# Application of Hazard Analysis to Software Quality Modelling

Hong Zhu, Yanlong Zhang, Qingning Huo and Sue Greenwood
*Dept of Computing, Oxford Brookes University, Wheatley Campus, Oxford, OX33 1HX, UK*
*Emails: (hzhu | yzhang | qhuo | sgreenwood) @brookes.ac.uk*

## Abstract

*Quality is a fundamental concept in software and information system development. It is also a complex and elusive concept. A large number of quality models have been developed for understanding, measuring and predicting qualities of software and information systems. It has been recognised that quality models should be constructed in accordance to the specific features of the application domain. This paper proposes a systematic method for constructing quality models of information systems. A diagrammatic notation is devised to represent quality models that enclose application specific features. Techniques of hazard analysis for the development and deployment of safety related systems are adapted for deriving quality models from system architectural designs. The method is illustrated by a part of web-based information systems.*

## 1. Introduction

Software quality is a complex and elusive concept [1]. There are dozens, even hundreds, of attributes that are associated with the quality of software and information systems. In the past few decades, researchers have built models to understand, measure and predict the quality of software and information systems [2]. These models form general guidelines for the elicitation of users' quality requirements. They help software designers to seek technical solutions to achieve required quality. They are the foundation for effective organisation of quality assurance activities, for example, testing can be directly targeted to the quality issues that are important to the system.

Existing work in the literature on software quality falls into two categories: quality models and their construction methods. Among the best known quality models are hierarchical models, such as the McCall model [3], the Boehm model [4], and the quality model of ISO 9126 [5]. The SOLE model [6] and its variants are hierarchical quality models of information systems. They organise the hierarchic structure according to the views from three different groups of stakeholders: users, technical staffs and managers. In [7], a quality model for websites of universities, called Website QEM, was proposed based on the users' view. It breaks down the quality of websites into more than a hundred attributes. Such models represent the positive influences between quality attributes, but fail to represent more complicated relationships. Relational models, such as the Perry model [8] and the Gillies model [2, 9], characterise the relationships between quality attributes by a number of stereotypes of relations, including positive, negative and neutral impacts of one attribute on another. Quantitative models of software quality usually appear in the form of software quality metrics so that the measurement of quality on an attribute is calculated from measurements of other attributes; cf. [10, 11]. Existing quality models were intended to be comprehensive and applicable to all software development. However, as pointed out in [1], there can be no single and simple measure of software quality acceptable to everybody. Every software system may have its own quality concerns [12]. Special requirements of the application must be considered in the use of quality models [12,13]. With the ever-growing range of computer applications, software engineers are seeking for quality models that can provide useful insight information not only for quality management, but also for supporting other development activities. How to develop such quality models remains an open problem.

Existing quality models are constructed based on many years' experience in the development and maintenance of software and information systems. The validation of such models is by empirical studies, by analysing data collected from questionnaires and interviews, for example [8]. A systematic method is necessary to construct testable, assessable, and refineable quality models for different software products and key products of software development. In [14, 15], Dromey proposed a generic quality model and a process to systematically develop software quality models. The generic quality model consists of three principal elements: product properties that influence quality, a set of high-level quality attributes, and a means of linking them. The generic model is instantiated and refined for a particular software product through a five-step model construction process. Dromey demonstrated the application of the method to software requirements definition, design and implementation. Recently, Bansiya and Davis [16] also applied the method to build a hierarchical model of object-oriented design quality. Although Dromey correctly recognised that a

quality model must be built through quality-carrying properties of the components of the software product, the applications of the method only produced universal quality models. The specific features of the application area and system design and implementation were not considered. The card sort method proposed in [17] to elicit the quality attributes of web-based applications provided a partial solution to the problem. A shortcoming of the approach is that it can only be applied after the completion of the development of the web site.

In [18, 19], we proposed a method to systematically derive quality models from architectural designs of information systems. It adapted hazard analysis methods to enable software engineers systematically identify certain types of quality attributes and the quality carrying properties of each component and connector, and to establish the links between them. Case studies of the methods have been carried out for b-to-c and b-to-b e-commerce systems. These case studies have demonstrated the applicability and a number of advantages of the quality modelling method. First, it enables the practitioners and researchers to develop their own quality models for individual systems. Second, as many applications in a specific application domain often share a common architectural structure, the method can also provide a quality model for those systems of the same architecture and in the same application domain. Moreover, the method is applicable at a relatively early stage of system development process. Finally, the method can provide more insight information of the system than existing quality models and quality modelling methods.

