A Case Study using a Protocol to Derive Safety Functional Requirements from Fault Tree Analysis

Luiz Eduardo Galvão Martins
Institute of Science and Technology
Federal University of São Paulo, UNIFESP
São José dos Campos, Brazil
legmartins@unifesp.br

Tiago de Oliveira
Institute of Science and Technology
Federal University of São Paulo, UNIFESP
São José dos Campos, Brazil
tiago.oliveira@unifesp.br

***Abstract*— State-of-the-art in Requirements Engineering offers many frameworks and techniques to enable requirements engineers in their work. However, for critical systems there are gaps in state-of-the-art, and these can result in dire consequences, potentially putting lives in danger and damage infrastructure and threaten the environment. A well known technique used to help requirements engineers to understand safety hazards situations in the context of safety-critical software is Fault Tree Analysis (FTA). This technique is a good one to decompose hazards identified in the system context into events that may put the system functionalities in risk. However, FTA does not offer a protocol of how to derive safety functional requirements from fault trees. In this paper we present a case study adopting a protocol to help requirements engineers to derive safety functional requirements from FTA. The proposed protocol was based on a study performed in a Brazilian company in the area of electronic medical devices. The development of prototype of a low cost insulin infusion pump, which is a critical system, offered the basis to propose and test a protocol to derive safety functional requirements from FTA. During the case study we collected evidences that help us to discuss if FTA is sufficient to guide software engineers to implement the corresponding control software and also if FTA offers enough information to help requirements engineers to derive safety functional requirements.**

***Index Terms*— Safety Functional Requirements, Fault Tree Analysis, Critical Systems, Hazard Situations, Embedded Systems.**

I. INTRODUCTION

Modern society is becoming more and more dependent on software. In the last decades there were important advances in computing technologies, and software assumed a central role in the control of equipments, devices and processes based on computers. On a daily basis, the common citizen is dependent on several services supported by software, and many services are transparent to the users. For example, a car contains millions lines of code, that control everything from the A/C unit to safety critical aspects such as breaking. Other examples range from banking services available in automated teller machines or web systems; services available in pay-per-view TVs; traffic-air control; road monitoring; telecommunication system control; medical equipments control, since a blood pressure monitor device until sophisticated magnetic resonance image equipments; automotive control systems; entertainment systems; to surveillance systems based on computer vision; and others [2][3].

Research in Software Engineering is related to the conception, design, development and evolution of software-intensive systems [4]. Furthermore, Software Engineering is becoming specialized in many fields, considering the scope and the complexity of several aspects concerned to software development, such as Requirements Engineering, which is also considered a field of Systems Engineering.

Requirements Engineering focuses on the development of processes, techniques, methods, models and tools to help in the conception of software and systems, covering the activities of elicitation, analysis, specification, validation and management of requirements [4]. Requirements are the base to further software design, but besides this it offers a description of the functional and non-functional aspects that must guide the implementation and evolution of the software, as well as the verification and validation activities central for safety and security.

The software development teams are concerned with the development of systems that address user needs. The correct identification of such needs is not a trivial task, and usually there are many issues to elicit and define the relevant software requirements. Software-intensive systems are becoming ubiquitous in daily life, making people highly dependent on them. Thus, systems must offer a high level of safety, security, reliability and robustness [5][11]. However, despite the efforts of development teams, the level of customer satisfaction and quality are often lacking [1][4]. This is both due to inadequacies in the ability to elicit and select the central and most important requirements to realize in the product, but also due to the fact that requirements understanding leads to miscommunication within the development companies.

State-of-the-art in Requirements Engineering offers many frameworks and techniques [4][12][14] to enable requirements engineers in their work. However, for critical systems there are gaps in state-of-the-art, and these can result in dire consequences, potentially putting lives in danger and damage infrastructure and threaten the environment [1][6][9].

A well known technique used to help requirements engineers understand hazards situations in the context of safety-critical software is Fault Tree Analysis (FTA) [18]. This technique is a good one to decompose hazards identified in the system context into events that may put the system functionalities in risk. However, FTA does not offer a protocol of how to derive safety functional requirements from fault trees. Despite the fault trees are useful to represent events that put systems in hazardous states they do not describe safety functional requirements and only the graphics representation supported by FTA seems to be insufficient to document safety requirements.

