An Approach of Automatically Performing Fault Tree Analysis and Failure Mode and Effect Techniques to Software Processes

Danhua Wang
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
wangdanhua6004@hotmail.com

Jingui Pan
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
panjg@cn.fujitsu.com

***Abstract*—In practice, engineers find that software quality depends heavily on the maturity and reliability of the software process. Therefore, organizations are placing increased attentions on finding an efficient way of integrating elements of software process in order to produce high quality software with lower cost and risk. Various kinds of techniques are used to manage, monitor and analyze software process. We proposed an idea of modeling software process in Little-JIL language, and then introduce automatic Fault Tree Analysis (FTA) technique and Failure Mode and Effect Analysis (FMEA) technique, two fully-fledged and widely used safety analysis techniques, to analyze software process. Results of these two techniques can be combined to improve software process, which may lead to software products with higher quality and reliability, also lower cost and risk.**

Keywords-FTA; FMEA; software process; automatic

I. INTRODUCTION

Software process is a series of ordered activities developing and maintaining software products. Each activity indicates its artifacts, resources, agents, architectures, and so on [1, 2]. That is to say, software process is the way of producing software, and it's a complicated system consisting of various essential elements which are related to software production [3]. In practice, more and more organizations realize that quality of software products depends heavily on the software process they use [1, 4, 5]. Software process management is necessary to high quality software products [6, 7]. Organizations are placing increased attentions on finding an efficient way of integrating elements of software process in order to produce high quality software with lower cost and risk [8]. Various kinds of techniques can be used to manage, monitor and analyze software process. Fault Tree Analysis (FTA) technique and Failure Mode and Effect Analysis (FMEA) technique, two widely accepted and fully-fledged safety analysis techniques, can be used to detect weaknesses and safety problems in software process. Before applying these two techniques to analyze software process, there is still one issue to be addressed, software process should be modeled by languages with enough precise semantics. Little-JIL process definition language is a language which satisfies this requirement. It is an executable, high-level process programming language with a formal (yet graphical) syntax and rigorously defined operational semantics [9]. It provides a process modeling method basing on activities, which are defined as steps in Little-JIL processes. This paper puts forward to an approach of applying automatic FTA and FMEA techniques to analyze software process after modeling it using Little-JIL language, and combine analysis results to help improve software process.

The rest of this paper is organized as follows. Section II provides a brief introduction of the Little-JIL process definition language and how to model software process in Little-JIL process language. Section III gives a description of performing FTA technique and FMEA technique to software process, and also how to combine the analysis results of both FMEA technique and FTA technique to help improve software process. Section IV presents related works. The final section provides conclusions and suggests future work.

II. MODELING SOFTWARE PROCESS USING LITTLE-JIL PROCESS DEFINITION LANGUAGE

A. Introduction of Little-JIL Process Definition Language

A Little-JIL process (process specified using Little-JIL language) is a hierarchy of steps, each of which represents a single unit of task. Step is the basic element of Little-JIL process. Each step specifies all parameters and resources it uses in its interface. Parameters are passed between different steps via parameter bindings. According to different parameter flow directions and scopes, there are four parameter types, In, Out, In/Out and Locals. Little-JIL process assigns each step to an execution agent and coordinates execution sequence of steps. Step without any sub-steps is called leaf step. Each non-leaf step has a sequencing badge which indicates the executing order of its sub-steps. There are four types of step sequencing, a "sequential" step executes its sub-steps from left to right one by one and is completed when all of its sub-steps have been completed; a "parallel" step posts its sub-steps concurrently and is completed when all of its sub-steps have been completed; a "try" step executes its sub-steps from left to right one by one and is completed when one of its sub-steps has been completed; a "choice" step tries to choose one of its sub-step to be executed and is completed until one of its sub-steps has been completed. A step can optionally be proceeded by one or several pre-requisite step(s) or/and be followed by one

Sponsor: State Key Laboratory for Novel Software Technology at Nanjing University.
Project Number: KFKT2009A13

or several post-requisite step(s). In Little-JIL language, a resource is any entity for which there is contention for access [10], and agent is defined as a special kind of resource. Resources in Little-JIL language are managed by an external resource manager and their acquisitions need to be explicitly specified in step interfaces. An artifact produced by one step may be a resource to other steps. Resources can be passed between steps through parameter bindings just like parameter passing. Steps in Little-JIL language may throw exceptions, which can be handled by exception handler steps. There is a mechanism named "cardinality" for expressing the optionality or repetition of a step [10]. The number of instances of a step that should be created may be specified statically within a process, be determined by the agent assigned to the steps, or process programmers may indicate that step instances should be created based on the availability of artifacts or resources [10]. Space limitations prevent describing all the features of Little-JIL language. We only give a brief introduction of semantics of Little-JIL language here. Details of the language can be found in [10].

