, we focus on discussing a set of suitable NFRs for safety class 1 systems in NPPs such as system integration, security, mode of operation, time, and data, which represent crucial requirements for class 1 systems in NPPs. These requirements are essential for these systems and should be fulfilled in the systems to improve the safety. Each one of these NFRs may be further decomposed into a set of constraints. These NFRs, and related constraints are explained in the following subsections.

3.1 System Integration

Each critical system embedded in critical environment includes software system which controls the operation of the system and the monitoring and supervision system which receive information from the critical software system and send control signal to the critical software. In such environment the integration and compatibility between the system components (software, computer system hardware, sensors, actuators, network, and operators) is very important and should be taken into consideration during the software failure analysis, design, implementation, and testing. The integration between these components of that system may lead to accidents if the design didn't consider the constraints related to this integration. That there is an affective relation between software and other components in the critical systems and should take into account the integration between software and the following components:

- a) Hardware: through hardware interface module which can take inputs from sensors and give outputs to actuators, and other subsystems. During the development of software, the logical and physical characteristics of the interface between the critical software and the hardware components of the critical system should be identified. This may include the

supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used. For each sensor, the received values from the sensor in terms of data ranges, units, precision, error bounds, meaning, etc should described. For each actuator, describe the sent values to the actuator in terms of data ranges, units, precision, error bounds, meaning, etc. Also constraints and specifications for computer system hardware should be considered which includes physical devices that assist in the transfer of data, and perform logic operations such as busses, memory cards, and Central Processing Units (CPU). Based on the fact that the operating system has a crucial rule in the software operation. There are related constraints that should be considered such as process and stack management, exception handling, flow control, memory scheduling and allocation. These constraints have repercussion on the function safety.

- b) Network: The network and infrastructure for each software system should be identified in terms of networking traffic, data transfer rates, error checking mechanism, input and output communication ports, interrupts, message format and throughput, exception handling and error recovery, and finally synchronization mechanisms.
- c) Operator: through a human interface (human system interaction)

In safety critical systems, the main goal of the user interface is to allow operators to carry out activities such as monitor and supervise the system effectively and safely. The human system interaction has a great impact upon the human performance, which needs to be well designed. Many spectacular system failures are caused by human and user interface design errors. Many of accidents and events referenced to misinterpretation of system parameters consequently operators taking incorrect action, which leads to an accident as in Three Mile Island [19], the much publicized London Ambulance Service, and Therac-25 accidents, were attributable to poor operator Graphical User Interface (GUI) design as well as unreliable control software [20], [21]. High interface usability is aiming to make the operator more comfortable and reduce anxiety. The interface requirements should describe the logical characteristics of each interface between the critical software and operators. Establish criteria for monitoring the transmission of data between systems, including the identification of error conditions. This includes also sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, emergency windows, and so on. Define the software components for which an operator interface is needed. The system should interact with the user in an effective way that the system should notify the operators (send feedback) to operators that it takes a suitable time to complete an action. If the system cannot meet the required response time limits should keep users informed about what is going on for example if the required action takes more than one second, the system should notify the operator, and if it takes more than 10 seconds to allow the user to interact according to certain procedure. The following requirements and constraints are related to operator interfaces which should be considered:

1. The software interface should have the capability to handle a large amounts of information by means of scrolling, overlapping windows, and hierarchies of displays;
2. Error message should be visible, explicit, readable, precise, and constructive advice;
3. Human reaction and decision times (grace period) should be identified;
4. Menu techniques, colors, underlining and blinking on displays should be identified carefully and designed;
5. The alarms should be monitored easily and according to its criticality, by using color coding to distinguish the importance of alarms; the first priority of alarms is represented by red, the second by yellow, and the third by green;
6. Help menus, and emergency procedures menu should be clear, visible and not complicated;
7. Layout of controls and displays should be designed carefully;
8. The graphic module configuration should be identified and designed to be responsible for picture display parts, such as flow charts, trends, and alarms.

3.2 Operation Modes

Many of critical systems in NPPs have different modes such as startup mode, normal operation mode, maintenance mode, and shutdown mode. For each operation mode, there are many safety requirements. This type of NFRs should cover the different operating modes, and specify the protective action that should be taken in case of incidents. Each mode of operation determines the data to be processed, data to be monitored for operators, and certain allowed operator action. That some of operator actions can be locked. Critical software system should have the ability to perform self-management for procedures, and functions depending on different modes of critical system. Also should have the ability to switch between different procedures and functions and initiate other functions according to critical system modes of operation. These are not functional requirements in themselves, but constraints associated with each mode of operation such as:

- a) Identify sufficient error logging, that in case of software failure or critical system failure, the critical software should have the ability to detect such failures.
- b) Requirements regarding to modification request procedures according to different critical system operation mode;
- c) The allowed and not allowed operations in each mode such as safety setting valued modification is forbidden in operation mode;
- d) Time period required for moving from one mode to another;
- e) Error handling in each mode should be identified;
- f) Alarm or action triggers for each mode should be identified;
- g) Establish system health check procedures.