In this paper we present a case study adopting a protocol to help requirements engineers to derive safety functional requirements from FTA. This protocol was performed in a Brazilian company in the area of electronic medical devices to derive requirements in styles "should" and "should not", which helped software engineers understand the safety functional requirements to be implemented in the control software of an insulin infusion pump. During the case study we collected evidences that help us to discuss if FTA is sufficient to guide software engineers to implement the corresponding control software and also if FTA offers enough information to help requirements engineers to derive safety functional requirements

The motivation to the Brazilian company to develop a low cost insulin infusion pump is that until now there are no companies in Brazil developing such device. For this reason, diabetes patients in Brazil have to import insulin infusion pump whose price is about 5,000 USD including import tax, which is very expansive for the most of the Brazilian people.

This paper is organized as follows: section 2 presents background and related works in safety-critical software; section 3 outlines research methodology to conduct the case study based on development of a low cost insulin infusion pump in a Brazilian company; section 4 presents the results and analysis of the performed case study; and section 5 concludes the paper and points out to future works.

II. Background and Related Works

The main characteristic of safety-critical software, that distinguishes it from others, is that error or failure occurrence is potentially dangerous to people, properties and environment, with potential to cause accidents. For this reason, this kind of software demands strict management of safety aspects [14][16]. Table I shows complementary characteristics of safety-critical software, as well as the gaps in current requirements approaches.

According to Firesmith [1] most requirements specifications produced in industrial setting are incomplete and do not properly address hazard situations for safety-critical systems. Requirements specifications are particularly incomplete to consider exceptional situations that combine rare events sequences, culminating in unacceptable risks [19][20]. Even when requirements are specified for rare combinations of events, they may be ambiguous, incorrect or inconsistently implemented [6][10]. Therefore, safety related requirements must be carefully specified, demanding more sophisticated Requirements Engineering approaches. Fig. 1 presents a high level model showing the main system conditions taking into account the safety viewpoint in the development of safety-critical systems. The state machine highlights three main transitions to systems with potential to cause accidents. Hazard conditions are the central issue in this model. Capturing and specify requirements that help system engineers deeply understand all possible system hazard conditions is probably the main contribution that Requirements Engineering approaches can offer in the domain of Safety Engineering. For such intention a complete specification of the sequences of risk events, hazard mitigation events and catastrophic events must be done.

TABLE I. Complementary Characteristics of Safety-Critical Software and Gaps in Current Requirements Approaches.

Characteristics of Safety-Critical Software	**Gaps in Current Requirements Approaches**
High integrity level.	The majority of software safety standards adopt an integrity level approach, categorizing software in terms of its criticality [12]. Current requirements approaches do not support integrity compliance properly [17].
High complexity system.	Safety-critical software usually works in high complexity systems. As discussed in [13], abstraction layers facilitate reasoning about system characteristics, but existing requirements approaches are insufficient for handling requirements for critical systems.
High associated development cost.	Current software safety standards assume that safety is proportional to cost [12], that is because critical systems need more process and documentation. This is justified by the necessity of assurance activities during all the process. Requirements approaches that emphasize design simplicity can help to cost reducing. Current requirements approaches do not address simplicity as a key feature.
Risk tolerance.	Establishing risk tolerance and acceptance is a fundamental issue during safety requirements definition. Despite the safety standards give high attention for that, current requirements approaches do not properly address such issue.

Some hazard assessment approaches have been proposed in the context of Safety Engineering. The most common are Checklists, Failure Modes and Effects Analysis (FMEA), and Fault Tree Analysis (FTA) [18]. The Checklist approach is easy to use and very common, and can be used at all phases of system project. However, Checklists are difficult to compile and easy to misuse. No matter how much effort is dedicated to create a Checklist, there is no guarantee that it is complete.

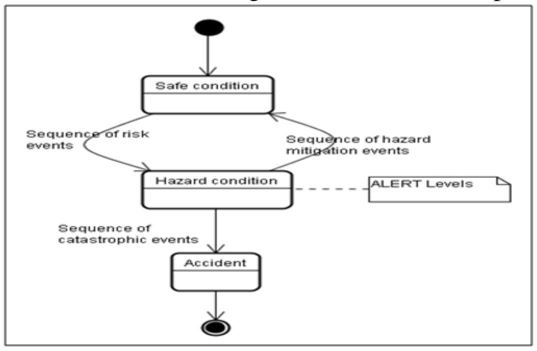

Fig. 1. High level model of system conditions from the safety viewpoint.