B. Modeling Software Processes Using Little-JIL Process Definition Language

Different software organizations apply different software processes. However, once the organization determines the elements of its software process, Little-JIL language can be used to model the software process in the following way. Fig. 1 shows a simple software process modeled in Little-JIL language.

1) Activity: The activities and their sub-activities are represented as steps and their sub-steps in Little-JIL language. We take the simple software process in Fig. 1 as an example. In our software process model, "Requirement Analyze", "Plan", "System Design", and so on, are activities in software process. They are represented as steps and they can be decomposed into sub-steps as far as necessary to describe them. The execution order of sub-steps depends on step sequencing of their parent step. "Modular Design" should select one of its sub-steps, either "Object-Oriented Design" or "Procedure-Oriented Design", to be executed. If the sub-step chosen to be executed succeeds, step "Modular Design" is completed. Activities in software process may be proceeded or followed by other requisite activities. For example, "Requirement Review" is the post-requisite step of "Requirement Analyze". Steps in Little-JIL language may throw exceptions, which can be captured and handled by exception handler steps. For example, step "System Design" may throw exception "e2: requirement inconsistency" and exception "e3: requirement vulnerability". There is a "+" displayed adjacent to where the connector is attached to the sub-step "Modular Requirement Analysis", which indicates that the sub-step should be executed at least one time and the executing time is optional. In our example model, the executing number of times depends on the number of modules.

2) Artifact: All products produced in software process are artifacts, such as design document, code, test cases, and so on. They are represented as parameters specified in the step interface. They are passed between steps via parameter bindings.

3) Resource: Resources in software process are specified in the step interface. For example, computers, development tools, and so on, are all resources. Agents such as engineers, programmers, tester, are treated as a special kind of resources in Little-JIL language. Because we haven't taken the effects of resources into consideration in our approach, there are no resource instances in our example process model.

4) Module: Little-JIL language allows multiple modules existing in Little-JIL process. Software process can be decomposed into smaller modules as far as necessary to model it.

III AUTOMATIC FTA AND FMEA TECHNIQUES TO SOFTWARE PROCESSES

After modeling software process using Little-JIL language, various kinds of techniques can be used to manage, monitor or analyze them. FTA technique and FMEA technique, two fully-fledged and widely used safety analysis techniques, can be used to analyze software process specified using Little-JIL language, discover defects in software process, and then recommended actions can be proposed to improve the processes. Table 1 made a comparison between traditional FTA technique and FMEA technique. Both FTA technique and FMEA technique provide the engineers with tools that can assist in providing reliable, safe and customer-pleasing software products. With increasing maturity, FTA technique and FMEA technique are widely accepted and applied to complex systems or processes in various industries especially safety critical industries such as health care industry. We notice that customers are placing increased demands on software companies for high quality, reliable software. Also, as we mentioned before, software quality depends heavily on their production procedure, that is software process. Therefore, in our approach, we apply FTA technique and FMEA technique to analyze software process.

A. Automatic FTA Technique to Software Processes

FTA technique is a structured top-down deductive analysis technique [11], which involves two steps: deriving the fault tree and analyzing the fault tree. It can systematically identify and evaluate all possible events that could lead to a given hazard. Traditionally, the step of deriving fault trees is very time-consuming and error-prone because it is performed manually by a group of experts. In order to overcome these shortcomings, we introduce an automatic FTA technique [12] which can automatically derive fault trees from processes modeled in Little-JIL language. By performing qualitative and quantitative analysis to fault trees, it can help identify all potential causes of the given hazards. In order to automate the procedure of deriving fault trees from Little-JIL processes, there are two issues need to be addressed: how to automatically extracted fault tree events from a Little-JIL process, and how to identify all immediate and necessary events that could lead to a given event and connected them to this event using appropriate gates. For the first issue, a few

TABLE I. Comparison between Traditional FTA technique and FMEA technique

	FTA	FMEA
Direction of analysis	Top-down (Deductive)	Bottom-up (Inductive)
Analyze time	Reactive	Proactive
Focus of analysis	Focusing on effects of a given hazard	Focusing on causes of a given hazard
Presentation of analysis	Logic diagram	Table

types of events are predefined and they can be easily identified from the process definition. To address the second issue, a collection of templates are defined based on the Little-JIL process definition. Different templates are used to develop different types of events. Given a failure mode or hazard "Artifact 'Software Product' to 'Customer Test' is wrong", Fig. 2 shows the fault tree automatically derived for it. Also, Minimal Cut Sets (MCSs) of the fault tree will also be computed.