This paper further develops the method by proposing a new representation of quality models and a process for the derivation of such quality models. The paper is organised as follows. Section 2 proposes a diagrammatic notation for the representation of quality models. Section 3 adapts a hazard analysis method for the derivation of quality models from system architectural designs. Section 4 concludes the paper with analysis of the proposed method and discussion of future work.

## 2. Representation of quality models

Existing software quality models have been represented using simple and intuitive notations. For example, hierarchical models are represented in the form of a tree with nodes as quality attributes and arcs as positive relations between the attributes. Relational models use matrices that each row and column represents a quality attribute and the values of the matrix represent the stereotype relations between the corresponding attributes. Such representations do not refer to any elements of the software product whose quality is under investigation. Hence, they are independent of the software product. These representations are suitable for universal quality models that are intended to be applicable to all software systems. Although such models do play significant roles in software and information system development, quality models that make no explicit references to the product specific features have limited usefulness. We believe in Dromey's principle that abstract quality attributes must be linked to the tangible software properties through the quality-carrying properties of each component. However, in Dromey's method, the model of a software product is only used as a tool. The result quality models make no explicit reference to the components of the product. Here, we argue that how a quality-carrying property of a component is related to a quality attribute of the system is important because it provides the sort of insight information that can significantly improve the usability of quality models.

For example, safety is an important quality attribute of safety critical systems. It is of extreme importance for engineers to understand how faults and failures of the components are related to the safety of the system. Only when such information is available, can design solutions be put forward to eliminate the specific types of faults of the component and to prevent the occurrences of the specific types of failure modes that may contribute to safety. Moreover, testing of the system can then directly target the safety-related components and events to ensure system safety.

The quality attributes / quality-carrying properties of a component, such as usability and maintainability, are usually abstract. Consequently, the links between two abstract properties cannot be easily established or validated. However, abstract properties usually demonstrate themselves through various concrete events and observable phenomena, which are tangible and observable. For example, the poor usability of a web page is clearly demonstrated if the user cannot find the required information through the hyperlinks. While relationships between abstract properties are difficult to establish and validate, the relationships between observable phenomena are often self-evidence in the context of the system. For example, when an HTML file contains a large number of broken links (i.e. it is incorrect), the usability of the system will be poor because the user would not be able to find the information through the hyperlinks. This example shows that if the observation of one phenomenon implies the occurrence of another phenomenon, the corresponding abstract properties must have an implication relationship. Many authors have used such rationale in the construction of quality models. Unfortunately, such rationale has never been included in existing quality models. We believe that such information is crucial for the testability of the quality model. For example, in the design and analysis of safety critical systems, we not only need to know if the system is safe, we also need to know how the system will behave if certain event happens. This provides the crucial information for software testers to develop test cases to

check if the system correctly implements the safety as designed.

In summary, we identify the following requirements on the representation of quality models.

*Requirements 1*: A quality model should explicitly associate quality attributes / quality carrying properties to the components of the system.

*Requirements 2*: A quality model should associate abstract properties with observable and verifiable phenomena of the components / system.

*Requirements 3*: A quality model should be able present the rationale of the relationships between the properties. Such rationales can be system specific and should be able to be verified and validated in the context of the system.

Therefore, as shown in Figure 1, our proposed diagrammatic representation of quality models is a directed graph, which consists of two principal elements: the nodes and links. Each node contains three basic elements: (1) the component of the system; (2) the quality-carrying properties of the component; and (3) the observable phenomena of the property. The links are directed arcs between the nodes. A link from node A to node B means that the observation of the phenomenon on node A implies the occurrence of the phenomenon on node B. Each link can contain an optional annotation for the reasons why the two nodes are related.