FMEA was originally designed for reliability issues, the goal of this approach is to analyze the possible failure modes of a component and to identify the consequences of such failures. FTA offers a structure that helps requirements engineers to analyze risk events with the intention to determine the environmental conditions under which a state system becomes unsafe. However, for FTA to become useful, a small number of critical failures must be chosen which can be distinguished from a large number of possible failures by their catastrophic consequences [18].

In [19][20] Lutz and Mikulski discuss the importance to identify rare events in safety-critical systems. They present seven cases where rare events contributed to put spacecraft's missions in serious risks. As mentioned in their works, rare events usually have the potential to cause catastrophic risks for critical system. For this reason, effort to identify rare events is strongly advised. However, the current requirements approaches are not adequate for such intention.

III. Research Methodology

The case study presented in this paper was performed during the initial phase of a project to develop a prototype of a low cost insulin infusion pump. That project has been developed in cooperation with a Brazilian company called *DeltaLife* (www.deltalife.com.br), which is a company in the area of electronic medical devices. It was identified a gap in Brazilian market that motivated industry and academia to join forces for such development.

Participated in that study six people: one mechanical engineer – with more than 5 years of industry experience; one electronic engineer – with more than 10 years of industry experience; one requirements engineer – with more than 10 years of experience in industry and academia; two software engineers with two years of industry experience; and one doctor with more than five years of experience in diabetes treatment using insulin infusion pump. Both mechanical and electronic engineers had high experience in embedded systems development. The requirements engineers and software engineers had few experience in embedded systems.

The first step of that study was to identify the main hazard situations concerned to insulin infusion pump. The sources to identify the hazards involving insulin infusion pump were: (i) literature review [7][8]; (ii) the doctor experience in the diabetes treatment using insulin infusion pump; and (iii) the mechanical and electronic engineers experience in the embedded systems development for electronic medical devices.

It was identified ten relevant hazard situations that could put the user in danger. The main hazard situation identified was "insulin overdose" that can lead the user to death. Considering that "insulin overdose" is the most severe hazard situation it was chosen to test the adopted protocol to derive safety functional requirements from FTA.

FTA is a well known technique used to analyze hazard situations in safety-critical systems, it is a technique very used in safety engineering community and it is being adopted by requirements engineers to help in the safety requirements capturing. In that study FTA was used to decompose "insulin overdose" into risk events as presented in Fig. 2.

Considering that is becoming common requirements engineers to adopt FTA as technique to analyze hazard situations in safety-critical systems, two research questions (RQ) were investigated in the conducted case study:

- *RQ1 – Does FTA offer enough information to software engineers implement safety requirements in the corresponding control software?*

- *RQ2 – Is FTA a good enough representation to derive safety functional requirements which describe system hazard situations?*

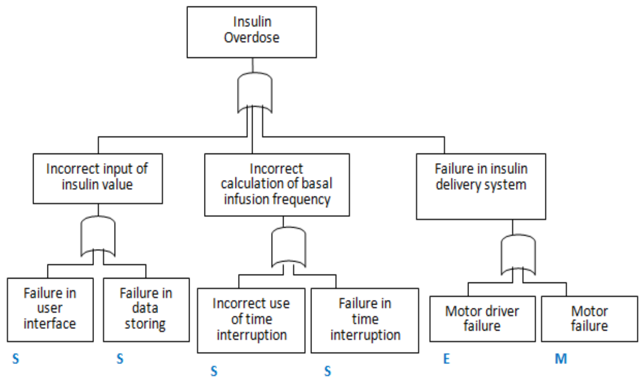
Insulin Overdose
Incorrect input of insulin value
Incorrect calculation of basal infusion frequency
Failure in insulin delivery system
Failure in user interface
Failure in data storing
Incorrect use of time interruption
Failure in time interruption
Motor driver failure
Motor failure
S
S
S
S
E
M

Fig. 2. Fault tree for the "insulin overdose" hazard with classification tags produced during the case study.