B. *Automatic FMEA Technique to Software Processes*

FMEA technique is a systematic bottom-up inductive method of analyzing and evaluating safety problems in a system or process in order to avoid server hazards or consequences. FMEA technique essentially consists of identifying and listing all potential failure modes, accessing effects on the overall system for each failure mode, and then identifying all potential causes which may lead to each failure mode. We propose an automatic FMEA technique to analyze software process. Our approach involves four steps: define the failure mode, identify potential failure mode, identify effects for each failure mode, and indentify causes for a given failure mode. In our approach, we focus on the first three steps. We introduce automatic FTA technique described in Section III.A. to the final step, indentify causes for a given failure mode. In our work, we limit our attention to the failure modes which are related to artifacts. We believe that a large number of interesting failure modes are artifact-related failure modes or can be easily turned into artifact-related failure modes. For example, in many real world software processes, some hazards are caused by the delay of certain steps. To capture such faults, we can associate an artifact representing the execution time to each step. Our approach then can handle the delays just like the other artifact faults. Therefore, we predefine two types of artifact-related failure modes:

1) Type 1: Artifact p to Step S is wrong; (p is an "In" parameter in Step S)

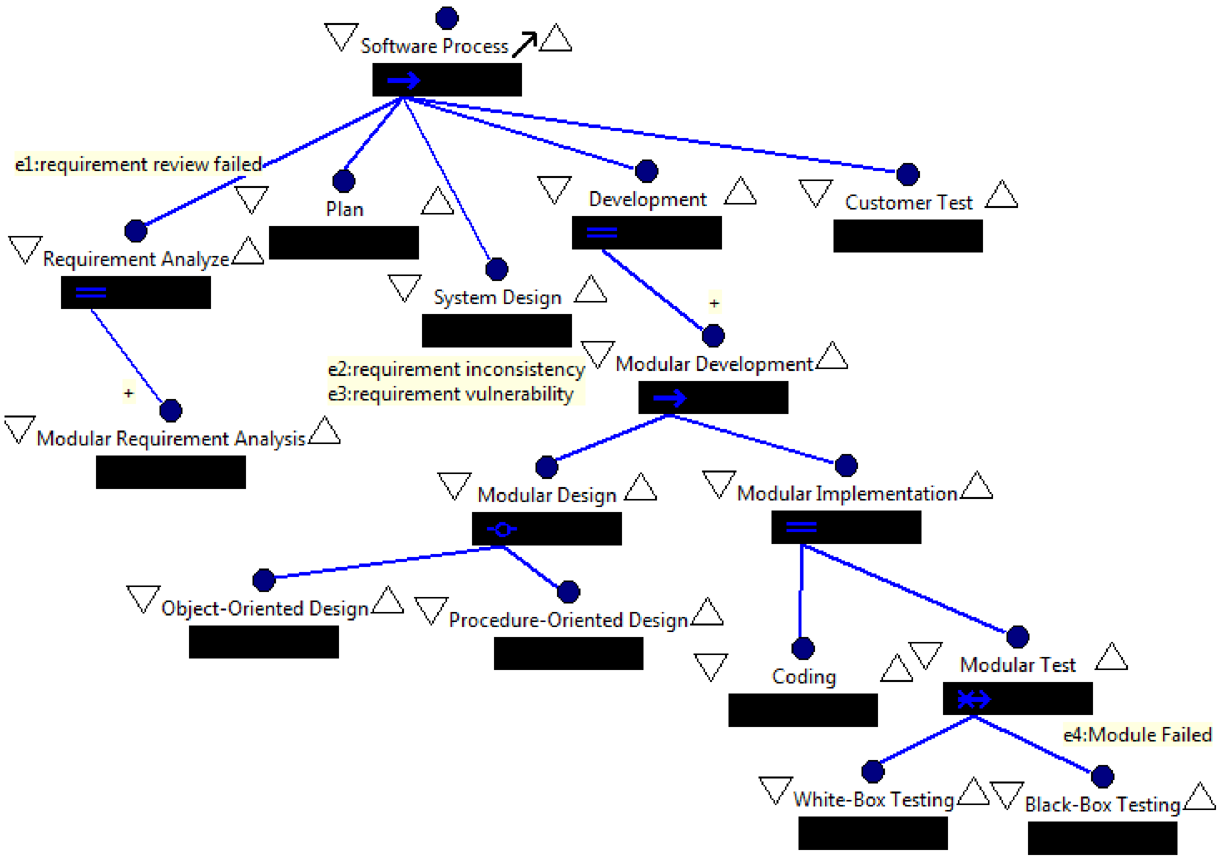


Figure 1. Simple Software Process Modeling in Little-JIL Language

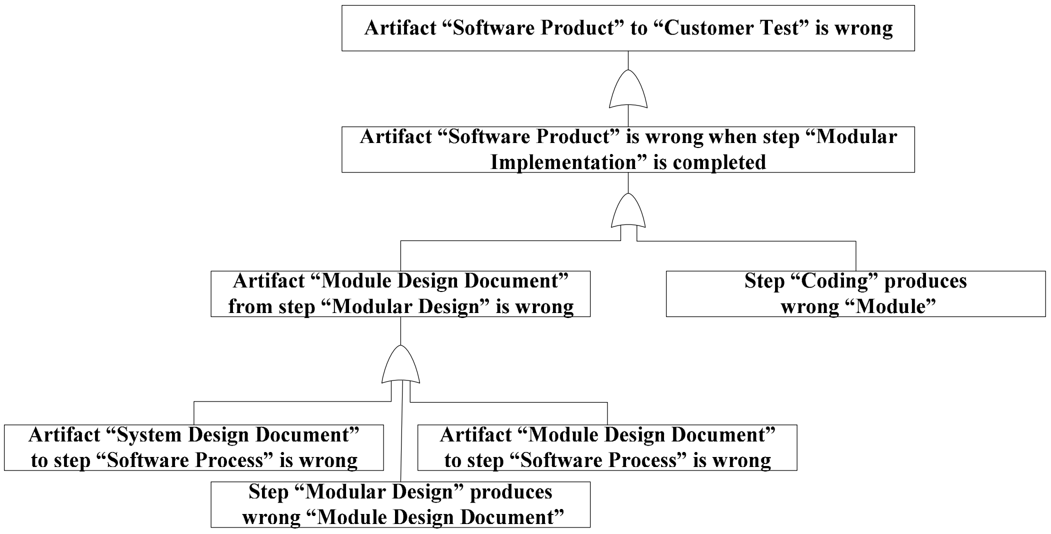
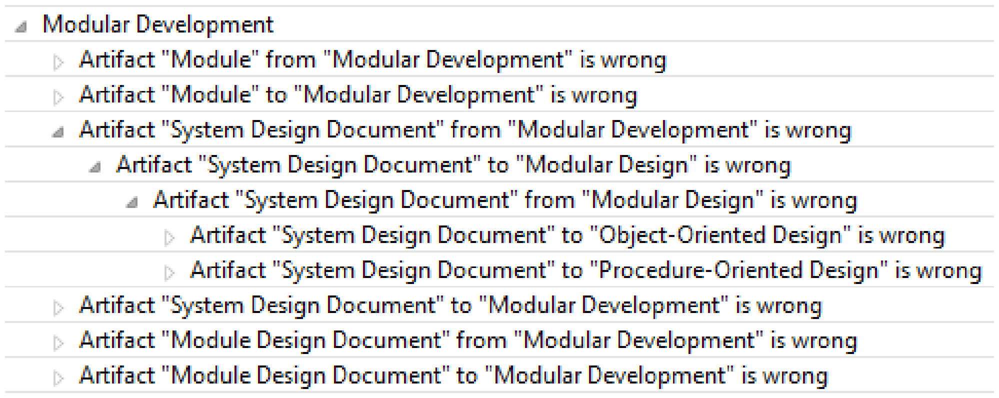