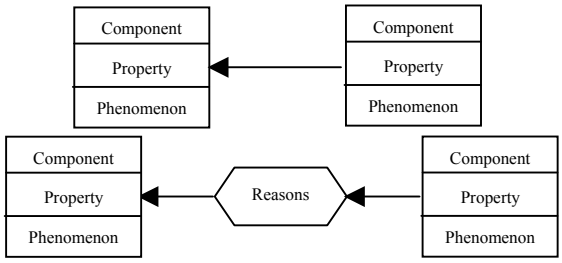


**Figure 1.** Notation for representation of quality models

Figure 2 shows a fragment of a quality model of Web-based information systems. This fragment of a quality model only shows that the usability of a web-based system is related to the correctness, responsiveness, structuredness of HTML files, compatibility of client-side platform, and the usability of the online help subsystem. It also indicates in detail how these properties are related to whether the user can find the required information. For example, if a file is of large size, it will have a long response time. If the response time is longer than the time-out setting, the browser will regard the requested file as unavailable. It also shows that the compatibility of the code on the client side will affect the usability, while the compatibility of the server side does not. According to this quality model, to achieve a good usability, the software designer should make each web page in a reasonable size to avoid excessive response time. The model also indicates that the testers should check if there are any broken links in the Web pages to ensure the usability, and so on. It should be noticed that, the links between the nodes must be understood as the implications of one phenomenon to another, rather than simply the relationship between two quality attributes. For example, large sized HTML files may contain less hyperlinks between them than smaller sized files. This makes the navigation between the files easier. Consequently, the user may find it is easier to use. Therefore, it is positively related to the usability of the system. On the other hand, large sized HTML files will increase the response time and in extreme cases it may cause poor usability. Such complexity cannot be represented in a quality model that only relates two abstract quality attributes as in hierarchical and relational models.

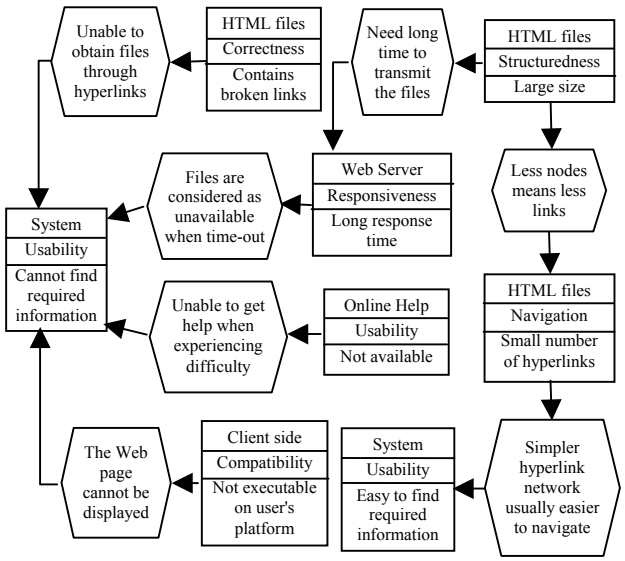


**Figure 2.** A fragment of quality model of Web-based systems

## 3. Derivation of quality models

Our method for the construction of quality models takes structural models of information systems as input. It applies hazard analysis methods to derive observable quality sensitive phenomena of the behaviour of the components or the system and establishes the causal relationships between the phenomena. The quality carrying property / quality attribute that a phenomenon demonstrates is then identified according to the nature of the phenomenon. These elements are then assembled together and represented in the diagrammatic notation given above.

### 3.1. Adaptation of hazard analysis method

Hazard analysis techniques have been widely used in the development and deployment of safety critical systems that involve computer software or not. Originally, hazard analysis intends systematically identifying, assessing and controlling hazards before a new work process, piece of

equipment, or other activity is initiated. In such a context, a hazard is a situation in which there is actual or potential danger to people or to the environment. Associated with each hazard is a risk, which is related to the likelihood of the event occurring and its consequences. Once the hazards are identified and analysed, safety as well as other quality requirements can be specified for each component. Risks can be avoided or reduced ultimately through technical design, management and organisational means. Consequently, the quality and reliability of the system are improved [20, 21, 22, 23].

In [18, 19], we adapted the methods of hazard analysis and extended the concept of hazard to construct quality models of information systems. In our context, the word hazard has its widest meaning, which means any situation that may cause harm. The more likely a hazard occurs and more serious the consequences of the hazard, the more important the corresponding quality attribute, and *vice versa*.