Fig. 2 shows the decomposition of "insulin overdose" into risk events using FTA technique. FTA is a top-down analysis starting from a hazard situation that should be decomposed into risk events. The number of decomposition levels depends on the complexity of the hazard situation analyzed. The fault tree presented in Fig. 2 is a partial view of the "insulin overdose" hazard, which was produced by the requirements engineer during the case study. We consider that it is enough to discuss the problem of safety functional requirements derivation.

The fault tree leaves represent possible causes that can put the system in hazard state. The intermediate levels of the fault tree represent groups of risk events and these levels are useful to organize the fault tree analysis. To improve the FTA understanding we put tags near the fault tree leaves. These tags identify if the failure cause is a software cause (S), an electronic cause (E), or a mechanical cause (M).

The development of a prototype of low cost insulin pump, which is a critical system, offered the basis to propose and test a protocol to derive safety functional requirements from FTA. The proposed protocol is discussed in section IV.

This protocol was performed to the "insulin overdose" hazard and 12 safety functional requirements were derived from the corresponding FTA. To validate the study a questionnaire was used which asked the difficulty to use FTA as a guide to implement safety features in the control software, and the improvement when FTA is accompanied with "should" and/or "should not" requirements.

The questionnaire was answered by two software engineers that participated in the case study. The results and analysis of that study are discussed in the next section.

IV. Results and Analysis

As stated in the previous section we chose the "insulin overdose" among ten relevant hazards identified in the context of insulin infusion pump to perform a protocol to help requirements engineer to derive safety functional requirements from FTA. The adopted protocol was divided into three steps:

1. Build FTA for hazard situation;
2. Classify fault tree leaves as software, electronic or mechanical failure causes;
3. For each failure cause (fault tree leaf) to write requirements descriptions in the style "should" and/or "should not" requirements.

A. Build FTA for hazard Situation

In the adopted protocol FTA assumed a central role to requirements engineer gets information to derive safety functional requirements. Therefore, FTA was carefully performed by the requirements engineer that participated in the case study.

The hazard that was chosen to be analyzed was directly mapped as the root in the fault tree. Starting from this point the requirements engineer used all the experience from the mechanical and electronic engineers as well the doctor experience to identify the risk events presented in Fig. 2. The goal was to find failure causes at the low level that put the system in risk condition and consequently could cause "insulin overdose" to the user.

We suggest organizing the fault tree at least in three decomposition levels: first level to identify the hazard to be analyzed; second level should be formed by labels that organize failure causes into groups; and third level to describe the failure causes.

Of course more levels can be created depends on the system complexity, but the levels between first and last are levels to organize failure causes into groups and not to describe them in details. As observed during this case study the failure causes were really identified at the last level in the fault tree.

Columns (1) and (2) presented in Table II were directly mapped from fault tree presented in Fig. 2. Column (1) corresponds to the second level of the fault tree and then the risk events identified by FTA were managed as high level safety requirements, which are: "incorrect input of insulin value", "incorrect calculation of basal infusion frequency" and "failure in insulin delivery system"; column (2) corresponds to the third level of the fault tree and identify the failure causes that can lead the system to cause insulin overdose to the user.

TABLE II. Safety Functional Requirements Derived from FTA.

		Safety Functional Requirements	
(1) Safety Requirements	**(2) Failure causes**	**(3) "Should" Requirements**	**(4) "Should Not" Requirements**
Incorrect input of insulin value	Failure in user interface (S)	1. The system ***should*** delimit insulin value range during the specification of basal infusion profile. 2. The input of insulin value to basal infusion profile specification should be done using up and down buttons.	1. The system ***should not*** allow the user to choose insulin value out of the safety specified range for basal infusion profile.
	Failure in data storing (S)	3. The system ***should*** store at least 3 basal infusion profiles, each one specifying 24 insulin values (one by hour).	
Incorrect calculation of basal infusion frequency	Incorrect use of time interruption (S)	4. The system ***should*** use the time interruption function offered by microcontroller to count time.	2. The system ***should not*** continue basal infusion if "watch dog" timer is executed more than 5 times by hour.
	Failure in time interruption (S)	5. The system ***should*** check if the time interruption is working as specified by user.	3. The system ***should not*** continue basal infusion if the accumulated error between real time and specified time is more than 0.1%.
Failure in insulin delivery system	Motor driver failure (E)	6. The system ***should*** implement acknowledge procedure to confirm the motor steppes performed.	4. Motor driver ***should not*** failure during *bolus* infusion.
	Motor failure (M)	7. The system ***should*** adopt a high precision motor to delivery insulin.	5. Motor ***should not* failure** during *bolus* infusion.