Artifact “Software Product” to “Customer Test” is wrong
Artifact “Software Product” is wrong when step “Modular Implementation” is completed
Artifact “Module Design Document” from step “Modular Design” is wrong
Step “Coding” produces wrong “Module”
Artifact “System Design Document” to step “Software Process” is wrong
Artifact “Module Design Document” to step “Software Process” is wrong
Step “Modular Design” produces wrong “Module Design Document”

Figure 2. Fault Tree Automatically Derived for a Given Hazard



Modular Development
- Artifact "Module" from "Modular Development" is wrong
- Artifact "Module" to "Modular Development" is wrong
- Artifact "System Design Document" from "Modular Development" is wrong
  - Artifact "System Design Document" to "Modular Design" is wrong
    - Artifact "System Design Document" from "Modular Design" is wrong
      - Artifact "System Design Document" to "Object-Oriented Design" is wrong
      - Artifact "System Design Document" to "Procedure-Oriented Design" is wrong
- Artifact "System Design Document" to "Modular Development" is wrong
- Artifact "Module Design Document" from "Modular Development" is wrong
- Artifact "Module Design Document" to "Modular Development" is wrong

Figure3. A Small Piece of FMEA Tree View

2) *Type 2:* Artifact p from Step S is wrong; (p is an “Out” parameter in Step S)

As we mentioned in Section II.A., there are four types of parameters. For any “In” parameter p of step S, a failure mode of “Type 1” is created; for any “Out” parameter p of step S, a failure mode of “Type 2” is created; for any “In/Out” parameter p of step S, failure modes of both “Type 1” and “Type 2” are created; “Locals” parameters are used the same as other three types of parameters, just with a limited scope, therefore, failures modes are created for them just in the same way as “In”, “Out” or “In/Out” parameters.

In order to generate effects for each failure mode, Artifact Flow Graph (AFG) is abstracted from software process. It includes all data dependence relationships among artifacts in software process. Given a failure mode “Artifact p to Step S is wrong” (Type 1) or “Artifact p from Step S is wrong” (Type 2), we can decide all artifacts which p can flow to by traversing the AFG. Faults in p many be propagated to these artifacts. Therefore, faults of these artifacts are defined as effects of the given failure mode. We organize the result as an effect tree. Fig. 3 presents a small piece of FMEA tree view analyzed for the software process showed in Fig. 1. The root level is the step in the software process. The first level is the failure modes in that step, and other levels lists effects for each failure mode. The effect tree can be expanded to different levels until reaching final effects of each failure mode.

C. *Combination of FTA technique with FMEA technique*

The effect we defined is a failure mode too. Also, both the effect and the failure mode we predefined are events which can be used as top-events in the automatic FTA technique we described in III.A. Fault trees (potential causes) can be automatically generated for a given failure mode or effect.

Performing FTA for all potential failure modes and effects in a software process might be a huge job and sometimes a waste of time and energy. Also, not every failure leads to disastrous consequences. The failure modes or effects which may cause hazards should be paid more attention to than others. We suggest that after generating all potential failure modes and effects in a software process, the results should be examined carefully. If the failure mode or effect is critical, fault tree need to be generated to find out all potential causes of it, and then actions can be recommended to avoid the hazard. Also subsequent changes can be made to the software process to remove its weakness. In this way, it realizes the combination of FTA technique and FMEA technique and we can use the analysis results of these two techniques to improve the software process. It can prevent hazards before the software process comes into use and result in software products with high quality and reliability.

IV RELATED WORKS

Most failure analyses and studies are based on either FTA technique or FMEA technique. Rarely, both FTA and FMEA techniques will be performed, and when both are performed, these will be separate activities executed one after another – without significant intertwining [16]. However, there still exist several approaches for combining these two techniques together for safety analysis. Bettina proposed an approach of combining manual FTA and FMEA technique for software safety analysis [15]. In that approach, FTA technique provides selection criteria for FMEA technique, and FMEA technique provides feedbacks on completeness of FTA technique. Ref.[16] realized an innovative combination of FMEA and FTA techniques, which is able to maximize the advantages and at the same time to minimize the shortcomings of both known methodologies. IBM puts forwards to combining FTA technique with FMEA technique by performing FMEA technique first to identify potential failure modes in software development life cycle, and then using FTA technique to discover these causes of those failures [17]. We prefer applying these two safety analysis techniques to automatically analyze software processes.