There are a number of hazard analysis techniques available in the literature. We are particularly interested in the FMEA (failure modes and effects analysis) technique. FMEA progressively selects the individual components or functions within a system and investigates their possible modes of failure. It then considers possible causes for each failure mode and assesses their likely consequences. In the original FMEA, the effects of the failure are determined for the unit itself and for the complete system. Possible remedial actions are also suggested. A simple example of FMEA is given in Figure 3.

| FMEA for a microswitch | | | | | | |
|---|---|---|---|---|---|---|
| No | Unit | Failure mode | Possible cause | Local effects | System effects | Remedial action |
| 1 | Tool guard switch | Open-circuit contacts | Faulty component | Failure to detect tool guard in place | Prevents use of machine - system fails safely | Select switch for high reliability and low probability of dangerous failure |
| | | | Excessive current | | | |
| | | | Extreme temperature | | | Rigid quality control on switch procurement |
| 2 | | Short-circuit contact | Faulty component | System incorrectly senses guard to be closed | Allows machine to be used when guard is absent - dangerous failure | Modify software to detect switch failure and take appropriate action |
| | | | Excessive current | | | |
| 3 | | Excessive switch-bounce | Ageing effects | Slight delay in sensing state of guard | Negligible | Ensure hardware design prevents excessive current through switch |
| | | | Prolonged high currents | | | |

**Figure 3.** An example of FMEA chart [23]

FMEA enables us to identify a system's potential failure modes, their possible causes and the consequences. Each cause of a failure indicates what quality attribute that the system is sensitive from developer's point of view. The corresponding consequences of the failure indicate what quality attributes the system is sensitive from the users' point of view. Both causes and their consequences are observable phenomena of the system. Therefore, the relationships between the quality attributes or quality-carrying properties can be established. However, the original FMEA chart is ambiguous about which component causes the failure. As discussed in the previous sub-section, which component causes the failure is important for quality models of information systems. Therefore, to adapt FMEA for analysing software quality, we modify the FMEA chart to the following format so that the component that causes a failure becomes clear. Another modification to FMEA is that the effects of a failure mode are not charted. There are two reasons for this. First, we found that for a complicated software system, the indirect effects such as those at system level may not be so clear when a component fails. Second, because indirect effects will be analysed subsequently as the effect of other failures, the system level effects of a component failure will eventually emerge from such a chain of cause-effect. The direct effects of a failure mode should have been charted if the direct causes of all failure modes are charted because the 'effect' relation is the inverse of the relation of 'cause'. Finally, we also included an explanation column in the SFMEA chart so that the reasons why a failure mode is caused by another can be Provided.

For example, Figure 4 shows a failure mode that the user cannot find the required information on the Web page. A cause of the failure mode is charted as that the Web page is unable to obtain a file through a hyperlink. Two further causes of the failure are charted: (1) the hyperlink is broken; (2) the Web server is down. (Notice that, there are more causes of the failure than what have been charted in the example.)

| Software FMEA for Web-Based information system | | | | | |
|---|---|---|---|---|---|
| No | Failure mode | | Possible cause | | Explanations |
| | Component | Phenomena | Component | Fault / failure mode | |
| 1 | The user | Cannot find required information | Web page | Unable to obtain a file through a hyperlink | When the user searches for information by browsing through hyperlinks |
| 2 | Web page | Unable to obtain a file through a hyperlink | HTML files | The link is broken | The file cannot be found due to the broken of the link. |
| 3 | | | Web server | Server is down | The file cannot be retrieved and transmitted. |

**Figure 4.** The format of SFMEA chart

In the application of SFMEA, each of the causes and consequences of a failure mode become a new entry to the chart. These causes and consequences are further investigated until the cause is primitive and the consequences are terminal. A failure mode is primitive if it is caused by a fault of a component and its causes cannot be further identified without additional knowledge about the system. A failure mode is terminal if it does not effect any other component of the system or does not cause any other failures. In the example shown in Figure 4, the consequence of 'user cannot find required information' can be considered as terminal. The failure 'broken link' can be considered as primitive. The failure mode that the server is down is neither terminal nor primitive. It should be further investigated for its causes, which might be hackers' attack, maintenance shutdown, system crash due to software failure, etc. The consequences of the failure also need further investigation, which may be more than just that the user cannot find the information.

### 3.2. Construction of quality model

In our method, the construction of a quality model takes the information charted in the SFMEA as input. Each failure mode in the chart forms a node with the component and phenomenon as specified in the SFMEA chart. Each row in the chart forms a link from the node that represents the cause to the node that represents the failure mode. The explanation column of the row forms the reason of the link. For example, for the first row in Figure 4, the following nodes and link are generated.