B. Classify Fault Tree Leaves as Software, Electronic or Mechanical Failure Causes

In safety-critical systems, particularly in embedded systems, usually there are three main types of failure causes: software, electronic and mechanical causes [15]. We consider that to improve the understanding of the safety requirements is very important to classify the failure causes according to these types.

Despite the FTA technique gives none suggestion about such classification we introduce this procedure as part of the adopted protocol because the recognition of the failure causes in a high level drives the investigation of the failures in a deeper way and consequently improves safety requirements specification. During the performed case study these classification helped the requirements engineer to conduct the safety requirements elicitation with the right stakeholders.

C. Deriving Safety Functional Requirements in Style "Should" and "Should Not" Requirements

Safety functional requirements are those requirements that describe what actions and/or constraints should or should not be performed to maintain the system in a safe state. During the case study for each failure cause identified by FTA safety functional requirements were written in style "should" and "should not". These requirements complemented FTA and offered more details to software engineers implement the software to control the insulin infusion pump.

Columns (3) and (4) in the Table II show "should" and "should not" requirements that were derived from the failure causes identified in the fault tree (Fig. 2). The failure causes guided the requirements engineer to conduct interviews with the stakeholders that participated in the study, i.e. the mechanical and electronic engineers and the doctor. Only failure causes described in fault tree do not offered sufficient information to derive safety functional requirements, but they proved to be a good base to begin the elicitation of the safety functional requirements with the stakeholders.

For instance, for the failure cause "failure in user interface" three safety functional requirements were described, of which two were "should" requirements and one was "should not". During the elicitation of these requirements the doctor experience in diabetes treatment using insulin infusion pump helped requirements engineer to understand the necessity of basal infusion profile specification which defines valid insulin values to the continuous infusion process.

It is worthwhile to note that we tried to produce, as much as possible, requirements description using "should" and "should not" style. This can cause some redundancy but to the safety-critical system this kind of redundancy helps software engineers to better understand the safety requirements of interest to implement the control software.

D. Results from the Questionnaire Applied During the Study

With the intention to validate the performed study and also to offer evidences to answer the research questions defined in the beginning of this study – RQ1 and RQ2 - we prepared a questionnaire with five questions related to the usage of FTA to derivate safety functional requirements. Two software engineers that participated in the study answered the following questions:

- ***Q1**: Do you believe that the fault tree offers complete and sufficient information to guide you during the implementation of the control software of the insulin infusion pump?*
- ***Q2**: Do you feel comfortable adopting fault tree as the main source to understand the safety requirements to implement the control software of the insulin infusion pump?*
- ***Q3**: Do you consider that the safety functional requirements presented in style "should" and "should not" are more adequate to drive you in the implementation of the control software of the insulin infusion pump than fault tree presented early?*
- ***Q4**: Do you think that "should" and "should not" requirements, being complementary in the safety functional requirements descriptions, improve the understanding of what should be implemented in the control software of the insulin infusion pump?*
- ***Q5**: As software engineer in charge to implement the control software of the insulin infusion pump, you prefer a software requirements document composed by:*
  - o *Only fault tree*
  - o *Only requirements in style "should" and "should not"*
  - o *Both representations*

To answer the question 1 to 4 the software engineers had to choose only one alternative available in style of Likert scale. The available alternatives were: totally agree, partially agree, partially disagree and totally disagree. The question 5 had three alternatives, as shown above *(Q5)*. Table III presents the questionnaire results.

The software engineers that answered the questionnaire did not participate in the confection of the fault tree nor the "should" and "should not" requirements. They had a general view about the insulin infusion pump functionalities obtained from the literature. The FTA was done by the requirements engineer that participated in the study, which interacted with the electronic and mechanical engineers as well as the doctor to elicit the safety requirements.