3.3 Data

The NF data requirement should be identified based on a set of constraints as shown in Figure 5. These constraints are very important for critical software systems in NPPs. According to the input data, there is a decision will be taken, this decision may be related to a safety function such as shutdown the nuclear reactor. Also, the data have importance in the calculations which are performed in the reactor. So, the input

data should be accurate, in range, arrive in the required time, and represented correctly. In addition to the retention of the data which should be identified. These constraints should be identified and checked during the software design, and operation. The representation of the data is important also for the operator to easily monitor the plant and take the appropriate action according to the data represented.

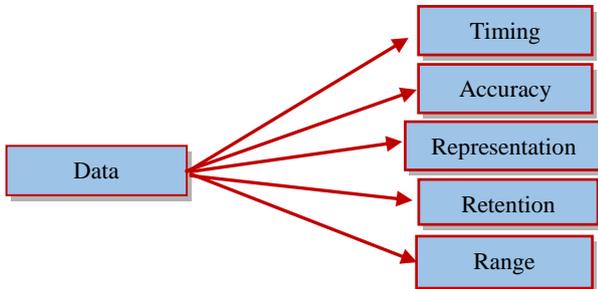


Figure 5 Data constraints

The constraints of data attribute are illustrated as follows:

- Timing:** the input data should be received on time not after or before the required time;
- Accuracy:** the accepted degree of data accuracy should be identified. The data to be correct, its values should be in the right value and represented in a consistent and unambiguous form;
- Representation:** the convenient way data representation for operator usage should be identified to make an easy and fast evaluation for system status;
- Retention:** define the length of time data needs to be retained after it is no longer considered active. Define whether the data are required to be available real time or can be stored in an archive;
- Range:** the input data range with both limits, high and low should be identified and checked.

3.4 Time

The time as NFR has a vital role in critical real time software systems for NPPs. These systems should response to any designed or undesigned action within a certain period of time. There are constraints of time that should be considered in the design and implementation phase to guarantee that the system response within the specified time as shown in Figure 6 such as response time, startup time, processing time, and hardware failure detection time. These constraints should be continuously checked during the runtime of the software to update the operator in case of any degradation occurred.

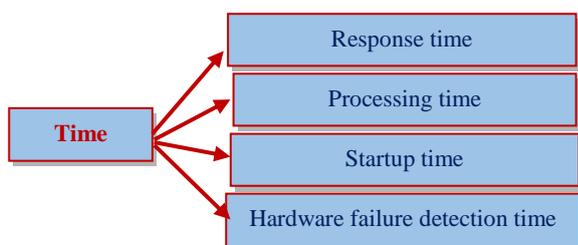


Figure 6 Time requirements and constraints

3.5 Security

With the growing trend in using safety critical software as an embedded system in many critical applications in NPPs,

where such software systems contain sensitive data and perform a safety function, software security becomes an essential, and crucial NFR. Software security is concerned about preventing unauthorized access to the running programs, and related data used for such access could result in a system malfunctioning due to intentional interference. The consequence of such interference could be an accident or system fail to perform its intended protective action. So Software security aims to maintain and preserve confidentiality, integrity, and availability as shown in Figure 7. The result or impact of the attack might include:

- Denial of service/loss of function: blocking the operator's ability to observe and/or respond to changing system conditions, speeding the system down and may affect the system availability.
- Interception: intercepting and modifying data streams passed between systems or corrupting the data and this may affect the system behavior and consequently its safety.
- Unobserved system monitoring and data collection: unauthorized file access and data recording, including the message (information) intercept.
- Operator spoofing leading to incorrect action: injecting anomalous readings into a control panel, causing the operator to take incorrect action.
- Direct manipulation of computer systems: giving the attacker independent control over processes and machinery.



Figure 7 security requirements

Based on the fact that attacks may come in each phase of software as a result of drawbacks and shortage in these phases such as design errors, and source code bugs. So critical software system should be developed in secured environment and each phase of software development should be designed, implemented, and executed under suitable security measures for each phase. Security for critical software should provide in terms of secure models, secure coding practices, and secure development procedures. Also security should be assured during deployment and maintenance phases. Exploitable faults and other weaknesses are eliminated to the greatest extent possible by efficient design, and well-intentioned engineers. The security design should be based upon certain threats/threat types, identified security goals, security requirements and security functions as shown in Figure 8. Assurance for such objectives are achieved by implementing the identified security functions. Security controls such as auditing, reviewing, and testing should not be limited to the requirements, design, implementation, and test phases of the software lifecycle. It is important to continue performing code reviews, auditing and security tests, during deployment, operations, and in case of updating to ensure that updates do

not add security weaknesses or malicious logic to the software.