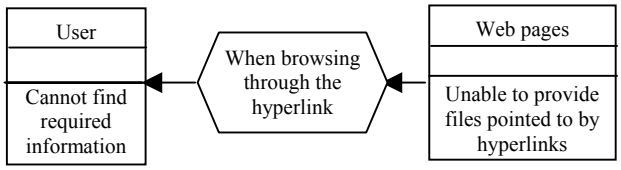


**Figure 5.** The fragment of diagram derived from row 1 of Figure 4

Similarly, from the second and third rows in Figure 4, we can derive the following nodes and links.

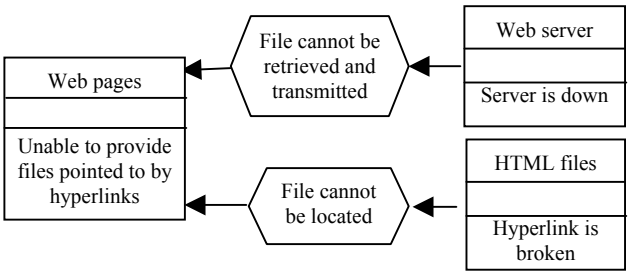


**Figure 6.** Fragment of diagram for row 2~3 of Figure 4.

These nodes and links can be assembled together to obtain a diagram. However, diagrams generated from SFMEA charts as above are incomplete. The property slots need to be filled. Therefore, for each node in the diagram, the observable phenomenon is compared with the definitions of a set of quality attributes and quality-carrying properties of the components. The quality attribute or quality carrying property that the phenomenon demonstrates is, then, identified, or a new attribute or property is recognised. This property is filled into the slot of each node. For example, 'a hyperlink is broken' demonstrates the quality attribute correctness. 'Server is down' is related to the reliability of the system. 'User cannot find required information' is associated to the usability of the system. Therefore, we can derive the following quality model from Figure 4.

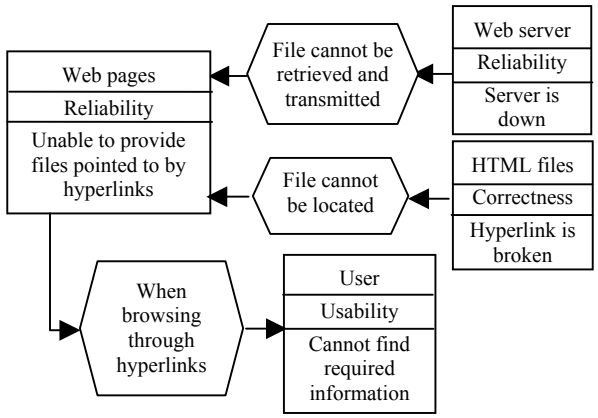


**Figure 7.** The quality model derived from Figure 4.

## 4. Conclusion

In this paper, we further developed the method of hazard analysis based approach to quality modelling of information systems proposed in [18, 19]. The main contribution of the paper is two-fold. First, a diagrammatic notation is proposed to represent quality models of information systems. It enables the explicit references to the components of the information system whose quality-carrying properties affect the system quality attribute. It also enables the explicit annotation of the reasons why two properties or attributes are related. Containing such information in quality models can significantly improve the usability of quality models in information system development. Second, the failure mode and effect analysis method originally developed for hazard analysis of safety critical systems is adapted for the analysis of software and information systems. The result of this adapted method can be directly used to construct quality models of information systems. It provides a logic that bridges the gap between abstract system quality attributes and the tangible quality-carrying properties of components and the observable behaviour of the system and their components.

There are also a number of other advantages of the proposed method. First, it enables software engineers to derive quality models at an earlier stage of software

development in comparison with similar methods such as the card sorting method [17]. This is particularly important because the awareness of a sensitive quality attribute at early stage such as at design stage enables software engineers to seek for technique solutions to achieve the required quality standard.

Second, deriving quality models at architectural level enables software engineers to understand the quality of a type of software systems in a particular application domain and of the same architectural features. The results of such quality modelling have a wide applicability. Yet, when more details of the structure and functions of the components are provided, more details of the quality model can be obtained, and thus, provide more insight information for follow up development activity.

Finally, the quality models constructed by our method include not only the abstract properties and attributes but also their observable phenomena of the components of the system and the rationale of the links between the phenomena. The representation of models proposed in this paper is not only more expressive than existing ones, but also makes software quality models more testable and verifiable.