TABLE III. ANSWERS FROM THE SOFTWARE ENGINEERS THAT PARTICIPATED IN THE CASE STUDY.

Questions	**Answers from the Software Engineer *A***	**Answers from the Software Engineer *B***
Q1	Partially agree	Partially agree
Q2	Partially agree	Partially agree
Q3	Totally agree	Totally agree
Q4	Totally agree	Totally agree
Q5	Only requirements described in style "should" and "should not"	Fault tree and requirements described in style "should" and "should not"

The software engineers answered the questions *Q1* and *Q2* based on the fault tree presented in Fig. 2. Only after they answered these questions it was presented to them the requirements in style "should" and "should not", as presented in Table II. They answered *Q3* based on Table II, and finally they answered *Q4* and *Q5* considering the information available in Fig. 2 and Table II.

Although each software engineer answered the questionnaire alone, without communication between them, it is interesting to note that the answers to questions Q1 to Q4 were the same. Only the answers to Q5 were different, as can be observed in Table III. The questionnaire result is analyzed as follows.

Both software engineers partially agreed that fault tree offered complete and sufficient information to guide him during the implementation of the control software *(Q1)*. They also partially agreed to adopt fault tree as the main source to understand the safety requirements to implement the control software *(Q2)*.

However, when the safety functional requirements presented in style "should" and "should not" were shown to the software engineers, both of them totally agreed that requirements in that way are more adequate to drive them to implement the control software than fault tree *(Q3)*. The software engineers also totally agreed that requirements in style "should" and "should not" improve the understanding of what should be implemented in the control software *(Q4),* despite the redundancy that such style of description may bring.

Finally, when the software engineers were asked for preferences about the requirements document to guide them during control software implementation *(Q5)*, none of them chose requirements document composed by only fault tree. Software engineer *A* chose requirements document composed by only requirements described in style "should" and "should not", and software engineer *B* chose requirements document composed by both representations. These answers reveal that software engineers, in this case study, feel more comfortable using safety requirements described in style "should "and "should not" than FTA graphics representation.

E. Findings Emerged During the Application of the Adopted Protocol

In subsections *A, B* and *C* we reported the application of the proposed protocol to derive safety functional requirements to "insulin overdose" hazard. During the application of the protocol some findings emerged which are discussed as follows.

1) ***Finding****: Information available in FTA is not enough to derive safety functional requirements.*

This finding emerged during the study when the requirements engineer was trying to derive safety functional requirements from FTA. Despite the FTA to offer a good picture of the overall failure causes that can lead to analyzed hazard, when the requirements engineer was analyzing "insulin overdose" several important details to specify the safety requirements were missed. FTA offered a base to organize the safety requirements concerned to "insulin overdose", but the details to mitigate the failure causes, both related to constraints and functionalities, were not available in the fault tree.

The details were obtained from the stakeholder that participated in the study. For instance, the details to derive safety functional requirements to mitigate the failure cause *"Incorrect use of time interruption"* were not available in the fault tree. But this failure cause guided the requirements engineer to elicit the necessary information with the electronic engineer that participated in study, because he was the stakeholder that had the knowledge to derive the safety requirements to mitigate the failure cause under investigation. Thus, the requirement *"The system* ***should*** *use the time interruption function offered by microcontroller to count time"* – presented in Table II - only could be derived consulting the electronic engineer.

All the requirements presented in Table II, both in style "should" and "should not" were not directly available in the fault tree. To derive such requirements it was necessary to elicit them from the stakeholders that participated in the study. However, the failure causes identified using FTA were the drive to the elicitation process and helped the requirements engineer to derive the safety functional requirements to mitigate "insulin overdose". This finding helped us to answer the RQ2. In fact in the performed study the FTA representation was not enough to derive safety functional requirements, albeit it had offered a base to guide the elicitation process that helped requirements engineer to derive the requirements. This base was formed by all failure causes identified by FTA.

2) ***Finding****: The intermediate levels of fault tree help to organize safety requirements into groups.*

Despite the intermediate levels of fault tree have been created to organize the FTA until the analyzers discover the failure causes at the last level – fault tree leaves – during the case study we noted that the intermediate levels also can help requirements engineers to organize the understanding about the analyzed hazard into groups of safety requirements.