Figure 8 Security design flow

4. CONCLUSION

This paper established the base for designing and implementing NFRs applied for NPPs critical software systems and presented a new model for NFRs. More NFRs means more quality, but on the side leads to more cost and more systems complexity, based on this fact, selecting and implementing NFRs for a given system should be decided by its the importance to the system and how much the selected NFRs will improve the system's quality. For this reason, in this paper designing and implementing NFRs for NPPs were engineered by using the graded approach as a well-known approach in the nuclear field. Different NPPs systems are classified into three classes, safety, safety related, and non-safety according to severity of consequences if one of an NPP systems failed to perform its required function. The new NFRs model based on a graded approach to assign NFRs for each class and justify the required constraints associated with each attribute specially for safety class. This model is characterized by correlating between the system function importance to safety and supportive NFRs to be designed and implemented in the system to improve the system performance quality without more unrequired excessive complexity.

5. REFERENCES

- [1] K. Gerald and S. Ian, Requirements engineering: Process and Techniques, Wiley 2000.
- [2] K.K. Breitman, J.C.S.P. Leite, and A. Finkelstein, "The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study," J. the Brazilian Computer Soc., vol. 6, no. 1, pp. 13-38, July 1999.
- [3] D.R. Lindstrom, "Five Ways to Destroy a Development Project," IEEE Software, pp. 55-58, Sept. 1993
- [4] A. Finkelstein and J. Dowell, "A Comedy of Errors: The London Ambulance Service Case Study," Proc. Eighth Int'l Workshop Software Specification and Design, pp. 2-5, 1996
- [5] F.P. Brooks Jr., "No Silver Bullet: Essences and Accidents of Software Engineering," Computer, no. 4, pp. 10-19, Apr. 1987.
- [6] A. Davis, Software Requirements: Objects Functions and States. Prentice Hall, 1993.
- [7] L.M. Cysneiros and J.C.S.P. Leite, "Integrating Non-Functional Requirements into Data Model," Proc. Fourth Int'l Symp. Requirements Eng., June 1999.
- [8] Jane Cleland-Huang, Raffaella Settini, Oussama BenKhadra, Eugenia erezhanskaya, Selvia Christina, Goal-Centric Traceability for Managing Non-Functional Requirements, ICSE'05, May 15-21, 2005, St. Louis, Missouri, USA. Copyright 2004 ACM 1-581 13-963-2/05/0005.
- [9] Christoph Marhold¹, Clotilde Rohleder², Camille Salinesi², Joerg Doerr³, Clarifying Non-Functional Requirements to Improve User Acceptance – Experience at Siemens.
- [10] Brooks Jr., F.P., "No Silver Bullet: Essences and Accidents of Software Engineering," IEEE Computer., 4, (Apr. 1987),10-19.
- [11] Ralph R. Young, The requirements Engineering Handbook, Artech House, 2004.
- [12] Ben Swarup Medikonda, Seetha Ramaiah Panchumarthy, "A Framework for Software Safety in Safety Critical Systems", SIGSOFT Software Engineering Notes, Vol. 34, No.2, March 2009.
- [13] Gilb T., Towards the Engineering of Requirements", Requirements Engineering Journal, vol. 2, no. 3, pp. 165-169, 1997.
- [14] IEEE, IEEE Recommended Practice for Software Requirements Specification. IEEE standard 830-1993, 1993.
- [15] Glinz M., "Rethinking the Notion of Non-Functional Requirements", in Proceedings of the 3rd World Congress for Software Quality, Munich,Germany, pp. 55-64, 2005.
- [16] ISO/IEC 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation, 2011.
- [17] I. Sommerville, Software Engineering.: Addison Wesley, 2006.
- [18] Timo Frank, Martin Merz, Karin Eckert, Thomas Hadlich, Birgit Vogel-Heuser, Alexander Fay, Christian Diedrich, " Dealing with non-functional requirements in Distributed Control Systems Engineering", 978-1-4577-0018-7/11/\$26.00, IEEE ETFA'2011.
- [19] Timo Frank, Martin Merz, Karin Eckert, Thomas Hadlich, Birgit Vogel-Heuser, Alexander Fay, Christian Diedrich, " Dealing with non-functional requirements in Distributed Control Systems Engineering", 978-1-4577-0018-7/11/\$26.00, IEEE ETFA'2011.
- [20] Ben Swarup Medikonda, Seetha Ramaiah Panchumarthy, "A Framework for Software Safety in Safety Critical Systems", SIGSIFT software Engineering Notes, Vol.34, No.2, 2009.
- [21] M.Ben Swarup, P. Seetha Ramaiah, "A Software Safety Model for Safety Critical Applications", International Journal of Software Engineering and Its Applications, Vol.3, No.4, 2009.