A preliminary empirical study of the method has been carried out to develop quality models of several types of web-based information systems. The result seems very promising. There are a number of directions for further work. For example, how to identify failure modes systematically for each component needs further investigation. It seems that the HAZOP technique can be adapted for this purpose and integrated into the process of modelling proposed in this paper. Moreover, quality attributes are of different importance in different application systems. How to assign weights to quality attributes needs further investigation. We can also learn from the methods and techniques of hazard analysis where the safety and risks of a system are quantitatively analysed according to the consequences of a hazard and its probability of occurrences.

## References


[1] Kitchenham, B. and Pfleeger, S. L., "Software Quality: The Elusive Target", *IEEE Software*, Vol. 13, No. 1, Jan. 1996, pp12-21.

[2] Gillies, A., *Software Quality: Theory and Management*, 2nd Edition. International Thomson Computer Press, 1997.

[3] McCall, J., Richards, P. and Walters, G., "Factors in Software Quality", *Technical Report CDRL A003*, US Rome Air Development Centre, Vol. I, 1977.

[4] Boehm, B.W., Brown, J., Kaspar, H., Lipow, M., MacLeod, G. and Merrit, M., *Characteristics of Software Quality*. TRW Serious of Software Technology, Vol. 1, North-Holland, New York, 1978.

[5] ISO 9126, *Information Technology -- Software Product Evaluation -- Quality Characteristics and Guidelines for Their Use*, International Organisation for Standardization, Geneva, 1992.

[6] Eriksson, I. and Torn, A., "A Model for IS Quality", *Software Engineering Journal*, July 1991, pp152-158.

[7] Olsina, L., Godoy, D., Lafuente, G.J. and Rossi, G., "Specifying Quality Characteristics and Attributes for Websites", in *Proceedings of the First ICSE Workshop on web Engineering*, pp16-17 May 1999, Los Angeles, USA.

[8] Perry, W.E., *Quality Assurance for Information systems: Methods, Tools and Techniques*. New York: John Wiley & Sons, 1991.

[9] Gillies, A., "Modelling Software Quality in The Commercial Environment". *Software Quality Journal*, Vol. 1, 1992, pp175-191

[10] Fenton, N.E., *Software Metrics -- A Rigorous Approach*, Chapman & Hall, 1991.

[11] Shepperd, M., *Foundations of Software Measurement*, Prentice Hall, 1995.

[12] Kitchenham, B. A. and Walker, J. G., "A Quantitative Approach to Monitoring Software Development", *Software Engineering Journal*, Vol. 4 , No.1, 1989, pp2-13.

[13] Kitchenham, B.A. and Pickard, L.M., "Towards a Constructive Quality Model", *Software Engineering Journal*, Vol. 2, No.4, 1987, pp114-126.

[14] Dromey, R. G., "A Model for Software Product Quality", *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, Feb. 1995, pp146~162

[15] Dromey, R. G., "Cornering the Chimera", *IEEE Software*, Vol. 13, No. 1, Jan. 1996, pp33~43.

[16] Bansiya, J. and Davis, C. G., "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, Jan. 2002, pp4-17.

[17] Upchurch, L., Rugg, G. and Kitchenham, B. "Using Card Sorts to Elicit Web Page Quality Attributes", *IEEE Software*, July/Aug, 2001, pp84-85.

[18] Zhang, Y., Zhu, H., Greenwood, S., and Huo, Q., "Deriving Quality Models of Web-based Information Systems", In *Proc. of SQE'2002*, Feb, 2002, Banff, Canada.

[19] Zhang, Y., Zhu. H., Greenwood, S. and Huo, Q., "Quality Modelling of Web-based Information Systems", in *Proc. of the IEEE Workshop on FTDCS*, Italy, Oct 30-Nov 2 , 2001, pp41-47.

[20] Kletz, T., *Computer control and Human Error*, Rugby, Institute of Chemical Engineers, 1995.

[21] Leveson, N. G., *Safeware: System Safety and Comuters*, Reading, MAL Addison-Wesley, 1995.

[22] Neumann, P. G., *Computer-Related Risks*, ACM Press, New York, 1995.

[23] Storey, N. *Safety-Critical Computer Systems*, Reading, MA, Addison, 1996.