The second level of the fault tree created during the case study – shown in Fig. 2 - helped the requirements engineer to organize the safety requirements concerned to "insulin overdose". The safety requirements were organized into three groups, as shown in column (1) - Table II. The safety functional requirements were derived – columns (3) and (4) - following the organization defined by FTA, thus all "should" and "should not" requirements already had a group to be accommodated during the elicitation process.

3) ***Finding****: The classification of the failure causes helps during the safety functional requirements elicitation.*

During the study we include a new procedure when FTA was being performed. This procedure was to classify the failure causes as software, electronic or mechanical causes. The tags (S), (E) and (M) close to the failure causes in Fig. 2 show the classification done during FTA.

This classification helped requirements engineer during the elicitation process to derive safety functional requirements. For instance, the failure causes classified as software or electronic failures drove the requirements engineer to elicit safety requirements with the electronic engineer, which had experience both in software and electronic development. Mechanical failures drove the requirements engineer to elicit safety requirements with the mechanical engineer.

4) ***Finding**: Requirements descriptions in style "should" and "should not" help software engineer to better understand the system safety requirements.*

During the elicitation process we decided to derive the safety functional requirements in the style "should" and "should not" requirements. Such decision was based on two reasons: (1) it is a very common style to describe requirements and well known in the Requirements Engineering community; (2) we believe that "should" requirements could be complemented by "should not" requirements in many situations improving the general understanding about the safety requirements.

The answers from the software engineers that participated in the case study supported our assumption, as we can observe from the *Q4* results in the applied questionnaire. Even if some redundancy is present in the requirements it is accepted because the improvement in the understandability compensates the redundancy.

V. Conclusion

This paper presented results from a case study performed in a Brazilian company in the area of the electronic medical devices. During the case study a protocol to derive safety functional requirements was performed in the context of a prototype development of a low cost insulin infusion pump.

The findings that emerged during the case study and the answers from the software engineers that answered the questionnaire brought some evidences that helped us to answer the researches questions under investigation – RQ1 and RQ2.

Answering RQ1, we can say that based on the evidences from the performed case study FTA does not offer enough information to drive software engineers to implement safety requirements. According to the questionnaire results the software engineers did not feel safe to use only fault tree to understanding what safety requirements should be implemented in the control software. They considered that safety requirements described in style "should" or "should not" are more adequate than fault tree representation, or if fault tree is used it should come accompanied by safety functional requirements in style "should" or "should not".

Answering RQ2, we can say that based on the evidences from the performed case study the fault tree resulting from the FTA is not a good enough representation to derive safety functional requirements, i.e. the information present in the fault tree is not sufficient to derive safety functional requirements. Fault tree representation proved to be very useful to organize safety requirements into groups, but it was necessary to appeal to the stakeholders to get necessary information to derive safety functional requirements, because only fault tree didn't supply the requirements engineer with necessary information to write safety functional requirements.

Despite the FTA is a well known technique used to analyze risks in safety-critical systems, we didn't find in the literature any formal protocol to derive safety functional requirements from FTA. We believe that such gap difficult requirements engineers to produce high quality safety requirements specification when FTA is adopted. The proposed protocol adopted in the performed case study to derive safety functional requirements of course is not yet a formal protocol, but it can be improved to become one. Nevertheless, the proposed protocol in the performed case study helped the requirements engineer to derive safety functional requirements from FTA to the "insulin overdose" hazard.

In the context of researches in safety-critical systems we point out the following future work:

- To use the proposed protocol to derive safety functional requirements for other hazards in the context of the insulin infusion pump, which is under development in collaboration with a Brazilian company;
- To perform an evaluation of the adopted protocol inviting more requirements engineers to using of such protocol in other safety-critical systems;
- To create formal guidelines to help requirements engineers when using the adopted protocol to derive safety functional requirements from FTA;
- To replicate the performed case study in other safety-critical domains, e.g. in automotive and avionics domains.

References

[1] D. Firesmith, "Engineering Safety- and Security-Related Requirements for Software-Intensive Systems", 32nd IEEE International Conference on Software Engineering, One-Day Tutorial (ICSE'2010). South Africa, May 2010.