V Conclusions and Future Work

Nowadays, customers are placing increased demands on software organizations for high quality, reliable products with low cost and risk. In practice, engineers find that software quality depends heavily on the maturity and reliability of the software process. They come up with modeling software process in languages with enough precise semantics. Thanks to this, software development becomes predictable and controllable. Various kinds of techniques can be used to manage, monitor or analyze software process. We propose an idea of modeling software process using Little-JIL language, and then introduce FTA and FMEA, two fully-fledged and widely used safety analysis techniques, to automatically analyze software process. Automatically performing FMEA technique and FTA technique to software processes modeled in Little-JIL process definition language addresses the major weaknesses of traditional FMEA and FTA approach – traditional approach requires good understanding of processes and is time-consuming. Analysis results can be combined to improve software process, which may lead to software products with higher quality and reliability, also lower cost and risk.

Software process is a complicated system compounded by various kinds of elements. Effects of FTA and FMEA techniques are closely related with the completeness of our software process model specified using Little-JIL language. Understanding of software process and semantics of Little-JIL language can have a strong impact on software process modeling. How to completely model real-world software process is one part of our future work. Also, the execution of software process is heavily influenced by the availability of people and materials [10], which are defined in the Little-JIL step interface as resources. In our work, we haven't set up dependence relationships between resources and artifacts. The next work we have to do is to take the influence of resources into account.

One more limitation is that subsequent steps could be dependent on a previous step being done correctly. If we create a hypothetical output artifact to represent the erroneous step behavior, this artifact would not propagate erroneous information beyond this step. Therefore, we have to find a way to add fault propagations introduced by erroneous step behavior to generate the FMEA information.

ACKNOWLEDGMENT

We would like to thanks B. Chen, G.S. Avrunin and L.A. Clarke for their many helpful suggestions with this work.

REFERENCES

[1] M.S. Li, Q.S. Yang, J. Zhai, "Systematic review of software process modeling and analysis." Journal of Software. Vol 20(3), pp.524-545, 2009.

[2] J. Lonchamp, "A structured conceptual and terminological framework for software process engineering." ICSP. pp.41-53, 1993.

[3] Stephen R. Schach. "Software Engineering with Java." China Machine Press, Beijing, 1999.

[4] C. Montangero, J.C. Derniame, B.A. Kaba, B. Warboys, "The software process: modelling and technology." Proc. of the Software Process: Principles, Methodology, Technology. Springer-Verlag, pp.1-14, 1999.

[5] SEI, CMU, "CMMI for Development," 2nd , Improving Processes for Better Products", 2006.

[6] F.Q. Yang, "Thinking on the development of software engineering technology." Journal of Software. Vol 16(1), pp.1-7, 2005.

[7] J.D. Gu, Q,Hu, L.V. H, "A Multi-view software process model based on object petri nets." Journal of Software. Vol 19(6), pp.1363-1378, 2008.

[8] Y. Peng, G. Kou, G. Wang, H. Wang, F. Ko, "Empirical Evaluation Of Classifiers For Software Risk Management," International Journal of Information Technology and Decision Making, Vol. 8, pp.749-768, 2009.

[9] http://laser.cs.umass.edu/tools/littlejil.shtml

[10] S.W. Alaxander, "Little-JIL 1.5 Language Report." Department of Computer Science, University of Massachusetts, Amherst, 2006. Unpubilshed.

[11] W.E. Vesely, "Fault Tree Handbook with Aerospace Applications," NASA Headquarters, Washington, D.C., pp.300-322, 2002.

[12] B. Chen, G.S. Avrunin, L.A. Clarke, L.J. Osterweil, "Automatic fault tree derivation from Little-JIL process definitions." Proc. of the Int'l Software Process Workshop and Int'l Workshop on Software Process Simulation and Modeling. Springer-Verlag, pp.150−158, 2006.

[13] Department of Defence, "Procedures for Performing a Failure Mode, Effects and Criticality Analysis." Washington, 1980.

[14] G. Nancy, S. Leveson, "Safeware: System Safety and Computers." Published by Addison-Wesley.

[15] www.rvs.unibielefeld.de/Bieleschweig/ninth/ButhB9Slides.pdf

[16] Z. Bluvband, R. Polak, P. Grabov, "Bouncing failure analysis (BFA): The Unified FTA-FMEA methodology." Annual Reliability and Maintainability Symposium, 2005.

[17] M. McDonald, R. Musson, R. Smith, "The practical guide to defect prevention," Microsoft press, 2008.