[2] P. Liggesmeyer and M. Trapp, "Trends in Embedded Software Engineering", IEEE Software Magazine, v. 26, n. 3, 2009, pp. 19-25.

[3] T. Sanislav, and L. Miclea, "Cyber-Physical Systems - Concept, Challenges and Research Areas", Journal of Control Engineering and Applied Informatics (CEAI), vol. 14, 2012, pp. 28–33.

[4] I. Sommerville, Software Engineering. Addison-Wesley, 9th edition, 2010.

[5] B. S. Medikonda and S. R. Panchumarthy, "A Framework for Software Safety in Safety-Critical Systems", SIGSOFT Software Engineering Notes, vol. 34, n. 2. March 2009. pp. 1-9.

[6] D. Firesmith, "Engineering Safety Requirements, Safety Constraints, and Safety-Critical Requirements", Journal of Object Technology, vol. 3, n. 3, march/april 2004.

[7] Y. Zhang, P. L. Jones and R. A. Jetley, "Hazard Analysis for a Generic Insulin Infusion Pump", Journal of Diabetes Science and Technology, vol. 4, n. 2, March/2010. pp. 263-283.

[8] S. Yaturu, "Insulin Therapies: Current and future trends at dawn", World Journal of Diabetes, vol. 4, n. 1, February/2013. pp. 1-7.

[9] S. Kumari, G. Kondeti, S. Pakki, T. L. V. Chandrasekhar and S. Balu, "Method of Safety Critical Requirements Flow in Product Life Cycle Processes", International Conference on Integrated Communications, Navigation and Surveillance Conference (ICNS), May/2011. pp. N2-1 – N2-4.

[10] D. Gollub, M. Kretschmer and A. Heyl, "Defining Safety Requirements for Hybrid Electric Vehicles Using System Simulation", 3rd IET International Conference on System Safety, Oct/2008. pp. 1-5.

[11] S. B. Driskell, J. Murphy, J. B. Michael and Man-Tak Shing "Independent validation of software safety requirements for systems of systems", 5th International Conference on System of Systems Engineering (SoSE), June/2010. pp. 1-6.

[12] M. J. Squair, "Issues in the application of Software Safety Standards", 10th Australian Workshop on Safety Related Programmable Systems (SCS), vol. 55. Sidney, 2005. pp. 13-26.

[13] E. Sikora, B. Tenbergen and K. Pohl, "Industry Needs and Research Directions in Requirements Engineering for Embedded Systems", Requirements Engineering Journal, vol. 17, n. 1, 2012. pp. 57-78.

[14] E. Navarro, P. Sánchez, P. Letelier, J. A. Pastor and I. A. Ramos, "Goal-Oriented Approach for Safety Requirements Specification", Proceedings of the 13th IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS), 2006. pp. 318-326.

[15] F. Vahid, T. Givargis, Embedded System Design: A Unified Hardware/Software Design. John Willey & Sons, 2002.

[16] R. R. Lutz, A. Patterson-Hine, C. R. Frost, S. Nelson, D. Tal and R. Harris, "Using Obstacle Analysis to Identify Contingency Requirements on an Unpiloted Aerial Vehicle", Requirements Engineering Journal, vol. 12, n. 1, 2007. pp. 41-54.

[17] F. Modugno, N. G. Leveson, J. D. Reese, K. Partridge and S. D. Sandys, "Integrated Safety Analysis of Requirements Specifications", Requirements Engineering Journal, vol. 2, n.1, 1997. pp. 65-78.

[18] E. J. Broomfield and P. W. H. Chung, "Safety Assessment and the Software Requirements Specification", Journal of Reliability Engineering and System Safety, Elsevier. Vol. 55, n. 3, 1997. pp. 295-309.

[19] R. R. Lutz and I. C. Mikulski, "Operational Anomalies as a Cause of Safety-Critical Requirements Evolution", Journal of Systems and Software, Elsevier, vol. 65, n. 2, 2003. pp. 155-161.

[20] R. R. Lutz and I. C. Mikulski, "Empirical Analysis of Safety-Critical Anomalies During Operations", IEEE Transactions on Software Engineering, vol. 30, n. 3, 2004. pp. 172-180.